UNICOS® Resource Administration

S–2302–10008

New Features

UNICOS® Resource Administration S–2302–10008

This book supports the 10.0.0.8 release of the UNICOS operating system. The following changes to the resource administration documentation were made for this release:

Section 5.12.1 (System Startup)	Corrected `rc.mid` and `qurun` example scripts.
Section 7.3.3.9 (Disk Data)	Removed all physical disk data names that had a `pd_` prefix from the IOS-E Physical Disk Data table.
Section 7.3.3.14 (Network Interface Data)	Changed the prefixes for the network interface data names from `net_` to `nw_`. See the Network Interface Data table.
Other changes	Minor text changes.

Record of Revision

Version	*Description*
9.0	August 1995 Original Printing. Documentation supports the administration of UNICOS 9.0 release running on Cray computer systems. This manual contains the contents of and supersedes the information formerly provided in chapters “Accounting,” “Automated Incident Reporting (AIR),” “Fair-share Scheduler,” File System Quotas,” “File System Space Monitoring,” “System Activity Monitoring,” “Unified Resource Manager (URM)” and appendix “Automatic Incident Reporting Tests” in UNICOS System Administration, publication SG-2113 8.0.
9.2	December 1996 This rewrite supports the 9.2 release of the UNICOS operating system.
9.3	August 1997 This rewrite supports the 9.3 release of the UNICOS operating system.
10.0	November 1997 This rewrite supports the 10.0 release of the UNICOS operating system.
10.0.0.2	May 1998 This rewrite supports the 10.0.0.2 release of the UNICOS operating system.
10.0.0.3	September 1998 This rewrite supports the 10.0.0.3 release of the UNICOS operating system.
10008	November 2000 This rewrite supports the 10.0.0.8 release of the UNICOS operating system.

Contents

Page

Page

Page

Page

Preface

This manual documents UNICOS release 10.0.0.8 running on Cray systems. It contains information needed in the administration of various UNICOS features available to all UNICOS systems.

> **Note:** Previously in the UNICOS operating system documentation, the term *Trusted UNICOS* was used to refer to the configuration that most closely approximated the B1 evaluated configuration of UNICOS release 8.0.2. In the current UNICOS release, this configuration is referred to as the *Cray ML-Safe configuration* of the UNICOS operating system. Although the Cray ML-Safe configuration of the UNICOS operating system is not an evaluated product, this configuration fully supports all functionality described in the B1 evaluation criteria.

UNICOS System Administration Publications

This guide is one of a set of related manuals that cover information on the structure and operation of a Cray computer system running the UNICOS operating system, as well as information on administering various products that run under the UNICOS operating system. This set includes the following documents:

- *General UNICOS System Administration*
- *UNICOS Resource Administration*
- *UNICOS Configuration Administrator's Guide*
- *UNICOS Networking Facilities Administrator's Guide*
- *UNICOS NQS and NQE Administrator's Guide*
- *Kerberos Administrator's Guide*
- *Tape Subsystem Administration*

General UNICOS System Administration, contains information on performing basic administration tasks as well as information about system and security administration using the UNICOS multilevel (MLS) feature. This publication contains chapters documenting file system planning, UNICOS startup and shutdown procedures, file system maintenance, basic administration tools, crash and dump analysis, the UNICOS multilevel security (MLS) feature, and administration of online features.

UNICOS Resource Administration, contains information on the administration of various UNICOS features available to all UNICOS systems. This publication contains chapters documenting accounting, automatic incident reporting (AIR), the fair-share scheduler, file system quotas, file system space monitoring, system activity and performance monitoring, and the Unified Resource Manager (URM).

UNICOS Configuration Administrator's Guide, provides details about UNICOS configuration files created when the UNICOS operating system is installed and configured.

UNICOS Networking Facilities Administrator's Guide, contains information on administration of networking facilities supported by the UNICOS operating system. This publication contains chapters documenting TCP/IP for the UNICOS operating system, the UNICOS network file system (NFS) feature, the network information system (NIS) feature, and the Cray-based network monitor.

UNICOS NQS and NQE Administrator's Guide, contains information on administration of the Network Queuing System (NQS) and the Network Queuing Environment (NQE) features.

Kerberos Administrator's Guide, contains information on administration of the Kerberos feature, a set of programs and libraries that provide distributed authentication over an open network. This publication contains chapters documenting Kerberos implementation, configuration, and troubleshooting.

Tape Subsystem Administration, contains information on administration of UNICOS and UNICOS/mk tape subsystems. This publication contains chapters documenting tape subsystem administration commands, tape configuration, and tape troubleshooting.

Related Publications

For detailed information on UNICOS installation procedures, see the following Cray Inc. publications:

- *UNICOS Installation Guide for Cray J90se and Cray SV1 GigaRing based Systems*
- *UNICOS Installation Guide for Cray C90, Cray T90, and Cray T90 IEEE Model E based Systems*
- *UNICOS Installation Guide for Cray J90, Cray J90se, and Cray SV1 Model V based Systems*

- *UNICOS Installation Guide for Cray T90 and Cray T90 IEEE GigaRing based Systems*
- *UNICOS System Configuration Using ICMS*

For additional information about the UNICOS operating system and its features, see the following Cray publications:

- *UNICOS Administrator Commands Reference Manual*
- *UNICOS User Commands Reference ManualCray C++ Compiler, C++ MathPack, and C++ Tools for MPP Release Letter*
- *Application Programmer's I/O Guide*
- *UNICOS Basic Administration Guide for Cray J90, Cray J90se, and Cray SV1 Model V based Systems*
- *Tape Subsystem User's Guide*
- *UNICOS Source Manager (USM) User's Guide*
- *UNICOS Multilevel Security (MLS) Feature User's Guide*
- *Cray Message System Programmer's Guide*
- *Cray/REELlibrarian (CRL) Administrator's Guide*
- *Cray Data Migration Facility (DMF) Administrator's Guide*
- *Cray Data Migration Facility (DMF) Release and Installation Guide*
- *Cray Data Migration Facility (DMF) MSP Writer's Guide*
- *FTA User and Administrator Manual*
- *OLNET Online Diagnostic Network Communications Program Maintenance Manual for UNICOS*. Cray PRIVATE. This document contains information private to Cray, Inc. It can be distributed to non-Cray personnel only with approval of the appropriate Cray manager.

For more information about the system workstation (SWS), see the following Cray publications:

- *SWS-ION Administration and Operations Guide*
- *SWS Solaris Operating System and Devices Installation Guide*
- *SWS-ION Release Overview*

For more information about the I/O subsystem model E (IOS-E) and the support system (OWS-E and OWS-T), see the following Cray publications:

- *I/O Subsystem Model E (IOS-E) Support Guide*. Cray RESEARCH PRIVATE. This document contains information private to Cray, Inc. It can be distributed to non-Cray personnel only with approval of the appropriate Cray manager.
- *Support System Reference Manual*
- *Support System Operator's Guide*
- *Support System Administrator's Guide*
- *Support System Ready Reference*
- *Support System and IOS-E Release Overview*

For more information about the networking and communications software supported under the UNICOS operating system (and referenced in this manual), see the following Cray publications:

- *TCP/IP Network User's Guide*
- *Network Queuing System (NQS) User's Guide*
- *OSI Administrator's Guide*
- *Kerberos User's Guide*
- *Remote Procedure Call (RPC) Reference Manual*

Design specifications for the UNICOS multilevel security (MLS) feature are based on the trusted computer system evaluation criteria developed by the U. S. Department of Defense (DoD). If you require more information about multilevel security on UNICOS, you may find the following sources helpful:

- DoD Computer Security Center. *A Guide to Understanding Trusted Facility Management* (DoD NCSC-TG-015). Fort George G. Meade, Maryland: 1989.
- DoD Computer Security Center.*Department of Defense Trusted Computer System Evaluation Criteria* (DoD 5200.28-STD). Fort George G. Meade, Maryland: 1985. (Also known as the *Orange book*.)
- DoD Computer Security Center. *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria* (DoD NCSC-TG-005-STD). Fort George G. Meade, Maryland: 1987. (Also known as the *Red book*.)

- DoD Computer Security Center.*Summary of Changes, Memorandum for the Record* (DoD 5200.28-STD). Fort George G. Meade, Maryland: 1986.
- DoD Computer Security Center.*Password Management Guidelines*(CSC-STD-002-85). Fort George G. Meade, Maryland: 1985.
- Wood, Patrick H. and Stephen G. Kochan. *UNIX System Security*. Hasbrouck Heights, N.J.: Hayden Book Company, 1985.

  Note: If your site wants to purchase the optional SecurID card used with UNICOS MLS network security, the necessary hardware, software, and user publications can be obtained from Security Dynamics, Inc., 2067 Massachusetts Avenue, Cambridge, MA, 02140, (617) 547-7820.

Obtaining Publications

The *User Publications Catalog* describes the availability and content of all Cray hardware and software documents that are available to customers. Customers who subscribe to the Cray Inform (CRInform) program can access this information on the CRInform system.

To order a document, call +1–651–605–9100. Cray employees may send e-mail to `orderdsk@cray.com`

Customers who subscribe to the CRInform program can order software release packages electronically by using the `Order Cray Software` option.

Customers outside of the United States and Canada should contact their local service organization for ordering and documentation information.

Conventions

The following conventions are used throughout this document:

Convention	Meaning
`command`	This fixed-space font denotes literal items (such as commands, files, routines, pathnames, signals, messages, programming language structures, and e-mail addresses) and items that appear on the screen.
`manpage`(*x*)	Man page section identifiers appear in parentheses after man page names. The following list describes the identifiers:

1	User commands
1B	User commands ported from BSD
2	System calls
3	Library routines, macros, and opdefs
4	Devices (special files)
4P	Protocols
5	File formats
7	Miscellaneous topics
7D	DWB-related information
8	Administrator commands

Some internal routines (for example, the `_assign_asgcmd_info`() routine) do not have man pages associated with them.

variable	Italic typeface denotes variable entries and words or concepts being defined.
`user input`	This bold, fixed-space font denotes literal items that the user enters in interactive sessions. Output is shown in nonbold, fixed-space font.
[ ]	Brackets enclose optional portions of a command or directive line.

...	Ellipses indicate that a preceding element can be repeated.

The following machine naming conventions may be used throughout this document:

Term	Definition
Cray PVP systems	All configurations of Cray parallel vector processing (PVP) systems.

The default shell in the UNICOS and UNICOS/mk operating systems, referred to as the *standard shell*, is a version of the Korn shell that conforms to the following standards:

- Institute of Electrical and Electronics Engineers (IEEE) Portable Operating System Interface (POSIX) Standard 1003.2–1992
- X/Open Portability Guide, Issue 4 (XPG4)

The UNICOS and UNICOS/mk operating systems also support the optional use of the C shell.

Cray UNICOS Version 10.0 is an X/Open Base 95 branded product.

Reader Comments

If you have comments about the technical accuracy, content, or organization of this document, please tell us. Be sure to include the title and number of the document with your comments.

You can contact us in any of the following ways:

- Send e-mail to the following address:

  `pubs@cray.com`

- Send a fax to the attention of “Software Publications” at: +1–651–605–9001.
- File an SPR; use `PUBLICATIONS` for the group name, `PUBS` for the command, and `NO-LICENSE` for the release name.
- Call the Software Publications Group, through the Cray Support Center, using one of the following numbers: 1–800–950–2729 (toll free from the United States and Canada) or +1–651–605–8805.
- Send mail to the following address:

Software Publications
Cray Inc.
1340 Mendota Heights Road
Mendota Heights, MN 55120

We value your comments and will respond to them promptly.

Introduction to System Administration [1]

This guide is a teaching and reference document for people who manage the operation of Cray computer systems running the UNICOS operating system. It contains information needed in the administration of various UNICOS features available to all UNICOS systems.

This manual provides information on the use and administration of the following UNICOS products:

- Accounting: Cray system accounting (CSA) and standard UNIX accounting.
- Automatic incident reporting (AIR), a UNICOS facility that allows you to automate the monitoring of UNICOS components, such as online tapes, Network Queuing System (NQS), Transmission Control Protocol/Internet Protocol (TCP/IP), and the kernel. This facility allows you to monitor existence, ability to respond, and ability to function. This manual also contains an appendix that provides descriptions of the tests available with the automatic incident reporting (AIR) feature.
- Fair-share scheduler (also referred to as *fair-share*), a UNICOS facility that provides resource control and allows a machine to be shared among groups in an organized fashion.
- File system quotas, a UNICOS feature that allows you to control the amount of file system space in blocks and the number of files used by each account, group, and user on an individual basis.
- File system monitor, a UNICOS feature that improves the usability and reliability of the system by observing the amount of free space on the mounted file systems and taking remedial action if warning or critical thresholds are reached.
- System activity and performance monitoring, including real-time performance monitoring with the `sam` utility or the standard UNIX system activity package, and disk use monitoring.
- Unified Resource Manager (URM), a job scheduler that balances the demands of both batch and interactive sessions.

This guide replaces neither experience nor other documents that more fully describe specific system areas. Familiarity with the references listed in the preface is necessary to effectively manage Cray computer systems running UNICOS.

Accounting [2]

The UNICOS operating system supports two types of accounting: Cray system accounting and standard UNIX accounting. Both types of accounting are described in this chapter.

2.1 Cray System Accounting (CSA)

Cray system accounting (CSA) is designed to meet the unique accounting requirements of Cray sites. Like the standard UNIX accounting package, CSA provides methods to collect per-process resource utilization data, record connect sessions, monitor disk usage, and charge fees to specific logins. CSA also provides other facilities that are not available from the standard accounting package. These include the following:

- Per-job accounting
- Accounting for socket usage
- Device accounting
- Daemon accounting (for the Network Queuing System (NQS) and the UNICOS tape subsystem)
- Disk accounting by account ID
- Arbitrary accounting periods
- Flexible system billing unit (SBU) system
- One file containing all data for an accounting period
- Off-line archiving of accounting data

Sites may run either the standard UNICOS accounting programs or the CSA package by invoking the appropriate shell scripts and programs. Both packages are installed with the UNICOS 10.0 release.

UNICOS system features in the CSA package include configurable parameters located in a single file, `/etc/config/acct_config`, and a set of user-defined exits that allows sites to tailor the daily run of accounting to their specific needs.

2.1.1 Concepts and Terminology

The following concepts and terms are important in CSA:

Term	Description
Daily accounting	Unlike the standard daily accounting, CSA's accounting can be run as many times as necessary during a day. However, this feature is still referred to as *daily accounting*.
Periodic accounting	Accounting similar to the standard UNICOS monthly accounting. CSA, however, lets system administrators specify the time periods for which "monthly" or cumulative accounting is to be run. Thus, periodic accounting can be run more than once a month.
Recycled data	By default, accounting data for active sessions is recycled until the session terminates. CSA reports only data for terminated sessions unless `csarun`(8) is invoked with the `-A` option. `csarun` places recycled data into data files in the `/usr/adm/acct/day` directory. These data files are suffixed with `0`; for example, per-process accounting data for active sessions from previous accounting periods is in the `/usr/adm/acct/day/pacct0` file.
Session	CSA organizes accounting data by sessions and boot times and then places the data into a session record file. For non-NQS jobs, a *session* consists of all accounting data for a given job ID during a single boot period. A *session* for an NQS job consists of the accounting data for all job IDs associated with the job's NQS sequence number/machine name identifier. NQS jobs may span multiple boot periods. If a job is restarted, it has the same job ID associated with it during all boot periods in which it runs. Rerun NQS jobs have multiple job IDs. CSA treats all phases of an NQS job as being in the same session.

Uptime period or boot period	A period delineated by the system boot times found in `/etc/csainfo`. The `csaboots`(8) command writes to this file during system boot.

2.1.2 Files and Directories Overview

This section provides a brief overview of the CSA file and directory structure. A more complete description of the files and directories can be found in Section 2.1.7, page 23.

2.1.2.1 Structures of the `acct` and `tmp` Directories

The directory structure of `/usr/adm/acct` is set up so that it is easy to find CSA data files and reports. The `/tmp` structure is also used while `csarun`(8) is running. Figure 1 illustrates the directory structure for both directories.

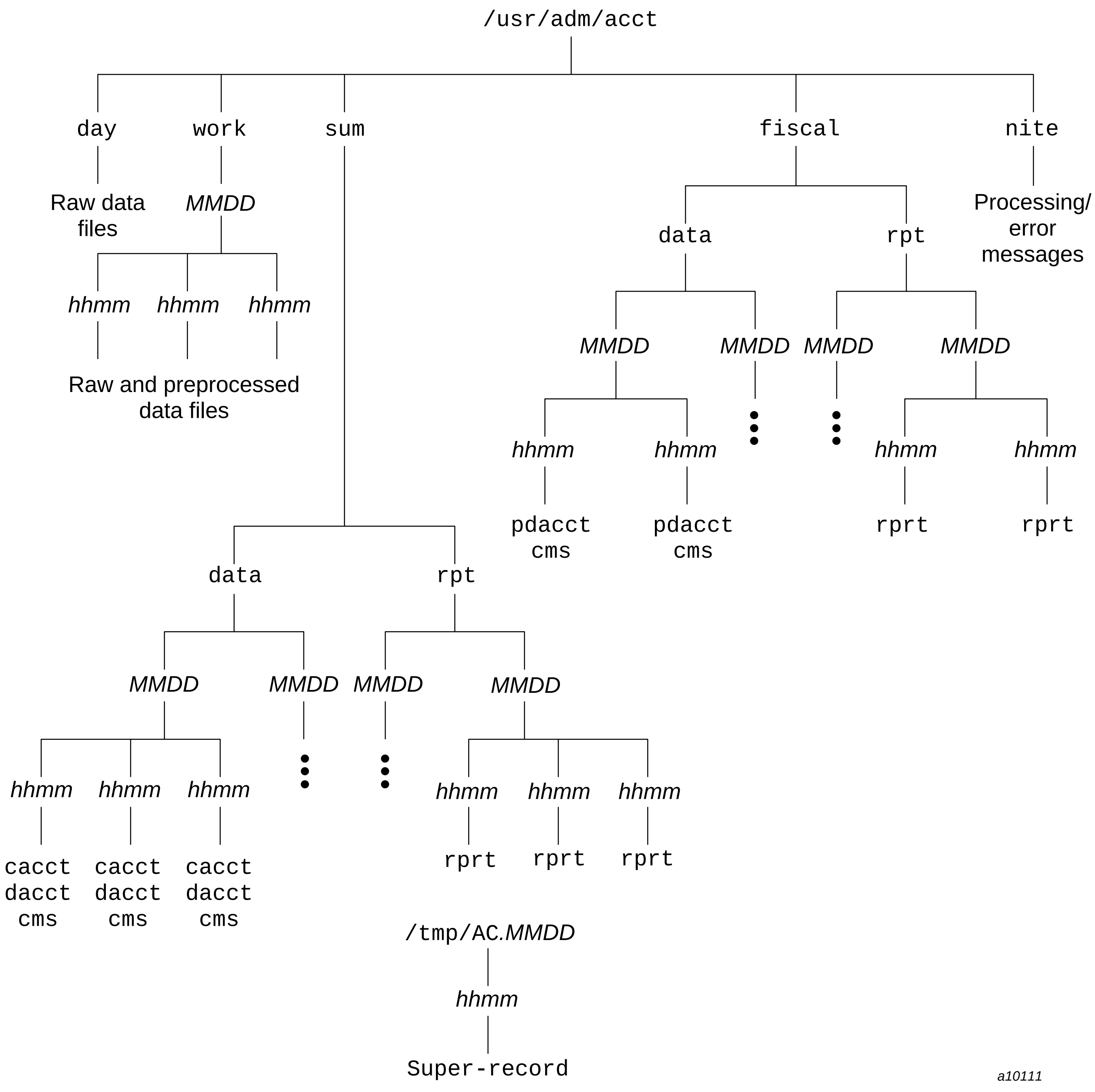


Figure 1. `/usr/adm/acct` and `tmp` Directory Structures

Note: As distributed, only the directory `/usr/adm/acct/day` is readable by all users. Within the `day` directory, only the `pacct*` files are readable by all users. This allows any user to examine the `pacct*` files by using the `acctcom`(1) command. All other directories and files within `/usr/adm/acct` are accessible only by `root` and users in the group `adm`.

Warning: `acctcom`(1) on a Cray ML-Safe configuration of the UNICOS system is considered to be a covert channel. You may want to consider restricting access to this command to the `adm` group.

The following abbreviations have these meanings:

Abbreviation	Definition
MMDD	Month, day
hhmm	Hour, minute

2.1.2.2 Shell Scripts and C Binaries

The `/usr/lib/acct` directory contains virtually all of the programs and scripts used by both the standard accounting and CSA packages. The only CSA program not located here is `/etc/csaboots` (see `csaboots`(8)), which records boot times at system startup. Programs used only by CSA begin with the characters `csa`.

2.1.2.3 Unprocessed Data Files

Both CSA and the standard accounting package expect most unprocessed accounting files to be located in the `/usr/adm/acct/day` directory. The use of this directory simplifies tracking of the current accounting files. The following table shows the location of the raw data files.

Accounting file	Description
`/usr/adm/acct/day/dtmp`	Disk accounting data
`/usr/adm/acct/day/nqacct*`	NQS daemon accounting data
`/usr/adm/acct/day/pacct*`	Per-process accounting data
`/usr/adm/acct/day/tpacct*`	Tape daemon accounting data

`/usr/adm/acct/day/soacct*`	Socket accounting data
`/etc/csainfo`	Boot times
`/etc/wtmp`	Connect time accounting data

Warning: On a Cray ML-Safe configuration of the UNICOS system, `/etc/wtmp` is considered a covert channel. You may want to consider restricting access to this file to the `adm` group.

Accounting files in `/usr/adm/acct/day` whose names include the suffix `0` contain data from sessions that did not complete during the previous accounting periods.

During CSA data processing, sites may select to archive the raw and/or processed data off-line. Section 2.1.5, page 16, describes how to do this. By default, all raw data files are deleted after use and are not archived.

2.1.2.4 Data Files Being Processed

At the start of a daily accounting run, CSA moves the raw data files from `/usr/adm/acct/day` to the appropriate `/usr/adm/acct/work/`*MMDD/hhmm* directory. The files in the work directory are as follows:

File	Description
`Ever.tmp`	Data verification work file
`Pctime*`	Preprocessed connect time data
`Pnqacct*`	Preprocessed NQS data
`Puptime*`	Uptimes
`Rctime0`	Connect data to be recycled in the next accounting run
`Rnqacct0`	NQS data to be recycled in the next accounting run
`Rpacct0`	Per-process accounting data to be recycled in the next accounting run
`Rtpacct0`	Tape data to be recycled in the next accounting run
`Ruptime0`	Uptimes to be recycled in the next accounting run

`Wctime*`	Verified raw connect time data
`Wdisktacct`	Disk accounting data (`cacct.h` format)
`Wdtmp`	Disk accounting data from `diskusg`(8) or `acctdusg`(8)
`Wnqacct*`	Raw NQS accounting data
`Wpacct*`	Raw per-process accounting data
`Wsoacct*`	Raw socket accounting data
`Wtpacct*`	Raw tape accounting data
`Wwtmp`	Raw connect time data

2.1.2.5 Processed Data Files

CSA outputs the following data files:

File	Description
`/tmp/AC.`*MMDD*/*hhmm*`/Super-record`	Session record file; this file is usually deleted after it has been used by CSA.
`/usr/adm/acct/fiscal/data/`*MMDD*/*hhmm*`/pdacct`	Consolidated periodic data.
`/usr/adm/acct/fiscal/data/`*MMDD*/*hhmm*`/cms`	Periodic command usage data.
`/usr/adm/acct/sum/data/`*MMDD*/*hhmm*`/cacct`	Consolidated daily data; this file is deleted by `csaperiod`(8) if the `-r` option is specified.
`/usr/adm/acct/sum/data/`*MMDD*/*hhmm*`/cms`	Daily command usage data; this file is deleted by `csaperiod`(8) if the `-r` option is specified.

`/usr/adm/acct/sum/data/`*MMDD*`/`*hhmm*`/dacct`

Daily disk usage data; this file is deleted by `csaperiod`(8) if the `-r` option is specified.

2.1.2.6 Reports

CSA generates daily and periodic reports. The locations of these reports are as follows:

File	Description
`/usr/adm/acct/fiscal/rpt/`*MMDD*`/`*hhmm*`/rprt`	Periodic accounting report
`/usr/adm/acct/sum/rpt/`*MMDD*`/`*hhmm*`/rprt`	Daily accounting report

2.1.3 Daily Operation Overview

When the UNICOS operating system is run in multiuser mode, accounting behaves in a manner similar to the following process. However, because sites may customize CSA, the following may not reflect the actual process at a particular site:

1. System boot time is written to `/etc/csainfo`. Each time the system is booted, the boot time is written to `/etc/csainfo` by the `/etc/csaboots` command, which is invoked by `rc` (see `brc`(8)) during system startup.
2. Process accounting is enabled. When the system is switched to multiuser mode, the `/usr/lib/acct/startup` (see `acctsh`(8)) script is called by `/etc/rc` and performs the following functions:

   a. Writes an `acctg on` record to `/etc/wtmp`; the `acctwtmp` program is used to write this record.

   b. Enables process accounting with the command line `/usr/lib/acct/turnacct on`; `turnacct`(8) calls the `accton` program with the argument `/usr/adm/acct/day/pacct`.

   c. Removes lock files and saved `pacct` and `wtmp` files. `/usr/lib/acct/remove` is invoked to clean up saved `pacct` and `wtmp` files in `/usr/adm/acct/sum`. Unlike the standard accounting package, CSA does not leave files in this directory. In addition, the lock files are removed from `/usr/adm/acct/nite`.

3. By default, daemon accounting for NQS, tape, and sockets is handled by the `/usr/lib/acct/startup` script. However, in order to run NQS and tape daemon accounting, you must modify the appropriate subsystem. Section 2.1.4, page 11, describes this process in detail.
4. The amount of disk space used by each user is determined periodically. `/usr/lib/acct/dodisk` (see `dodisk`(8)) is run periodically by `cron` to generate a snapshot of the amount of disk space being used by each user. `dodisk` should be run at most once for each time `/usr/lib/acct/csarun` (see `csarun`(8)) is run. Multiple invocations of `dodisk` during the same accounting period write over previous `dodisk` output.
5. A fee file is created. Sites desiring to charge fees to certain users can do so by invoking `/usr/lib/acct/chargefee` (see `chargefee`(8)). Each accounting period's fee file (`/usr/adm/acct/day/fee`) is merged into the consolidated accounting records by `/usr/lib/acct/csaperiod` (see `csaperiod`(8)).
6. Daily accounting is run. At specified times during the day, `csarun` is executed by `cron` to process the current accounting data. The output from `csarun` is a consolidated daily accounting file and an ASCII report.
7. Periodic accounting is run. At a specific time during the day, or on certain days of the month, `/usr/lib/acct/csaperiod` (see `csaperiod`(8)) is executed by `cron` to process consolidated accounting data from previous accounting periods. The output from `csaperiod` is a consolidated periodic accounting file and an ASCII report.
8. Accounting is disabled. When the system is shut down gracefully, the script `/usr/lib/acct/shutacct` (see `shutacct`(8)) is executed by `/etc/shutdown` (see `shutdown`(8)). `shutacct` writes an "acctg off" record to `/etc/wtmp`. It then calls `/usr/lib/acct/turnacct` and `/usr/lib/acct/turndacct` to disable per-process and daemon accounting (see `turnacct`(8) and `turndacct`(8)).

2.1.4 Setting up CSA

The following is a brief description of setting up CSA. Site-specific modifications are discussed in detail in Section 2.1.10, page 39. As described in this section, CSA is run by a person with super-user permissions. CSA also can be run by users who have acct permissions and are in the adm group. See Section 2.1.10.7, page 54, for the necessary modifications.

1. Change the default system billing unit (SBU) weighting factors, if necessary. By default, no SBUs are calculated. If your site wants to report SBUs, you must modify the configuration file /etc/config/acct_config.

2. Modify any necessary parameters in the /etc/config/acct_config file, which contains configurable parameters for the accounting system. Ensure that parameters, such as MEMINT, reflect the needs of your site.

3. If you want daemon accounting, you must enable daemon accounting at system startup time by performing the following steps:

   a. Ensure that the variables in /etc/config/acct_config for the subsystems for which you want to enable daemon accounting are set to on. Set the NQS_START, TAPE_START, and SOCKET_START parameters to on to enable NQS, online tapes, and socket accounting, respectively.

   b. If necessary, enable accounting from the daemon's side. Specifically, NQS and tape accounting must also be enabled by the associated daemon. Use the qmgr(8) set accounting on command to turn on NQS accounting. To enable tape daemon accounting, execute tpdaemon(8) with the -c option. Socket accounting does not require any additional processing.

4. Prior to setting up the following cron jobs, ensure that the /etc/checklist file exists. By default, dodisk(8) performs disk accounting on the special files listed in checklist. For most installations, entries similar to the following should be made in /usr/spool/cron/crontabs/root so that cron(8) automatically runs daily accounting:

```
0 4 * * 1-6 /usr/lib/acct/csarun 2> /usr/adm/acct/nite/fd2log
0 3 * * 1-6 /usr/lib/acct/dodisk -a -v 2> /usr/adm/acct/nite/dk2log
```

   csarun(8) should be executed at such a time that dodisk has sufficient time to complete. If dodisk does not complete before csarun executes, disk accounting information may be missing or incomplete.

   dodisk must be invoked with either the -a or the -A option. If it is not, csaperiod(8) aborts when it attempts to merge the disk usage information with other accounting data.

5. Periodically check the size of the acct files. Entries similar to the following should be made in /usr/spool/cron/crontabs/root:

   ```
   0 * * * * /usr/lib/acct/ckdacct nqs tape socket
   0 * * * * /usr/lib/acct/ckpacct
   ```

   cron(8) should periodically execute the ckpacct(8) and ckdacct(8) shell scripts. If the pacct file grows larger than 500 blocks (default), ckpacct calls the command /usr/lib/acct/turnacct switch to start a new pacct file. ckpacct also makes sure that there are at least 500 free blocks on the file system containing /usr/adm/acct (/usr by default). If there are not enough blocks, per-process accounting is turned off. The next time ckpacct is executed, it turns per-process accounting back on if there are enough free blocks.

   ckdacct performs an analogous function for daemon accounting. If a daemon's accounting file is larger than 500 blocks (default), the command /usr/lib/acct/turndacct switch is executed in order to start a new accounting file. In addition, ckdacct also checks the amount of free blocks on the ACCT_FS file system (/usr by default).

   Ensure that the ACCT_FS and MIN_BLKS variables have been set correctly in the /etc/config/acct_config configuration file. ACCT_FS is the file system containing /usr/adm/acct; the default is /usr. MIN_BLKS is the minimum number of free blocks needed in the ACCT_FS file system. The default is 500.

   It is very important that ckpacct and ckdacct be run periodically so that an administrator is notified when the accounting file system (/usr by default) runs out of disk space. After the file system is cleaned up, the next invocation of ckpacct and ckdacct enables per-process and daemon accounting. You can manually reenable accounting by invoking turnacct(8) and turndacct(8) with the on operand.

   If ckpacct and ckdacct are not run periodically, and the accounting file system runs out of space, an error message is written to the console stating that a write error occurred and that accounting is disabled. If you do not free disk space as soon as possible, a vast amount of accounting data can be lost unnecessarily. Additionally, lost accounting data can cause csarun(8) to abort or report erroneous information.

6. To run periodic accounting, an entry similar to the following should be made in `/usr/spool/cron/crontabs/root`. This command generates a periodic report on all consolidated data files found in `/usr/adm/acct/sum/data/*` and then deletes those data files:

```
15 5 1 * * /usr/lib/acct/csaperiod -r 2> /usr/adm/acct/nite/pd2log
```

   This entry is executed at such a time that `csarun`(8) has sufficient time to complete. This example results in the creation of a monthly accounting file and report on the first day of each month. These files contain information about the previous month's accounting.

7. Update the `holidays` file. The `/usr/lib/acct/holidays` file contains the prime/nonprime time table for the accounting system, which should be edited to reflect your site's holiday schedule for the year.

   By default, the `holidays` file is located in the `/usr/lib/acct` directory. You can change this location by modifying the *HOLIDAY_FILE* variable in `/etc/config/acct_config`. If necessary, modify the *NUM_HOLIDAYS* variable (also located in `/etc/config/acct_config`), which sets the upper limit on the number of holidays that can be defined in *HOLIDAY_FILE*. The format of this file is composed of the following types of entries:

   - Comment lines: These lines may appear anywhere in the file as long as the first character in the line is an asterisk (*).
   - Version line: This line must be the first uncommented line in the file and must only appear once. It denotes that the new holidays file format is being used. This line should not be changed by the site.
   - Year designation line: This line must be the second uncommented line in the file and must only appear once. The line consists of two fields. The first field is the keyword `YEAR`. The second field must be either the current year or the wildcard character, asterisk (*). If the year is wildcarded, the current year is automatically substituted for the year. The following are examples of two valid entries:

     ```
     YEAR        1997
     YEAR        *
     ```

   - Prime/nonprime time designation lines: These must be uncommented lines 3, 4, and 5 in the file. The format of these lines is as follows:

     period   *prime_time_start*   *nonprime_time_start*

The variable *period* is one of the following: `WEEKDAY`, `SATURDAY`, or `SUNDAY`. The *period* can be in either upper or lowercase.

The prime and nonprime start time can be one of two formats:

– Both start times are 4-digit numeric values between 0000 and 2359. The *nonprime_time_start* value must be greater than the *prime_time_start* value. For example, it is incorrect to have prime time start at 07:30 A.M. and nonprime time start at 1 minute after midnight. Therefore, the following entry is wrong and can cause incorrect accounting values to be reported.

  ```
  WEEKDAY  0730  0001
  ```

  It is correct to specify prime time to start at 07:30 A.M. and nonprime time to start at 5:30 P.M. on weekdays. You would enter the following in the holiday file:

  ```
  WEEKDAY  0730  1730
  ```

– Start times specify that the entire period is to be either all prime time or all nonprime time. To specify that the entire period is to be considered prime time, set *prime_time_start* to `ALL` and *nonprime_time_start* to `NONE`. If the period is to be considered all nonprime time, set *prime_time_start* to `NONE` and *nonprime_time_start* to `ALL`. For example, to specify Monday through Friday as all prime time, you would enter the following:

  ```
  WEEKDAY  ALL  NONE
  ```

  To specify all of Sunday to be nonprime time, you would enter the following:

  ```
  SUNDAY  NONE  ALL
  ```

• Company holidays lines: These entries follow the year designation line and have the following general format:

  day-of-year Month Day Description of Holiday

  The *day-of-year* field is a number in the range 1 through 366, indicating the day for a given holiday (leading white space is ignored). The other three fields are commentary and are not currently used by other programs. Each holiday is considered all nonprime time.

If the `holidays` file does not exist or there is an error in the year designation line, the default values for all lines are used.

If there is an error in a prime/nonprime time designation line, the entry for the erroneous line is set to a default value. All other lines in the `holidays` file are ignored and default values are used.

If there is an error in a company holidays line, all holidays are ignored.

The default values are as follows:

`YEAR`	The current year.
`WEEKDAY`	Monday through Friday is all prime time.
`SATURDAY`	Saturday is all nonprime time.
`SUNDAY`	Sunday is all nonprime time.

No holidays are specified

2.1.5 The `csarun`(8) Command

The `/usr/lib/acct/csarun` command is the primary daily accounting shell script. It processes connect, disk, per-process, and daemon accounting files and is normally initiated by `cron`(8) during nonprime hours.

`csarun`(8) also contains four user-exit points allowing sites to tailor the daily run of accounting to their specific needs (see Section 2.1.10.3, page 51 for information on setting up user exits callable from `csarun` and Section 2.2.3.1, page 84, for information on setting up a user exit callable from `runacct`).

The `csarun` command does not damage files in the event of errors. It contains a series of protection mechanisms that attempt to recognize an error, provide intelligent diagnostics, and terminate processing in such a way that `csarun` can be restarted with minimal intervention.

2.1.5.1 Daily Invocation

The `csarun` command is invoked periodically by `cron`(8). It is very important that you ensure that the previous invocation of `csarun` completed successfully before invoking `csarun` for a new accounting period. If this is not done, information about unfinished sessions will be inaccurate.

Data for a new accounting period can also be interactively processed by executing the following:

```
nohup csarun 2> /usr/adm/acct/nite/fd2log &
```

Before executing `csarun` in this manner, ensure that the previous invocation completed successfully. To do this, look at the files `active` and `statefile` in `/usr/adm/acct/nite`. Both files should specify that the last invocation completed successfully.

2.1.5.2 Error and Status Messages

The `csarun` error and status messages are placed in the `/usr/adm/acct/nite` directory. The progress of a run is tracked by writing descriptive messages to the file `active`. Diagnostic output during the execution of `csarun` is written to `fd2log`. The `lock` and `lock1` files prevent concurrent invocations of `csarun`; `csarun` will abort if these two files exist when it is invoked. The `clastdate` file contains the month, day, and time of the last two executions of `csarun`.

Errors and warning messages from programs called by `csarun` are written to files that have names beginning with `E` and ending with the current date and time. For example, `Ebld.11121400` is an error file from `csabuild`(8) for a `csarun` invocation on November 12, at 14:00.

If `csarun` detects an error, it sends an informational message to the operator with `msgi`(1), sends mail to `root` and `adm`, removes the locks, saves the diagnostic files, and terminates execution. When `csarun` detects an error, it will send mail either to `MAIL_LIST` if it is a fatal error, or to `WMAIL_LIST` if it is a warning message, as defined in the configuration file `/etc/config/acct_config`.

2.1.5.3 States

Processing is broken down into separate reentrant states so that `csarun` can be restarted. As each state completes, `/usr/adm/acct/nite/statefile` is updated to reflect the next state. When `csarun` reaches the `CLEANUP` state, it removes various data files and the locks, and then terminates.

The following describes the events that occur in each state. *MMDD* refers to the month and day `csarun` was invoked. *hhmm* refers to the hour and minute of invocation.

State	Description
`SETUP`	The current accounting files are switched via `turnacct`(8) and `turndacct`(8). These files are then moved to the `/usr/adm/acct/work/`*MMDD/hhmm* directory. File names are prefaced with `W.` `/etc/wtmp` and `/etc/csainfo` are also moved to this directory.
`WTMPFIX`	The `wtmp` file in the `work` directory is checked for accuracy by `wtmpfix` (see `fwtmp`(8)). Some date changes cause `csaline`(8) to fail, so `wtmpfix` attempts to adjust the time stamps in the `wtmp` file if a date change record appears. If `wtmpfix` is unable to fix the `wtmp` file, the `wtmp` file must be manually repaired. This is described in Section 2.1.6.1, page 21.
`VERIFY`	By default, per-process and NQS accounting files are checked for valid data. In addition, tape and socket accounting files are verified. Records with invalid data are removed. Names of bad data files are prefixed with `BAD.` in the `/usr/adm/acct/work/*` directory. The corrected files do not have this prefix.
`PREPROC`	The NQS and connect time (`wtmp`) accounting files are run through preprocessors. File names of preprocessed files are prefixed with a `P` in the `/usr/adm/acct/work/`*MMDD/hhmm* directory.
`ARCHIVE1`	First user exit of the `csarun` script. If a script named `/usr/lib/acct/csa.archive1` exists, it will be executed through the shell . (dot) command. The . (dot) command will not execute a compiled program, but the user exit script can. You might use this user exit to archive the accounting files in `${WORK}`.
`BUILD`	The per-process, NQS, tape, socket, and connect accounting data is organized into a session record file.
`ARCHIVE2`	Second user exit of the `csarun` script. If a script named `/usr/lib/acct/csa.archive2` exists, it will be executed through the shell . (dot) command. The . (dot) command will not execute a compiled program, but the user exit script can. You might use this exit to archive the session record file.
`CMS`	Produces a command summary file in `cacct.h` format. The `cacct` file is put into the `/usr/adm/acct/sum/data/`*MMDD/hhmm* directory for use by `csaperiod`(8).

`REPORT`	Generates the daily accounting report and puts it into `/usr/adm/acct/sum/rpt/`*MMDD*`/`*hhmm*`/rprt`. A consolidated data file, `/usr/adm/acct/sum/data/`*MMDD*`/`*hhmm*`/cacct`, is also produced from the session record file. In addition, accounting data for unfinished sessions is recycled.
`DREP`	Generates a daemon usage report based on the session file. This report is appended to the daily accounting report, `/usr/adm/acct/sum/rpt/`*MMDD*`/`*hhmm*`/rprt`.
`FEF`	Third user exit of the `csarun` script. If a script named `/usr/lib/acct/csa.fef` exists, it will be executed through the shell . (dot) command. The . (dot) command will not execute a compiled program, but the user exit script can. `csarun` variables are available, without being exported, to the user exit script. You might use this exit to convert the session record file to a format suitable for a front-end system.
`USEREXIT`	Fourth user exit of the `csarun` script. If a script named `/usr/lib/acct/csa.user` exists, it will be executed through the shell . (dot) command. The . (dot) command will not execute a compiled program, but the user exit script can. `csarun` variables are available, without being exported, to the user exit script. You might use this exit to run local accounting programs.
`CLEANUP`	Cleans up temporary files, removes the locks, and then exits.

2.1.5.4 Restarting `csarun`(8)

If `csarun`(8) is executed without arguments, the previous invocation is assumed to have completed successfully.

The following operands are required with `csarun` if it is being restarted:

`csarun [`*MMDD* `[`*hhmm* `[`*state*`]]]`

MMDD is month and day, *hhmm* is hour and minute, and *state* is the `csarun` entry state.

To restart `csarun`, follow these steps:

1. Remove all lock files by using the following command line:

   ```
   rm -f /usr/adm/acct/nite/lock*
   ```

2. Execute the appropriate `csarun` restart command, using the following examples as guides:

   a. To restart `csarun` using the time and state specified in `clastdate` and `statefile`, execute the following command:

   ```
   nohup csarun 0601 2> /usr/adm/acct/nite/fd2log &
   ```

   In this example, `csarun` will be rerun for June 1, using the time and state specified in `clastdate` and `statefile`.

   b. To restart `csarun` using the state specified in `statefile`, execute the following command:

   ```
   nohup csarun 0601 0400 2> /usr/adm/acct/nite/fd2log &
   ```

   In this example, `csarun` will be rerun for the June 1 invocation that started at 4:00 A.M., using the state found in `statefile`.

   c. To restart `csarun` using the specified date, time, and state, execute the following command:

   ```
   nohup csarun 0601 0400 BUILD 2> /usr/adm/acct/nite/fd2log &
   ```

   In this example, `csarun` will be restarted for the June 1 invocation that started at 4:00 A.M., beginning with state `BUILD`.

Before `csarun` is restarted, the appropriate directories must be restored. If the directories are not restored, further processing is impossible. These directories are as follows:

`/usr/adm/acct/work/`*MMDD/hhmm*
`/usr/adm/acct/sum/data/`*MMDD/hhmm*
`/usr/adm/acct/sum/rpt/`*MMDD/hhmm*
`/tmp/AC.`*MMDD/hhmm*

If you are restarting at state `ARCHIVE2`, `CMS`, `REPORT`, `DREP`, or `FEF`, the session file must already exist in `/tmp/AC.`*MMDD/hhmm*. If the file does not exist, `csarun` will automatically restart at the `BUILD` state. Depending on the tasks performed during the site-specific `USEREXIT` state, the session file may or may not need to exist.

2.1.6 Verifying and Correcting Data Files

This section describes how to remove bad data from various accounting files.

2.1.6.1 Fixing `wtmp`(8) Errors

The `wtmp` files generally cause the highest number of errors in the day-to-day operation of the accounting subsystem. When the date is changed, and the UNICOS system is in multiuser mode, a set of date change records is written into the `/etc/wtmp` file. The `wtmpfix` (see `fwtmp`(8)) program is designed to adjust the time stamps in the `wtmp` records when a date change is encountered.

Some combinations of date changes and reboots, however, slip by `wtmpfix` and cause `csaline`(8) to fail. The following example shows how to repair a `wtmp` file:

```
$ cd /usr/adm/acct/work/MMDD/hhmm
$ /usr/lib/acct/fwtmp < Wwtmp > xwtmp
$ ed xwtmp
    (delete corrupted records)
$ /usr/lib/acct/fwtmp -ic < xwtmp > Wwtmp
    (restart csarun at the WTMPFIX state)
```

If the `wtmp` file is beyond repair, create a null `Wwtmp` file. This prevents any charging of connect time.

2.1.6.2 Verifying Data Files

You can verify data files with the `csaedit`(8), `csapacct`(8), and `csaverify`(8) commands. `csaedit` and `csapacct` verify and delete bad data records, while `csaverify` only flags bad records. By default, `csaedit` and `csaverify` are invoked in `csarun` to verify the data files.

Note that these commands may allow files that contain bad data, such as very large values, to be successfully verified.

2.1.6.3 Editing Data Files

You can use the `csaedit`(8) and `csapacct`(8) commands to verify and remove records from various accounting files. The following example shows how you can use `csapacct` to verify and remove bad records from a per-process (`pacct`) accounting file.

In this example, `csapacct` is invoked with verbose mode enabled (valid data records are written to the file `pacct.NEW`):

```
/usr/lib/acct/csapacct -v pacct pacct.NEW
```

The output produced by this command line is as follows:

```
Bad record - starting byte offset is 077740 (32736)
   invalid pacct record - bad base parent process id 97867
Found the next magic word at byte offset 0100130, ignored 120 bytes


Found 394 BASE records
Found 4 EOJ records
Found 1 MTASK (multitasking) records
Found 0 ERROR records
Found 0 IO records
Found 0 SDS records
Found 0 MPP records
Found 0 PERFORMANCE records
Outputted records for 398 processes
Ignored 120 bytes from the input file
```

You can use `csaedit` and `csapacct` in conjunction with `csaverify`, by first running `csaverify` and noting the byte offsets of the first bad record. Next, execute `csaedit` or `csapacct` and remove the record at the specified offset. The following example shows how you can verify and then edit a bad `pacct` accounting file:

1. The `pacct` file is verified with the following command line, and the following output is received:

```
$  /usr/lib/acct/csaverify -P pacct

/usr/lib/acct/csaverify: pacct: invalid pacct record - bad base parent process id 97867
  byte offset: start = 077740 (32736)  word offset: start = 07774 (4092)
/usr/lib/acct/csaverify: pacct: invalid pacct record - bad magic word 03514000
  byte offset: start = 0100070 (32824)  word offset: start = 010007 (4103)
```

2. The record found at byte offset 32736 is deleted as follows (valid records are written to `pacct.NEW`):

   ```
   /usr/lib/acct/csapacct -o 32736 pacct pacct.NEW
   ```

3. The new `pacct` file is reverified as follows to ensure that all the bad records have been deleted:

   ```
   /usr/lib/acct/csaverify -P pacct.NEW
   ```

You can use `csaedit` to produce an abbreviated ASCII version of some of the daemon accounting files and `acctcom`(1) to generate a similar ASCII version of `pacct` files.

2.1.7 Files and Directories

This section describes the files and directories used by CSA.

2.1.7.1 `/usr/adm/acct` Directory

The `/usr/adm/acct` directory contains the following directories:

Directory	Description
`day`	Current accounting files
`fiscal`	Periodic accounting data and reports
`nite`	Processing messages and errors
`sum`	Daily accounting data and reports
`work`	Temporary work area

The `/usr/adm/acct/day` directory contains the current accounting files, as shown in the following list. Files with names ending with `0` contain data for uncompleted sessions from previous days.

File	Description
`dtmp`	Disk accounting data (ASCII) created by `dodisk`(8)
`nqacct*`	NQS daemon accounting data
`pacct*`	Per-process accounting data
`soacct*`	Socket accounting data
`tpacct*`	Tape daemon accounting data

The `/usr/adm/acct/fiscal/data/`*MMDD*`/`*hhmm* directory contains processed, periodic, binary accounting data in the form of the following files:

File	Description
`cms`	Periodic command usage data in command summary (cms) record format
`pdacct`	Consolidated periodic data generated on *MMDD* at *hhmm*

The `/usr/adm/acct/fiscal/rpt/`*MMDD*/*hhmm* directory contains the periodic accounting report, `rprt`, that was generated on *MMDD* at *hhmm*.

The `/usr/adm/acct/nite` directory contains error messages and status information about the accounting runs in the following files:

File	Description
`active`	Progress and status of `csarun`
`active`*MMDDhhmm*	Progress and status of `csarun` after an error has been detected
`clastdate`	Last two times `csarun` was executed; in *MMDD hhmm* format
`disktacct`	Disk accounting records in `cacct.h` format; created by `dodisk`(8)
`dk2log`	Diagnostic output created during execution of `dodisk`
`E*`*MMDDhhmm*	Error/warning messages for an accounting run done on *MMDD* at *hhmm*
`fd2log`	Diagnostic output created during execution of `csarun`
`lineuse`	tty line usage report
`lock, lock1`	Controls simultaneous invocations of `csarun`
`pd2log`	Diagnostic output created during execution of `csaperiod`
`pdact`	Progress and status of `csaperiod`
`pdact`*MMDDhhmm*	Progress and status of `csaperiod` after an error has been detected
`reboots`	The start and ending dates from `wtmp` and a listing of reboots
`statefile`	Current state during `csarun` execution

`tmpwtmp`	The `wtmp` file corrected by `wtmpfix` (see `fwtmp`(8))
`wtmperror`	`wtmpfix` error messages

The `/usr/adm/acct/sum/data/`*MMDD*/*hhmm* directory contains daily, binary accounting data in the following files:

File	Description
`cacct`	Consolidated daily data generated on *MMDD* at *hhmm* in `cacct.h` format
`cms`	Command usage data in command summary (cms) record format
`dacct`	Disk usage data in `cacct.h` format

The `/usr/adm/acct/sum/rpt/`*MMDD*/*hhmm* directory contains the daily accounting report, `rprt`, which was generated on *MMDD* at *hhmm*.

The `/usr/adm/acct/work/`*MMDD*/*hhmm* directory is used as a work area during the processing of the accounting data. It contains the following files:

File	Description
`BAD.Wnqacct*`	Unprocessed NQS accounting data containing bad records (verified by `csaedit`)
`BAD.Wpacct*`	Unprocessed per-process accounting data containing bad records (verified by `csaedit`)
`BAD.Wtpacct*`	Unprocessed tape accounting data containing bad records (verified by `csaedit`)
`Ever.tmp`	Data verification work file
`Pctime*`	Preprocessed connect time data
`Pnqacct*`	Preprocessed NQS data
`Puptime*`	Uptimes
`Rctime0`	Recycled connect data to be used in the next accounting period
`Rnqacct0`	Recycled NQS data to be used in the next accounting period
`Rpacct0`	Recycled per-process accounting data to be used in the next accounting run

`Rtpacct0`	Recycled tape data to be used in the next accounting period
`Ruptime0`	Recycled uptimes to be used in the next accounting period
`Wctime*`	Verified, unprocessed connect time data
`Wdisktacct`	Disk accounting data (`cacct.h` format) created by `acctdisk`(8)
`Wdtmp`	Disk accounting report (ASCII) created by `diskusg`(8) or `acctdusg`(8)
`Wnqacct*`	Unprocessed NQS accounting data
`Wpacct*`	Unprocessed per-process accounting data
`Wtpacct*`	Unprocessed tape accounting data
`Wsoacct*`	Unprocessed socket accounting data
`Wwtmp*`	Unprocessed connect time data

The `/tmp/AC.`*MMDD/hhmm* directory contains the session record file, `Super-record`, which is generated on *MMDD* at *hhmm*.

The `/usr/lib/acct` directory contains the following programs and shell scripts used by CSA:

Program / script	Description
`csaaddc`	Merges consolidated (`cacct`) accounting files
`csabuild`	Creates a session file
`csacon`	Creates a consolidated (`cacct`) accounting file
`csaconvert`	Converts UNICOS 8.0, 8.3, 9.0, 9.1, 9.2, and 9.3 accounting file(s), both System V and CSA, to a 10.0 format
`csacrep`	Generates consolidated accounting reports
`csadrep`	Reports daemon usage based on the session file
`csaedit`	Verifies, deletes records, and prints various data files
`csafef`	Template to convert session files to an IBM front-end format
`csafef2`	Template to summarize session file records by the tuple user name, job ID, and account ID.

`csagcon`	Consolidates accounting data for `session` and `pacct` files
`csagfef`	Formats consolidated accounting data
`csaibm`	Template to convert session files to an IBM front-end format
`csajrep`	Generates job reports from a session file
`csaline`	Preprocesses connect time data (`/etc/wtmp`)
`csanqs`	Preprocesses NQS accounting data
`csapacct`	Verifies and deletes records from a `pacct` file
`csaperiod`	Performs periodic accounting
`csaperm`	Changes the group ID and permissions on the accounting files
`csarecy`	Recycles session data for unfinished sessions
`csarun`	Performs daily accounting
`csaswitch`	Enables or disables kernel and daemon accounting
`csaverify`	Verifies various data files
`getconfig`	Extracts values from the configuration file

The `/usr/lib/acct` directory may also contain the following programs if your site uses the accounting user exits:

Program / script	Description
`csa.archive1`	Site-generated user exit for `csarun`
`csa.archive2`	Site-generated user exit for `csarun`
`csa.fef`	Site-generated user exit for `csarun`
`csa.user`	Site-generated user exit for `csarun`

2.1.7.2 `/etc` Directory

The `/etc` directory contains uptime and connect time data in the following files:

File	Description
`csaboots`	Captures system boot times
`csainfo`	Output file of `csaboots`
`wtmp`	Current connect time data

2.1.7.3 `/etc/config` Directory

The `/etc/config` directory is the location of the `acct_config` file that contains the configurable parameters used by the accounting commands. These parameters can be changed by using the UNICOS installation and configuration menu system (the *menu system*). To see the `acct_config` file parameters, use the following menu selection:

```
UNICOS 8.0 Installation / Configuration Menu System
    ->Configure System
        ->Accounting Configuration
```

The main menu for accounting configuration is as follows:

```
      Mainframe Dependent Parameters ==>
      Accounting Start Parameters ==>
      Block Device SBUs ==>
      Character Device SBUs ==>
      Connect Time SBU ==>
      Multitasking CPU SBUs=>
      NQS SBUs ==>
      Pacct File SBUs ==>
      Tape SBUs ==>
      Miscellaneous Settings ==>
      Parameters for CSARUN and CSAPERIOD ==>
      Site Defined Settings ==>

      Import accounting configuration ...
      Activate accounting configuration ...
      Reload default accounting configuration ...
```

Online help for the `acct_config` parameters is available through the menu system.

The main menu for accounting configuration displays a table of `acct_config` parameters and the current values.

The `Import accounting configuration ...` option gets the local site accounting configuration.

The `Activate accounting configuration ...` option rewrites the `/etc/config/acct_config` file with the current values selected in the menus.

The `Reload default accounting configuration ...` option reloads the default values for the `acct_config` file from the released `/usr/src/skl/etc/config/acct_config` file.

2.1.8 CSA Data Processing

The flow of data among the various CSA programs is explained in this section and is illustrated in Figure 2.

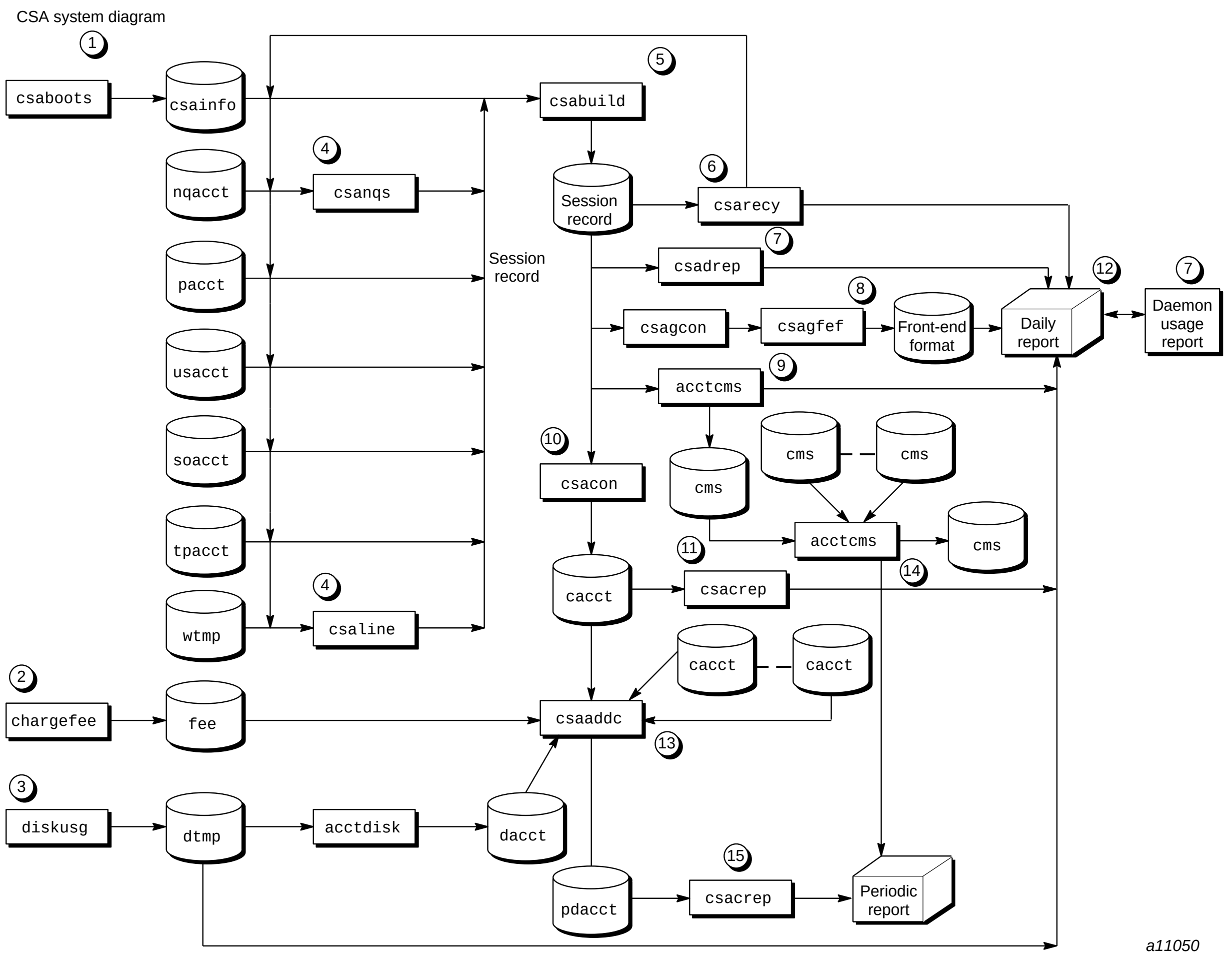


Figure 2. CSA Program Data Flow

1. Generate raw accounting files. Various daemons and system processes write to the raw accounting files. These accounting files include `pacct`, `nqacct`, `soacct`, `usacct`, `tpacct`, `wtmp`, and `csainfo`.

2. Create a fee file. Sites that want to charge fees to certain users can do so with the `chargefee`(8) command. `chargefee` creates a fee file that is processed by `csaaddc`(8).

3. Produce disk usage statistics. The `dodisk`(8) shell script allows sites to take snapshots of disk usage. `dodisk` does not report dynamic usage; it only

reports the disk usage at the time the command was run. Disk usage is processed by `csaaddc`.

4. Preprocess selected raw accounting files. Generally, a data file that must be preprocessed contains multiple records for a session. These records are scattered throughout the file, and the processing of the records often depends upon other events that are logged in the accounting file (for example, system reboot). The preprocessor collapses information about a session into one output record.

   NQS and connect time accounting data are preprocessed by `csanqs`(8) and `csaline`(8), respectively.

5. Organize the accounting data. `csabuild`(8) organizes the raw and preprocessed accounting data by sessions and boot times. With the exception of disk usage statistics and fees, the `csabuild` output file contains all of the accounting data available about each session.

   Sometimes data for terminated sessions is continually recycled. This can occur when accounting data is lost. To prevent data from recycling forever, edit `csarun` so that `csabuild` is executed with the `-o` *nday* option, which causes all sessions older than *nday* days to terminate. Select an appropriate *nday* value (see the `csabuild`(8) man page for more information).

6. Recycle information about unfinished sessions. Accounting data about uncompleted sessions is saved and processed again during the next accounting period. This information is recycled until the session completes or until manual intervention occurs. Accounting data for unfinished sessions is reported during each accounting period.

7. Generate the daemon usage report, which is appended to the daily report. `csadrep`(8) outputs information about interactive, NQS, tape, and socket usage.

8. Convert the session record file to a front-end format. Sites that process UNICOS accounting data on a front-end system can convert the session file to a format suitable for use on the front end by using the `csafef`(8), `csafef2`(8), or `csaibm`(8) command. These programs are templates, and you must modify them to suit your site's requirements. It is suggested that you use the user exit in the `FEF` section of `csarun` (see Section 2.1.5, page 16 and Section 2.1.10.3, page 51) to convert the session record file to your front-end format.

9. Generate command usage data. The information output by `acctcms`(8) is reported in the daily and periodic reports.

10. Consolidate the session record file. Session files are too large to retain on disk for any amount of time. Consequently, CSA consolidates the data and keeps the condensed version on disk. The accounting reports are based on the consolidated data. Data consolidation is done by `csacon`(8).

11. Generate an accounting report based on the consolidated data. `csacrep`(8) outputs the report.

12. Create the daily accounting report. The daily accounting report includes the following:

    - Connect time statistics (step 4)
    - Disk usage statistics (step 3)
    - Unfinished session information (step 6)
    - Command summary data (step 9)
    - Consolidated accounting report (step 11)
    - Last login information
    - Daemon usage report (step 7)

13. Generate periodic accounting data. Periodic accounting data is an accumulation of the consolidated data created in step 10. `csaaddc`(8) merges condensed data files together. The resulting file contains accounting information for numerous accounting periods.

14. Generate periodic command usage data. `acctcms`(8) merges command usage data from multiple accounting periods. The usage information was created in step 9. Both an ASCII and a binary file are created.

15. Produce a periodic accounting report. `csacrep`(8) is used to generate a report based on a periodic accounting file.

Steps 4 through 12 are performed during each accounting period by `csarun`(8). Periodic accounting (steps 13 through 15) is initiated by the `csaperiod`(8) command. Daily and periodic accounting, as well as fee and disk usage generation (steps 2 through 3), can be scheduled by `cron`(8) to execute regularly. See Section 2.1.4, page 11, for more information.

2.1.9 Data Recycling

A system administrator must correctly maintain recycled data in order to ensure accurate accounting reports. The following sections discuss data recycling and describe how an administrator can purge unwanted recycled accounting data.

Data recycling allows CSA to properly bill sessions that are active during multiple accounting periods. By default, `csarun`(8) reports data only for sessions that terminate during the current accounting period. Through data recycling, CSA preserves data for active sessions until the sessions terminate.

In the `Super-record` file, `csabuild`(8) flags each session as being either active or terminated. `csarecy`(8) reads the `Super-record` file and recycles data for the active sessions. `csacon`(8) consolidates the data for the terminated sessions, which `csaperiod`(8) uses later. `csabuild`, `csarecy`, and `csacon` are all invoked by `csarun`.

`csarun` puts recycled data in the `/usr/adm/acct/day` directory. Data files with names suffixed with `0` contain recycled data. For example, `ctime0`, `nqacct0`, `pacct0`, `tpacct0`, `usacct0`, and `uptime0` are generally the recycled data files that are found in `/usr/adm/acct/day`.

Normally, an administrator should not have to manually purge the recycled accounting data. This purge should only be necessary if accounting data is missing. Missing data can cause sessions to recycle forever and consume valuable CPU cycles and disk space.

2.1.9.1 How Sessions Are Terminated

Interactive sessions, `cron` jobs, and `at` jobs terminate when the last process in the job exits. Normally, the last process to terminate is the login shell. The kernel writes an end-of-job (`EOJ`) record to the `pacct` file when the session terminates.

When the NQS daemon delivers an NQS request's output, the request terminates. The daemon then writes an `NQ_DISP` record type to the `NQS` accounting file, while the kernel writes an `EOJ` record to the `pacct` file.

Unlike interactive sessions, NQS requests can have multiple `EOJ` records associated with them. In addition to the request's `EOJ` record, there can be `EOJ` records for pipe clients, net clients, and checkpointed portions of the request. The pipe client and net client perform NQS processing on behalf of the request.

The `csabuild` command flags sessions in the `Super-record` file as being terminated if they meet one of the following conditions:

- The session is an interactive, `cron`, or `at` job, and there is an `EOJ` record for the job in the `pacct` file.
- The session is an NQS request, and there is both an `EOJ` record for the request in the `pacct` file and an `NQ_DISP` record type in the NQS accounting file.
- The session is an interactive, `cron`, or `at` job and is active at the time of a system crash.
- The session is manually terminated by the administrator using one of the methods described in Section 2.1.9.3, page 35.

2.1.9.2 Why Recycled Sessions Should Be Scrutinized

Recycling unnecessary data can consume large amounts of disk space and CPU time. The session file and recycled data can occupy a vast amount of disk space on the file systems containing `/tmp` and `/usr/adm/acct/day`. Sites that archive data also require additional offline media. Wasted CPU cycles are used by `csarun` to reexamine and recycle the data. Therefore, to conserve disk space and CPU cycles, unnecessary recycled data should be purged from the accounting system.

Any of the following situations can cause CSA erroneously to recycle terminated sessions:

- Kernel or daemon accounting is turned off. At boot time, the `rc` command must execute `/usr/lib/acct/startup` in order to start kernel and daemon accounting.

  The kernel, `ckpacct`(8) command, or `ckdacct`(8) command can turn off accounting when there is not enough space on the file system containing `/usr/adm/acct/day`.
- Accounting files are corrupt. Accounting data can be lost or corrupted during a system or disk crash.
- Boot times are not recorded in `/etc/csainfo`. The `csaboots` command must be invoked by `rc` to write a boot time record to `/etc/csainfo`.
- Recycled data is erroneously deleted in a previous accounting period.

2.1.9.3 How to Remove Recycled Data

Before choosing to delete recycled data, you should understand the repercussions, as described in Section 2.1.9.4, page 37. Data removal can affect billing and can alter the contents of the consolidated data file, which is used by `csaperiod`.

You can remove recycled data from CSA in the following ways:

- Interactively execute the `csarecy -A` command. Administrators can select the active sessions that are to be recycled by running `csarecy` with the `-A` option. Users are not billed for the resources used in the sessions terminated in this manner. Deleted data is also not included in the consolidated data file.

  The following example is one way to execute `csarecy -A` (which generates two accounting reports and two consolidated files):

  1. Run `csarun` at the regularly scheduled time.
  2. Edit a copy of `/usr/lib/acct/csarun`. Change the `-r` option on the `csarecy` invocation line to `-A`. Also, do not redirect standard output to `${CRPT}/recyrpt`. The result should be similar to the following:

     ```
     csarecy -A -s ${SESSION_FILE} \
     -N ${WORK}/Rnqacct -P ${WORK}/Rpacct \
     -T ${WORK}/Rtpacct -U ${WORK}/Ruptime \
     -C ${WORK}/Rctime -u ${WORK}/Rusacct \
     2> ${NITE}/Erec.${DTIME}
     ```

     Since both the `-A` and `-r` options write output to `stdout`, the `-r` option is not invoked and `stdout` is not redirected to a file. As a result, the recycled job report is not generated.
  3. Execute the `jstat` command, as follows, to display a list of currently active jobs:

     ```
     jstat > jstat.out
     ```
  4. Execute the `qstat` command to display a list of NQS requests. The `qstat` command is used for seeing whether there are requests that are not currently running. This includes requests that are checkpointed, held, queued, or waiting.

In order to list all NQS requests, execute the `qstat` command, as follows, using a login that has either NQS manager or NQS operator privilege:

```
qstat -a > qstat.out
```

5. Interactively run the modified version of `csarun`. If you execute `csarun` soon after the first step is complete, this invocation of `csarun` completes quickly because not very much data exists.

   For each active session, `csarecy` asks you if you want to preserve the session. Preserve the active and nonrunning NQS sessions found in the third and fourth steps. All other sessions are candidates for removal.

- Execute `csabuild` with the `-o` *ndays* option, which terminates all active sessions older than the specified number of days. Resource usage for these terminated sessions is reported by `csarun`, and users are billed for the sessions. The consolidated data file also includes this resource usage.

  To execute `csabuild` with the `-o` option, edit `/usr/lib/acct/csarun`. Add the `-o` *ndays* option to the `csabuild` invocation line. Specify for *ndays* an appropriate value for your site.

  Recycled data for currently active sessions will be removed if you specify an inappropriate value for *ndays*.

- Execute `csarun` with the `-A` option. It reports resource usage for both active and terminated sessions, so users are billed for recycled sessions. This data is also included in the consolidated data file.

  None of the data for the active sessions, including the currently active sessions, is recycled. No recycled data files are generated in the `/usr/adm/acct/day` directory.

- Remove the recycled data files from the `/usr/adm/acct/day` directory. You can delete data for all of the recycled sessions, both terminated and active, by executing the following command:

  ```
  rm /usr/adm/acct/day/*[a-z]0
  ```

  The next time `csarun` is executed, it will not find data for any recycled sessions. Thus, users are not billed for the resources used in the recycled sessions, and this data is not included in the consolidated data file. `csarun` recycles the data for currently active sessions.

2.1.9.4 Adverse Effects of Removing Recycled Data

CSA assumes that all necessary accounting information is available to it, which means that CSA expects kernel and daemon accounting to be enabled and recycled data not to have been mistakenly removed. If some data is unavailable, CSA may provide erroneous billing information. Sites should be aware of the following facts before removing data:

- Users may or may not be billed for terminated recycled sessions. Administrators must understand which of the previously described methods cause the user to be billed for the terminated recycled sessions. It is up to the site to decide whether or not it is valid for the user to be billed for these sessions.

  For those methods that cause the user to be billed, both `csarun` and `csaperiod` report the resource usage.

- It may be impossible to reconstruct a terminated recycled session. If a recycled session is terminated by the administrator, but the session actually terminates in a later accounting period, information about the session is lost. If a user questions the resource billing, it may be extremely difficult or impossible for the administrator to correctly reassemble all accounting information for the session in question.

- Manually terminated recycled sessions be improperly billed in a future billing period. If the accounting data for the first portion of a session has been deleted, CSA may be unable to correctly identify the remaining portion of the job. Errors may occur, such as NQS requests being flagged as interactive sessions, or NQS requests being billed at the wrong queue rate. This is explained in detail in Section 2.1.9.5, page 38.

- CSA programs may detect data inconsistencies. When accounting data is missing, CSA programs may detect errors and abort.

The following table summarizes the effects of using the methods described in Section 2.1.9.3, page 35.

Table 1. Possible Effects of Removing Recycled Data

Method	Underbilling?	Incorrect billing?	Consolidated data file
`csarecy -A`	Yes. Users are not billed for the portion of the session that was terminated by `csarecy -A`.	Possible. Manually terminated recycled sessions may be billed improperly in a future billing period.	Does not include data for sessions terminated by `csarecy -A`.
`csabuild -o`	No. Users are billed for the portion of the session that was terminated by `csabuild -o`.	Possible. Manually terminated recycled sessions may be billed improperly in a future billing period.	Includes data for sessions terminated by `csabuild -o`.
`csarun -A`	No. All active and recycled sessions are billed.	Possible. All active and recycled sessions that eventually terminate may be billed improperly in a future billing period, because no data is recycled.	Includes data for all active and recycled sessions.
`rm`	Yes. All users are not billed for the portion of the session that was recycled.	Possible. All recycled sessions that eventually terminate may be billed improperly in a future billing period.	Does not include data for any recycled session.

By default, the consolidated data file contains data only for terminated sessions. Manual termination of recycled data may cause some of the recycled data to be included in the consolidated file. These cases are noted in the previous table.

2.1.9.5 NQS Requests and Recycled Data

In order for CSA to identify all NQS requests, data must be properly recycled. When an administrator manually purges recycled data for an NQS request, errors such as the following can occur:

- CSA flags the NQS request as an interactive session. This causes the request to be billed at interactive rates.
- The request is billed at the wrong queue rate.
- The wrong queue wait time is associated with the request.

These errors occur because valuable NQS accounting information was purged by the administrator. Only a few NQS accounting records are written by the NQS daemon, and all of the records are needed for CSA to properly bill NQS requests.

NQS accounting records are only written under the following circumstances:

- The NQS daemon receives a request.
- A request is routed to a queue.
- A request executes. This includes executing a request for the first time, and restarting and rerunning a request.
- A request terminates. A request can terminate because it is completed, requeued, preempted, held, checkpointed, or rerun by the operator.
- Output is delivered.

Thus, for long running requests that span days, there can be days when no NQS data is written. Consequently, it is extremely important that accounting data be recycled. If the site administrator manually terminates recycled sessions, care must be taken to be sure that only nonexistent NQS requests are terminated.

2.1.10 Tailoring CSA

This section describes the following actions in CSA:

- Setting up SBUs
- Setting up daemon accounting
- Setting up user exits
- Modifying the front-end formatting templates
- Modifying the charging of NQS jobs based on NQS termination status
- Tailoring CSA shell scripts
- Using `at`(1) instead of `cron`(8) to periodically execute `csarun`
- Allowing users without super-user permissions to execute CSA
- Using an alternate configuration file

2.1.10.1 System Billing Units (SBUs)

A *system billing unit* (SBU) is a unit of measure that reflects use of machine resources. You can alter the weighting factors associated with each field in each accounting record to obtain an SBU value suitable for your site. SBUs are defined in the accounting configuration file, `/etc/config/acct_config`. By default, all SBUs are set to `0.0`.

The source code for the default SBU calculations is located in `/usr/src/cmd/acct/lib/acct/sbu.c`. For sites that do not have source code, the default algorithms are also defined in `/usr/src/cmd/acct/lib/acct/user_sbu.c`. By modifying `/usr/src/cmd/acct/lib/acct/user_sbu.c`, compiling, and relinking the accounting programs, your site can use local SBU calculations.

Accounting allows different periods of time to be designated either prime or nonprime time (the time periods are specified in `/usr/lib/acct/holidays`).

Following is an example of how the prime/nonprime algorithm works:

Assume a user uses 10 seconds of CPU time, and executes for 100 seconds of prime wall-clock time, and pauses for 100 seconds of nonprime wall-clock time. Therefore, elapsed time is 200 seconds (100+100). If

prime = *prime time* / *elapsed time*
nonprime = *nonprime time* / *elapsed time*
cputime[PRIME] = *prime* * *CPU time*
cputime[NONPRIME] = *nonprime* * *CPU time*

then

cputime[PRIME] `== 5 seconds`
cputime[NONPRIME] `== 5 seconds`

Under CSA, an SBU value is associated with each record in the Session record file when that file is assembled by `csabuild`(8). Final summation of the SBU values is done by `csacon`(8) during the creation of the `cacct` record file.

Billing for SBU values is intended to be a combination of all the SBU values from each record associated with a job, as follows:

*Total SBU = (NQS queue SBU value) * (sum of all pacct record SBUs*
+ sum of all tape record SBUs
+ sum of all ctmp record SBUs)

This allows a site to bill different NQS queues at differing rates. Again, if the available formulas are insufficient to achieve the site's requirements, a site can modify the calculations found in the `sbu` library routine, `/usr/src/cmd/acct/lib/acct/user_sbu.c`.

2.1.10.1.1 `pacct` SBUs

The SBUs for `pacct` data are separated into prime and nonprime values. Prime and nonprime use is calculated by a ratio of elapsed time. If you do not want to make a distinction between prime and nonprime time, set the nonprime time SBUs and the prime time SBUs to the same value. Prime time is defined in `/usr/lib/acct/holidays`. By default, Saturday and Sunday are considered nonprime time.

The following is a list of prime time `pacct` SBU weights. Descriptions and factor units for the nonprime time SBU weights are similar to those listed here. SBU weights are defined in `/etc/config/acct_config`.

Value	Description
`P_BASIC`	Prime-time weight factor. `P_BASIC` is multiplied by the sum of prime time SBU values to get the final SBU factor for the `pacct` base record.
`P_TIME`	General-time weight factor. `P_TIME` is multiplied by the time SBUs (made up of `P_STIME`, `P_ITIME`, `P_SCTIME`, and `P_INTTIME`) to get the time contribution to the `pacct` base record SBU value.
`P_STIME`	System CPU-time weight factor. The unit used for this weight is *billing units* per second. `P_STIME` is multiplied by the system CPU time to get the system CPU factor.
`P_UTIME`	User CPU-time weight factor. The unit used for this weight is *billing units* per second. `P_UTIME` is multiplied by the user CPU time to get the user CPU factor. User time is the sum of user times after weighting for multitasking. Multitasking may affect the user CPU cost if the `MUTIME_WEIGHT` parameters have been set to values other than 1.0. See the following explanation of these values.

`P_ITIME`	This is the weight factor for the time spent waiting in the kernel for I/O while the process is locked in memory. The unit used for this weight is *billing units* per second. `P_ITIME` is multiplied by the I/O wait time.
`P_SCTIME`	Weight factor for system call time. The unit used for this weight is *billing units* per second.
`P_INTTIME`	Weight factor for interrupt time. The unit used for this weight is *billing units* per second.
`P_MEM`	General-memory-integral weight factor. `P_MEM` is multiplied by the memory SBUs (made up of `P_XMEM` and `P_IMEM`) to get the memory contribution to the `pacct` base record SBU value.
`P_XMEM`	CPU-time-memory-integral weight factor. The unit used for this weight is *billing units* per Kword-minute. `P_XMEM` is multiplied by the memory integral (see Section 2.1.12.1, page 63). This value is affected by your site's choice of `MEMINT` (in the accounting configuration file `/etc/config/acct_config`).
`P_IMEM`	The weight factor used with the I/O wait time memory integral. This integral includes the I/O wait time while the process is locked in memory. The unit used for this weight is *billing units* per Kword-minute. `P_IMEM` is multiplied by the I/O-wait-time memory integral.
`P_IO`	General-I/O weight factor. `P_IO` is multiplied by the I/O SBUs (made up of `P_BYTEIO`, `P_PHYIO`, and `P_LOGIO`) to get the I/O contribution to the `pacct` base record SBU value.
`P_BYTEIO`	Characters-transferred weight factor. The unit used for this weight is *billing units* per character transferred. `P_BYTEIO` is multiplied by the bytes of I/O transferred. If tape or device I/O is to be charged at a rate other than `P_BYTEIO`, the tape and device weight factors need to be adjusted accordingly. See Section 2.1.11.1, page 56 (field `ac_io`), for more information.

`P_PHYIO`	Physical-I/O-request weight factor. The unit used for this weight is *billing units* per physical I/O request. `P_PHYIO` is multiplied by the number of physical I/O requests made. Physical requests are the number of driver requests made.
`P_LOGIO`	Logical-I/O-request weight factor. The unit used for this weight is *billing units* per logical I/O request. `P_LOGIO` is multiplied by the number of logical I/O requests made. The number of logical I/O requests is the total number of `read`(2), `write`(2), `reada`(2), and `writea`(2) system calls. The number of strides, multiplied by the number of requests processed by each `listio`(2) call, is also added to the total.

2.1.10.1.2 Multitasking SBUs

The `MUTIME_WEIGHT` *i* variables define the weighting factors that are used to charge user CPU time for multitasking programs. It is used in conjunction with the `ac_mutime` array (see `/usr/include/sys/acct.h`), which defines the amount of CPU time the multitasking program spent with *i* + 1 CPUs connected.

`MUTIME_WEIGHT` *i* defines the marginal cost of getting the *i* + 1 CPU at one instant. If these values are set to less than 1.0, there is an incentive for multitasking. If the values are set to 1.0, multitasking programs are charged for user CPU time just as all other programs.

For more information on multitasking incentives, see Section 2.1.12, page 63.

2.1.10.1.3 SDS SBUs

Secondary data storage (SDS) system billing units are calculated from the statistics on SDS use in the `pacct` file. The SBU factors are defined in `/etc/config/acct_config`.

The values are as follows:

Value	Description
`NP_SDSMEM` or `P_SDSMEM`	SDS-memory-integral weight factor. The memory integral is based on residency time and not on execution time. `P_SDSMEM`

or NP_SDSMEM is multiplied by the SDS memory integral. The unit used for this weight is *billing units* per Mword-second.

NP_SDSLOGIO or P_SDSLOGIO

SDS-logical-I/O-request weight factor. P_SDSLOGIO or NP_SDSLOGIO is multiplied by the number of SDS logical I/O requests. The unit used for this weight is *billing units* per logical I/O request.

NP_SDSBYTEIO or P_SDSBYTEIO

SDS-characters-transferred weight factor. P_SDSBYTEIO or NP_SDSBYTEIO is multiplied by the number of SDS characters transferred. The unit used for this weight is *billing units* per character transferred.

The SBU values should be very small. On Cray systems, it is possible to submit a very large number of requests to SDS in a short time; therefore, to prevent these numbers from dominating the SBU values, small weight factors must be used. Values of 0 result in no charge.

2.1.10.1.4 MPP SBUs

Massively parallel processing (MPP) system billing units are calculated from the statistics on MPP use in the pacct file. The SBU factors are defined in /etc/config/acct_config.

The P_MPPPE or NP_MPPPE SBUs are the MPP processing elements (PEs) weight factors, prime and nonprime charges. The prime time billing units for PEs is calculated using the following equation:

$$\texttt{P\_MPPPE billing units = P\_MPPPE *} \sum_{0}^{\#\ of\ sessions} (\textit{no. MPP PEs used} \; * \; \textit{MPP time used})$$

The nonprime time billing units for PEs is calculated using the following equation:

$$\texttt{NP\_MPPPE billing units = NP\_MPPPE *} \sum_{0}^{\#\ of\ sessions} (\textit{no. MPP PEs used} \; * \; \textit{MPP time used})$$

The unit used for these weights is billing units per PE-second.

The P_MPPBB or NP_MPPBB SBUs are the MPP barrier bits weight factors, prime and nonprime charges.[1] The prime time billing units for barrier bits is calculated using the following equation:

$$\texttt{P\_MPPBB billing units = P\_MPPBB * } \sum_{0}^{\#\ of\ sessions} (\textit{no. MPP barrier bits used} * \textit{MPP time used})$$

The nonprime time billing units for barrier bits is calculated using the following equation:

$$\texttt{NP\_MPPBB billing units = NP\_MPPBB * } \sum_{0}^{\#\ of\ sessions} (\textit{no. MPP barrier bits used} * \textit{MPP time used})$$

The unit used for these weights is *billing units* per barrier bit-second.

The P_MPPTIME or NP_MPPTIME SBUs are the MPP time weight factors, prime and nonprime charges. The prime time billing units for MPP time is calculated using the following equation:

$$\texttt{P\_MPPTIME billing units = P\_MPPTIME * } \sum_{0}^{\#\ of\ sessions} (\textit{MPP time used})$$

The nonprime time billing units for MPP time is calculated using the following equation:

$$\texttt{NP\_MPPTIME billing units = NP\_MPPTIME * } \sum_{0}^{\#\ of\ sessions} (\textit{MPP time used})$$

The unit used for these weights is *billing units* per second.

The SBU values should be very small, which will prevent these numbers from dominating the SBU values. Values of 0 result in no charge.

2.1.10.1.5 Connect Time SBUs

There are SBUs for both prime- and nonprime-time connect data. The SBU values should reflect the system billing units per second of connect time. The weight factors, CON_PRIME and CON_NONPRIME, are defined in /etc/config/acct_config.

2.1.10.1.6 NQS SBUs

The /etc/config/acct_config file contains the configurable parameters that pertain to NQS SBUs.

[1] Deferred implementation.

The `NQS_NUM_QUEUES` parameter sets the number of queues for which you want to set SBUs (the value must be set to at least 1). Each `NQS_QUEUE` *x* variable in the configuration file has a queue name and an SBU pair associated with it (the total number of queue/SBU pairs must equal `NQS_NUM_QUEUES`). The queue/SBU pairs define weights for the queues. If an SBU value is less than 1.0, there is an incentive to run jobs in the associated queue; if the value is 1.0, jobs are charged as though they are non-NQS jobs; and if the SBU is 0.0, there is no charge for jobs running in the associated queue. SBUs for queues not found in the configuration file are automatically set to 1.0.

The `NQS_NUM_MACHINES` parameter sets the number of originating machines for which you want to set SBUs (the value must be at least 1). Each `NQS_MACHINE` *x* variable in the configuration file has an originating machine and an SBU pair associated with it (the total number of machine/SBU pairs must equal `NQS_NUM_MACHINES`). SBUs for originating machines not specified in `/etc/config/acct_config` are automatically set to 1.0.

The queue and machine SBUs are multiplied together to give an NQS multiplier. If the SBUs are set to less than 1.0, there is an incentive to run jobs in these queues or from these machines. SBUs of 1.0 indicate that jobs in the queues or from associated hosts are billed normally.

2.1.10.1.7 Socket SBUs

Currently, there is no way to charge for socket accounting. The socket accounting records produced are only processed in order to make the data available to the site-supplied user exits.

2.1.10.1.8 Tape SBUs

There is a set of weighting factors for each group of tape devices. By default, there are only two groups, `tape` and `cart`. The `TAPE_SBU`*i* parameters in `/etc/config/acct_config` define the weighting factors for each group. There are SBUs associated with the following:

- Number of mounts
- Device reservation time (seconds)
- Number of bytes read
- Number of bytes written

2.1.10.1.9 Device SBUs

Device accounting system billing units are calculated from the device statistics in the `pacct` file. SBUs can be set for both block and character devices in `/etc/config/acct_config`. The fields in the `acct_config` file that affect SBU factors for each device are as follows:

SBU factor	Description
`Logical I/O Sbu`	Weight given to each logical I/O request.
`Characters Xfer Sbu`	Weight given to the amount of data transferred.
`Device Name`	Device type name (see Section 2.1.14, page 66).

The `Logical I/O Sbu` factor is multiplied by the number of system calls that initiated I/O on a device type. The `Characters Xfer Sbu` factor is multiplied by the number of bytes of data transferred to a device type.

The SBUs for block devices are labeled `BLOCK_DEVICE` *x*, where *x* is a number from `0` to `MAXBDEVNO-1`. Character devices are labeled `CHAR_DEVICE` *x*, where *x* is a number from `0` to `MAXCDEVNO-1`. The numeric suffixes for the `CHAR_DEVICE` *x* variables must match the major numbers in `/dev`, which are defined in `/usr/src/uts/c1/cf/devsw.c` in the `cdevsw[]` array.

`MAXBDEVNO` and `MAXCDEVNO` are located in the `/usr/include/sys/param.h` include file and have default values of 10 and 35, respectively.

Device accounting is also discussed in Section 2.1.14, page 66.

The SBU values should be very small. On Cray systems, it is possible to perform a very large number of I/O requests very quickly; therefore, to prevent these numbers from dominating the SBU values, a small weight factor must be used. A value of 0 results in no charge.

2.1.10.1.10 Example SBU Settings

The following section provides an example showing how you could set up the SBU system. This example is restricted to `pacct` base records (you should also consider `pacct` multitasking, `pacct` I/O (device accounting), and all the daemon records). In this example, it is assumed that an SBU is equal to one dollar of charge.

The formula for calculating the whole `pacct` base record SBU value is as follows:

```
PSBU = ((P_TIME * (P_STIME * stime + P_UTIME * utime + P_ITIME *
iowtime)) + (P_MEM * (P_XMEM * cpumem + P_IMEM * iowmem)) +
(P_IO * (P_BYTEIO * bytes + P_PHYIO * phy + P_LOGIO * log)));

NSBU = ((NP_TIME * (NP_STIME * stime + NP_UTIME * utime + NP_ITIME
* iowtime)) + (NP_MEM * (NP_XMEM * cpumem + NP_IMEM * iowmem)) +
(NP_IO*(NP_BYTEIO * bytes + NP_PHYIO * phy + NP_LOGIO * log)));

SBU = P_BASIC * PSBU + NP_BASIC * NSBU;
```

The variables in this formula are as follows:

Variable	Description
stime	System CPU time in seconds.
utime	User CPU time in seconds. User CPU time is the sum of user times after weighting for multitasking.
iowtime	Time (in seconds) spent waiting in the kernel for I/O while the process is locked in memory.
cpumem	Memory integral (see Section 2.1.12.1, page 63).
iowmem	I/O-wait-time memory integral.
bytes	Number of bytes of data transferred.
phy	Number of physical I/O requests made.
log	Number of logical I/O requests made.

All time is considered prime time. Therefore, the nonprime time SBUs should be set to the same values as their prime time counterparts.

In order to produce a billing that is somewhat repeatable, this example omits various values, such as physical I/O (set `P_PHYIO` to `0.0`), that depend on the state of the machine at run time. In particular, system time varies greatly due to system load and will cause this example to be nonrepeatable. Information on which fields generate repeatable values is contained in Section 2.1.11.1, page 56.

In this example, users are charged for each logical request (`P_LOGIO`) and the total data moved (`P_BYTEIO`). This provides users with an incentive to use larger I/O requests, which may be more efficient. Processes that perform I/O that locks them into memory are penalized (`P_IMEM`), because this may result in memory fragmentation.

In this example, users are charged the following amounts for time (the accounting record fields associated with the charge are also identified):

- $100 per hour of user CPU time. This is equal to $100 per 3600 seconds, which is $0.02777777 per second (`P_UTIME`). To produce repeatable billing, system time must be excluded. Thus, `P_STIME` is set to `0.0`.
- $25 for each megaword of memory per hour of CPU time. The memory integrals are in units of Kword-minutes, so the weighting factor is $25/(60 minutes * 2^{10} Kwords) or 0.0004069010 (`P_XMEM`).
- $3 for each hour spent waiting on I/O while locked into memory. The wait time is in units of seconds. so the weighting factor is $3/3600 seconds or .0008333333 (`P_ITIME`).
- $25 for I/O wait time (locked in memory) per hour. This is the same value as the memory charge because the process is using memory during this time in the same way it would when executing. The weighting factor is $25/(60 seconds * 2^{10} Kwords) or 0.0004069010 (`P_IMEM`).
- A DD-XX disk drive can perform I/O at a maximum rate of 9.6 Mbytes per second. Assume that the original cost of the drive was $125,000, and it will be paid for in 2 years. Also assume that it is busy 5% of the time (63072000 seconds * 5% = 3153600 seconds). The amount of I/O that can be completed in 2 years is 31745177026560 bytes (9.6 Mbytes/second * 3153600 seconds). Thus, you would charge $125,000/31745177026560 bytes or $0.00000000393760 per byte, which is approximately $0.33/10 Mwords (`P_BYTEIO`).
- $0 for physical I/O requests. This charge makes the billing more repeatable. The byte I/O charge covers this activity (`P_PHYIO`).
- $0.01 per thousand logical I/O requests. This charge encourages the user to perform larger I/O requests by charging less for a lower number of larger I/O requests (instead of a lot of small I/O requests). The weighting factor is computed as $0.01/1000 I/O requests or 0.00001 (`P_LOGIO`).

Therefore, in this example, the `pacct` base record charges are as follows (the nonprime time SBUs are set to the same value as their prime time counterparts):

Weight factor	Charge
`P_BASIC`	1.0
`P_TIME`	1.0
`P_UTIME`	0.02777777777777
`P_STIME`	0.0
`P_ITIME`	0.00083333333333
`P_MEM`	1.0
`P_XMEM`	0.00040690104166
`P_IMEM`	0.00040690104166
`P_IO`	1.0
`P_BYTEIO`	0.00000000393760
`P_PHYIO`	0.0
`P_LOGIO`	0.00001000000000

`P_BASIC`, `P_TIME`, `P_MEM`, and `P_IO` are used to weight different factors of the equation; you can use these depending on how your other groups of weighting factors are picked. For example, you could change the `P_IO` and `P_BYTEIO` factors as follows and receive the same results:

Weight factor	Charge
`P_BASIC`	1.0
`P_TIME`	1.0
`P_UTIME`	0.02777777777777
`P_STIME`	0.0
`P_ITIME`	0.00083333333333
`P_MEM`	1.0
`P_XMEM`	0.00040690104166
`P_IMEM`	0.00040690104166
`P_IO`	0.00001

P_BYTEIO	0.000393760
P_PHYIO	0.0
P_LOGIO	1.0

2.1.10.2 Daemon Accounting

Accounting information is available from NQS, online tapes, and sockets. Data is written to the `nqacct`, `tpacct`, and `soacct` files, respectively, in the `/usr/adm/acct/day` directory.

In most cases, daemon accounting must be enabled by both the CSA subsystem and the daemon. Section 2.1.4, page 11, describes how to enable daemon accounting at system startup time. You can also enable daemon accounting after the system has booted.

You can enable accounting for a specified daemon with the `turndacct`(8) command. For example, to start tape accounting, you would execute the following:

```
/usr/lib/acct/turndacct on tape
```

The NQS and online tape daemon also must enable accounting. Use the `qmgr set accounting on` command to turn on NQS accounting. Tape daemon accounting is enabled when `tpdaemon`(8) is executed with the `-c` option.

Daemon accounting is disabled by `shutacct`(8) at system shutdown (see Section 2.1.4, page 11). It can also be disabled at any time by the `turndacct`(8) command when used with the `off` operand. For example, to disable NQS accounting, execute the following command:

```
/usr/lib/acct/turndacct off nqs
```

New daemon accounting files can be started when `turndacct` is invoked with the `switch` operand. No data is lost when files are switched. For example, to start a new NQS accounting file, execute the following command:

```
/usr/lib/acct/turndacct switch nqs
```

2.1.10.3 Setting up User Exits

CSA accommodates the following user exits, which can be called from certain `csarun` states:

`csarun` state	User exit
`ARCHIVE1`	`/usr/lib/acct/csa.archive1`
`ARCHIVE2`	`/usr/lib/acct/csa.archive2`
`FEF`	`/usr/lib/acct/csa.fef`
`USEREXIT`	`/usr/lib/acct/csa.user`

These exits allow an administrator to tailor the `csarun` procedure to the individual site's needs by creating scripts to perform additional site-specific processing during daily accounting.

While executing, `csarun` checks in the `ARCHIVE1`, `ARCHIVE2`, `FEF`, and `USEREXIT` states for a shell script with the appropriate name.

If the script exists, it is executed via the shell `.` (dot) command. If the script does not exist, the user exit is ignored. The `.` (dot) command will not execute a compiled program, but the user exit script can. `csarun` variables are available, without being exported, to the user exit script. `csarun` checks the return status from the user exit and, if it is nonzero, the execution of `csarun` is terminated.

If CSA is run by a user without super-user permissions, the user exits must be both readable and executable by this user (see page Section 2.1.10.7, page 54).

2.1.10.4 Charging for NQS Jobs

By default, SBUs are calculated for all NQS jobs regardless of the job's NQS termination code. If you do not want to bill portions of an NQS request, set the appropriate `NQS_TERM_` *xxxx* variable (termination code) in `/etc/config/acct_config` to `0`, which sets the SBU for this portion to `0.0`. By default, all portions of a request are billed.

The following table describes the termination codes:

Code	Description
`NQS_TERM_EXIT`	Generated when the request finishes running and is no longer in a queued state. At NQS shutdown time, requests that specified both the `-nc` (no checkpoint) and `-nr` (no rerun) options for `qsub` also have `NQS_TERM_EXIT` records written. In addition, this record is written for requests that specified the `-nr` option for `qsub` and were running at the time of a system crash.

`NQS_TERM_REQUEUE`	Written for running requests that are checkpointed and then requeued when NQS shuts down.
`NQS_TERM_PREEMPT`	Written when a request is preempted with the `qmgr preempt request` command.
`NQS_TERM_HOLD`	Written for a request that is checkpointed with the `qmgr hold request` command. The `hold request` command differs from the checkpoint done at daemon shutdown time because a "hold" keeps the job from being scheduled until a `qmgr release` command is executed.
`NQS_TERM_OPRERUN`	Written when a request is rerun with the `qmgr rerun request` command. At NQS shutdown time, jobs that cannot be checkpointed and do not have the `-nr` (no rerun) option for `qsub` specified have this type of termination record written. The requests are requeued with this status.

2.1.10.5 Tailoring CSA Shell Scripts and Commands

Modify the following variables in `/etc/config/acct_config` if necessary:

Variable	Description
`ACCT_FS`	File system on which `/usr/adm/acct` resides. The default is `/usr`.
`MAIL_LIST`	List of users to whom mail is sent if fatal errors are detected in the accounting shell scripts. The default is `root` and `adm`.
`WMAIL_LIST`	List of users to whom mail is sent if warning errors are detected by the `csarun` script at cleanup time. The default is `root` and `adm`.

MIN_BLKS	Minimum number of free blocks needed in ${ACCT_FS} to run csarun or csaperiod. The default is 500 free blocks.

2.1.10.6 Using at to Execute csarun(8)

You can use the at(1) command instead of cron(8) to execute csarun periodically. If your Cray system is down when csarun is scheduled to run via cron, csarun will not be executed until the next scheduled time. On the other hand, at jobs execute when the machine reboots if their scheduled execution time was during a down period.

You can execute csarun with at in several ways. For instance, a separate script can be written to execute csarun and then resubmit the job at a specified time. Also, an at invocation of csarun could be placed in a user exit script, /usr/lib/acct/csa.user, that is executed from the USEREXIT section of csarun. See Section 2.1.10.3, page 51, for more information.

2.1.10.7 Allowing Nonsuper Users to Execute CSA

Your site may want to allow users without super-user permissions to run CSA accounting. CSA can be run by users who are in the group adm and have permission bit acct set in their UDB entries.

Note: If root has run CSA, you must execute the shell script /usr/lib/acct/csaperm (see csaperm(8)) to change the group ID and file permissions of all accounting files in /usr/adm/acct so they can be accessed by a nonsuper user running CSA.

The following steps describe the process of setting up CSA so it is executed automatically on a daily basis by a user without super-user permissions. In this example, the user without super-user permissions is adm:

1. Ensure that user adm is a member of group adm and has the permission bit acct set in its UDB entry (see udbgen(8)).

2. As root, execute the shell script csaperm to change the group ID and file permissions of all accounting files in /usr/adm/acct so they can be accessed by a nonsuper user.

3. Ensure that, if they exist, the user exits /usr/lib/acct/csa.archive1, /usr/lib/acct/csa.archive2, /usr/lib/acct/csa.fef, and /usr/lib/acct/csa.user have the group ID adm and are both readable and executable by group adm.

4. Follow steps 1 through 5 of Section 2.1.4, page 11, to set up system billing units, record system boot times, and turn off accounting before system shutdown.

5. Include an entry similar to the following in `/usr/spool/cron/crontabs/root` so that `cron`(8) automatically runs `dodisk`(8):

   ```
   0 3 * * 1-6 /usr/lib/acct/dodisk -a -v 2> /usr/adm/acct/nite/dk2log
   ```

   `dodisk` must be executed by `root`, because no other user has the correct permissions to read `/dev/dsk/*`.

6. Include entries similar to the following in `/usr/spool/cron/crontabs/adm` so that user `adm` automatically runs daily accounting by using `cron`:

   ```
   0 4 * * 1-6 /usr/lib/acct/csarun 2> /usr/adm/acct/nite/fd2log
   0 * * * * /usr/lib/acct/ckdacct nqs tape
   0 * * * * /usr/lib/acct/ckpacct
   ```

   `csarun`(8) should be executed at a time that allows `dodisk` to complete. If `dodisk` does not complete before `csarun` executes, disk accounting information may be missing or incomplete.

7. To run periodic accounting, place an entry similar to the following in `/usr/spool/cron/crontabs/adm` (this command generates a periodic report on all consolidated data files found in `/usr/adm/acct/sum/data/*` and then deletes those data files):

   ```
   15 5 1 * * /usr/lib/acct/csaperiod -r 2>/usr/adm/acct/nite/pd2log
   ```

8. Update the holidays file as described in Section 2.1.4, page 11.

2.1.10.8 Using an Alternate Configuration File

By default, the `/etc/config/acct_config` configuration file is used when any of the CSA commands are executed. You can specify a different file by setting the shell variable `ACCTCONFIG` to another configuration file, and then executing the CSA commands.

For example, you would execute the following commands in order to use the configuration file `/tmp/myconfig` while executing `csarun`(8):

```
ACCTCONFIG=/tmp/myconfig /usr/lib/acct/csarun 2> /usr/adm/acct/nite/fd2log
```

2.1.10.9 Disk Usage Reporting (diskusg)(8)

The diskusg(8) command can be configured at your site. The site.c module of diskusg contains an example to help you in customizing a report for your site. You can delete your choice of comment-protection characters in the example, compile the routine, relink diskusg, then print a sample report of disk usage for your site. You can execute your modified diskusg command in the USEREXIT state in csarun or runacct scripts.

2.1.11 Per-process Accounting Data

This section describes some of the fields found in the pacct file. /usr/include/sys/acct.h defines the structure of this file.

2.1.11.1 Base Accounting Record

One base accounting record per process is written; each record contains the following fields:

Table 2. Base Accounting Record Fields by Function

Type	Field	Description
Header	ac_header	Accounting header record word (see /usr/include/sys/accthdr.h)
	ac_flag	Accounting flags.
Identifiers	ac_acid	Account ID.
	ac_gid	Real group ID.
	ac_jobid	Job ID.
	ac_pid	Process ID.
	ac_ppid	Parent process ID.
	ac_uid	Real user ID.
Process Information	ac_btime	Start time of the process.
	ac_comm	Command name (first 8 characters).
	ac_etime	Elapsed time while process executed (in clocks). This number is not repeatable.

Type	Field	Description
	`ac_himem`	Memory-use high water mark in words.
	`ac_nice`	Nice value, measured at the end of 1 second of system and user time or the most expensive value used thereafter. This allows a process to set the value at which most of its work should be done; only charges for increased cost are levied.
	`ac_stat`	Low-order 8 bits from process's exit value. See the `wait`(2) man page for more information.
	`ac_tty`	Controlling tty device.
Counters	`ac_ctime`	Process raw connect time in clocks. For multitasking jobs, `ac_ctime` is a sum of the connect time across all CPUs used by the job.
	`ac_io`	Number of characters transferred by the process. If any tape accouting information existed for this process, the number of tape bytes read and written is included in the `ac_io` field. Thus, the amount of tape I/O is reported twice: once in the `ac_io` field and again in the tape accounting record. The `ac_io` field generally is larger, because it includes additional I/O performed by the process. This number is repeatable. Device accounting I/O information is also reported twice: by `ac_io` and in the device accounting record field `acd_ioch`. Charges for doing I/O to tape or to a particular device can be adjusted by setting the SBU weight factors for tape and device I/O. These weights are defined in `/etc/config/acct_config`. The tape SBUs are labeled `TAPE_SBU` *x*, and the device SBUs are `BLOCK_DEVICE` *x* and `CHAR_DEVICE` *x*.

Type	Field	Description
		Set the weight factors relative to P_BYTEIO. The ac_io value is multiplied by P_BYTEIO. The tape or device I/O value is multiplied by the appropriate tape or device weight factor. For example, if a surcharge is to be applied to tape I/O, the read and write values for the TAPE_SBU *x* variables must reflect the amount over P_BYTEIO that should be charged.
	ac_iobtim	I/O wait time in clocks measured while the process is not locked in memory (unlike ac_iowtime). System buffer I/O accumulates here. This number may vary due to system load.
	ac_iosw	Swap count. This number may vary due to system load.
	ac_iowtime	I/O wait time (in clocks) measured while the process is locked in memory. This means that system buffered I/O does not appear here. Also, this is a measure of the time elapsed from when a process is removed from the run queue until the process is reconnected to a CPU; therefore, it may vary due to system load.
	ac_lio	Logical I/O request count; this is a count of the read, write, reada, writea, and listio (list entries) system calls made. This number is repeatable.
	ac_rw	Number of physical I/O requests initiated by the process. This number varies due to conditions in the system buffer cache. Therefore, if repeatable billing is desired, this number cannot be used.
	ac_sctime	System call time in clocks.
	ac_stime	System CPU time used (in clocks). This number is not repeatable, because it varies with system load.

Type	Field	Description
	`ac_utime`	User CPU time used (in clocks). For nonmultitasked processes, this number does not include semaphore wait time and is repeatable (within the limitations caused by memory conflicts).
Integrals	`ac_iowmem`	I/O-wait-time memory integral measured while the process is locked in memory (in click-ticks). This number may vary due to system load.
	`ac_mem`	Memory integral selected when `MEMINT = 1` (in clicks-ticks). (`MEMINT` is located in `/etc/config/acct_config`.) This is the only constant memory integral available (within the limitations caused by memory conflicts); therefore, if repeatable billing is required, this number must be used.
	`ac_mem2`	Memory integral selected when `MEMINT = 2` (in clicks-ticks). (`MEMINT` is located in `/etc/config/acct_config`.) This integral is not constant and varies with machine load.
	`ac_mem3`	Memory integral selected when `MEMINT = 3` (in clicks-ticks). (`MEMINT` is located in `/etc/config/acct_config`.) This integral is not constant and varies with machine load.

2.1.11.2 End-of-job Accounting Record

There is one end-of-job record per job. The record is written when the last process of a job is terminated. The record contains the following fields:

Table 3. End-of-job Accounting Record Fields by Function

Type	Field	Description
Header	`ac_header`	Accounting header record word (see `/usr/include/sys/accthdr.h`)
	`ac_flag`	Accounting flags.

Type	Field	Description
Identifiers	`ace_jobid`	Job ID of the job to which this record belongs.
	`ace_uid`	User ID from the job table.
Job Information	`ace_etime`	End time of the job (in seconds).
	`ace_fsblkused`	Sum of the file system storage used. This value may be negative if more space was freed up than was consumed.
	`ace_himem`	High-water memory use of job; sum of all processes in a job at any given time (in clicks). this can vary because of scheduling differences.
	`ace_nice`	Last nice value of the job.
	`ace_sdshiwat`	Secondary data segment high-water use; sum of all processes in a job at any given time (in SDS units). This can vary because of scheduling differences.

2.1.11.3 Multitasking Accounting Record

If a process is multitasked, a multitasking accounting record is written when the last member of the multitasked group is terminated. The record contains the following fields:

Field	Description
`ac_header`	Accounting header record word (see `/usr/include/sys/accthdr.h`)
`ac_flag`	Accounting flags.
`ac_smwtime`	(Not on Cray C90 systems.) Semaphore wait time (in clocks).
`ac_mutime[MUSIZE]`	Time a process was connected to exactly (*i* + 1) CPUs (in 1/100ths of a second format). The CPU time used when the process was connected to (*i* + 1) CPUs is `ac_mutime`[*i*] * (*i* + 1) 1/100ths of a second. For example, `ac_mutime`[1] is the time a process was connected to two CPUs, and `ac_mutime`[1] * 2) is the CPU time used while connected to two CPUs.

ac_mutime[] includes wait semaphore time.

Prior to UNICOS release 8.3, the multitasking CPU times were stored as 21-bit pseudo-floating point numbers. Beginning with release 8.3, these values are in 1/100ths of a second and are compressed as 16-bit pseudo-floating point numbers. The compression and unit changes were made so that multitasking information for a maximum of 32 CPUs can be stored in the pacct file without changing the size of the records.

2.1.11.4 SDS Accounting Record

If a process utilizes SDS, an SDS accounting record is written. The record contains the following fields:

Field	Description
ac_header	Accounting header record word (see /usr/include/sys/accthdr.h)
ac_flag	Accounting flags.
acs_mem	Memory integral based on residency time, not execution time (in click-ticks).
acs_lio	Logical I/O request count; this count is the number of ssread and sswrite system calls made.
acs_ioch	Number of characters transferred to and from the SDS, stored in bytes.

2.1.11.5 MPP Accounting Record

If a process uses a Cray MPP system, an MPP accounting record is written that contains the following fields:

Field	Description
ac_header	Accounting header record word (see /usr/include/sys/accthdr.h)
ac_flag	Accounting flags.
ac_mpppe	Number of MPP processor elements used.

`ac_mppbe`	Number of MPP barrier bits used.
`ac_mpptime`	Number of clocks that the MPP has been used.

2.1.11.6 Performance Accounting Record

When the optional performance accounting feature is enabled (by using the `devacct`(8) command with the `-b` option), a performance accounting record is written at the end of each process. Each record contains the following fields:

Field	Description
`ac_header`	Accounting header record word (see `/usr/include/sys/accthdr.h`)
`ac_flag`	Accounting flags.
`acp_rtime`	The process start time offset (in clocks) from the previous second (reported in the `ac_btime` field of the base accounting record). This field allows you to trace start times of processes that are spawned in the same second.
`acp_tiowtime`	The terminal I/O wait time (in clocks); in other words, the period of time starting when a process performing I/O to a tty or pseudo-tty is removed from the run queue and ending when the process is reconnected to a CPU. This number may vary due to system load.
`acp_srunwtime`	This field is currently disabled.
`acp_swapclocks`	The time (in clocks) that a process spends on the swap device.
`acp_rwblks`	The number of physical blocks transferred by the process using the system buffer I/O interface. This number varies due to conditions in the system buffer cache.

`acp_phrwblks`	The number of physical blocks transferred by the process using the raw I/O interface.

2.1.12 Multitasking Incentives

Some sites may want to provide accounting incentives for the use of multitasking. Several of these are available through the appropriate setting of installation parameters.

2.1.12.1 Memory Integrals

Three different memory integrals are available to the accounting software. The differences among them are important to those sites that want to give incentives for use of multitasking.

Memory integral #1 **-** At each change in memory size, memory integral #1 is incremented by the total CPU time used since the last memory change, times the memory size, as follows:

*MI #1: memory size * (total CPU time since last size change)*

Thus, a program that is connected to two CPUs for some period will pay twice the memory cost for that period. When using memory integral #1, a multitasking program incurs the same charges, no matter how many CPUs it gets. This is helpful if consistent billing is important to your site, but not as helpful if incentives for multitasking are important.

Memory integral #2 **-** The calculations for memory integral #2 are similar to those for #1, except that the increment is the sum of times when at least one CPU was connected, times the memory size, as follows:

*MI #2: memory size * (total time when program was connected*
to at least one CPU since last size change)

A multitasking program pays (in memory charges) only for the first CPU it receives; additional CPUs do not increase the memory charge. Using memory integral #2, a multitasking program can potentially decrease its memory charge by a factor equal to the number of CPUs in the machine. This is an incentive for using multitasking. However, because the amount of time a program is connected to a number of CPUs varies from run to run, memory integral #2 is not consistent. The maximum value for #2 is equal to #1 (if no connect time overlap occurs). Note that this also means that #1 is equal to #2 for single-tasked programs.

Memory integral #3 - Some sites with multi-CPU machines may wish to allow an individual program to use a proportionally large amount of memory only if it is also able to use more than one CPU. For instance, in a four-CPU machine, allowing one program to use 90% of memory may idle some CPUs if the program uses only one CPU.

Memory integral #3 allows the site to control this aspect of CPU use by adding an extra factor into the calculation for memory integral #2. The total memory available to user programs is divided by the number of CPUs to derive the value of “one CPU worth of memory.” The memory size of the program is then divided by the “CPU worth” factor to get the extra factor in memory integral #3, as follows (this extra factor cannot be less than 1):

*MI #3: memory size * (total time when program was connected*
*to at least one CPU since last size change) **
(memory size / "one CPU worth of memory")

Memory integral #3 provides an incentive for single-tasked programs to limit themselves to one CPU worth of memory. Multitasked programs will also pay more in memory charges for lots of memory, but they can reduce their memory charges by using multiple CPUs. However, memory integral #3 is as inconsistent as #2, and it can also affect the memory charges for single-tasked programs.

Note that the changes from #1 to #2 and #2 to #3 are, in a sense, opposite for multitasking programs. The changes from #1 to #2 reward multitasking programs by a factor of up to the number of CPUs. The changes from #2 to #3 penalize large-memory programs by up to the number of CPUs. Thus, if a multitasking program has used all memory (on a four-CPU machine), memory integrals #1 and #3 would be nearly equal, and the value of #2 would be approximately one-quarter the value of #1 or #3.

The accounting software is released with memory integral #2 as the default. The `MEMINT` variable in `/etc/config/acct_config` can be changed to match the site’s needs.

2.1.12.2 Reducing Charges

Another incentive you can provide for the use of multitasking is to reduce the charges for CPU time for multitasking programs. This can be accomplished with weighting factors. The operating system kernel maintains counters of the length of time spent by a user program with one processor connected, two processors connected, and so on.

By default, the charges for a multitasking program would be calculated as follows:

```
sum = 0;
for (i=0; i < ncpu; i++)
    sum += ac_mutime[i] * (i+1);
```

This calculation assumes that all CPU time is charged equally. With the weighting factors, the site can specify, for instance, that a second CPU should be only 75% as expensive as the first CPU. Therefore, a program that gets two CPUs as it executes would have its CPU time charges reduced. Note that, because this charge depends on how much overlap a program gets, charges may vary from execution to execution. However, charges are never more than the full price for all CPUs.

The accounting software is released with all CPUs having a cost of 1. The `MUTIME_WEIGHT` *x* variables, defined in `/etc/config/acct_config`, can be changed to meet the site's needs.

Note that the user time reported by the `time`(1) command is adjusted so that there is no charge for wait-semaphore time. (This is in order to provide consistent CPU time charges.) The multitasking overlap times do not adjust for wait-semaphore times and, hence, may actually calculate to a greater CPU time than the sum of the user times. In this case, the CPU charge is limited to the sum of the user times.

2.1.13 Socket Accounting

Socket accounting tracks network usage from the perspective of sockets, wherein one process may use several sockets, and several processes may use the same socket.

The recorded accounting information tracks all of a socket's usage, but it can only be linked to the process that most recently closed the socket. This information can help you resolve network problems and/or monitor system network usage.

You can use the `csasocket`(8) command to summarize and process the socket data; `csaswitch`(8) can be used to check the status of, enable, and disable accounting methods, including socket accounting. See the `csasocket`(8) and `csaswitch`(8) man pages for more information.

2.1.14 Device Accounting

This section describes device accounting. On large computer systems with expensive peripheral devices, it may be useful to associate device usage with the user who initiated the I/O. Cray device accounting allows a system administrator to specify the accounting data that should be collected for device use. This system allows a site to individually label each mounted disk's partitions and so enables the site to charge each type of secondary storage at a different rate. For example, the amount of I/O on a high-speed storage device such as an SSD may be charged at a different rate than I/O on a disk device.

2.1.14.1 Categories of Devices

The following three categories of devices under the UNICOS operating system are important in device accounting:

- Character special devices, which are devices such as terminals, pseudo tty devices, and the HSX channel.
- Block devices, which are devices such as disks, BMR, and the SSD.
- Logical devices, for device accounting, which are the individual file systems. Such devices do not always correspond to a single device, but are treated as such by device accounting.

The device accounting system accounts for device I/O by device type. For a character device, device type is equivalent to its major number. For example, tty devices are major number 1 (in the default system), so they are accounted for as character device 1 (ios-tty). No accounting is performed for block devices, because block devices are used to create file systems; instead, they are treated as logical devices. Logical devices consist of one or more partitions of disk, SSD, and BMR storage. Each logical device is formatted by the `mkfs`(8) command, which provides it with a superblock. The `devacct`(8) program allows you to write an accounting device type into the superblock of each logical device.

2.1.14.2 Structures and Device Names

The `BLOCK_DEVICE` *x* and `CHAR_DEVICE` *x* parameters in `/etc/config/acct_config` contain the SBU values and names for device accounting. Refer to Section 2.1.10.1.9, page 47, for an explanation of configuring these parameters.

Device accounting uses arbitrary ASCII names for the user interface to accounting; internally, these names are mapped by the accounting library routines `typetonam` and `namtotype`. To be useful, these names should be meaningful to even the beginning user, because the `ja`(1) (job accounting) command displays these names when invoked with the `-d` option. The ASCII names are defined in the device name field of the `BLOCK_DEVICE` *x* and `CHAR_DEVICE` *x* parameters.

Logical device accounting names are displayed to the user by `ja` and the accounting programs, and are used by `devacct`(8) to determine the numeric values the kernel uses.

Logical and character device names should not match; in fact no two names should match, because the user cannot distinguish between them.

If names contain spaces (the shell field separator (`SHELL IFS`), double quotes must be used around the device type names during command invocation.

Device names are used as output by `ja` and the accounting programs; therefore, keeping the names fairly short (less than 40 characters) will make them more readable.

System billing units (SBUs) have the following meanings:

SBU	Description
`Logical I/O Sbu`	The total number of system calls made to this type of device is multiplied by `Logical I/O Sbu` to determine the SBU cost. This value should be nonnegative.
`Characters Xfer Sbu`	The total number of characters transferred to this device type is multiplied by `Characters Xfer Sbu` to determine the SBU cost. This value should be nonnegative.

2.1.14.3 Configuration Changes

The system is released with the character devices configured to match the released configuration; any changes to `/usr/src/uts/c1/cf/devsw.c` should be reflected in the configuration file.

The block device configuration is released with a simple configuration. Several extensions are possible, although some may require altering the values of `MAXBDEVNO` and `MAXCDEVNO`, and rebuilding the system and accounting commands. First, if a site has a special temporary device that is restricted to a set of users, a special type might be placed on that device to allow the billing process to increase the cost of use, offsetting the lower rate of use. Second, SSD or BMR allocated to logical device cache may be reflected in the configuration.

2.1.14.4 System Header Files

The system header files discussed in this section are important in device accounting.

2.1.14.4.1 `param.h` Header File

The values `MAXBDEVNO` and `MAXCDEVNO` are contained in the `/usr/include/sys/param.h` file; they set the maximum size of the accounting structures in the user structure and the maximum size of the accounting data written. It is recommended that they not be increased beyond the current values unless necessary (although making `MAXCDEVNO` smaller and `MAXBDEVNO` larger by the same amounts is acceptable).

`MAXBDEVNO` is the maximum number of block (logical) device accounting types. This number can be changed from the current value of 10.

`MAXCDEVNO` is the maximum number of character device accounting types. This number can be changed from the current value of 35.

2.1.14.4.2 `acct.h` Header File

The `/usr/include/sys/acct.h` header file contains all the kernel structures for accounting and sets the following values related to device accounting:

Value	Description
`NODEVACCT`	The number of `devio` entries per accounting record. This value is the number of device accounting entries that fit into one accounting record.
`ACCT_CHSP`	A marker combined by an `OR` operation into the type field (`acd_type`) to indicate that the `devio` entry is for a character device.

`_MAXDEVIOREC`	The maximum number of device accounting records that can be written for any individual process.

2.1.14.5 Using Device Accounting (`devacct`(8))

Use the `devacct`(8) command to label file systems with accounting types while they are mounted. If a file system does not contain a device type label, device accounting ignores it.

In order to enable device accounting, the system may be built to automatically enable specific device types. However, an easier solution is to insert lines into the system startup procedure (`/etc/config/daemons`) to enable device accounting when bringing the system to multiuser mode. The following example shows a line that can be added to the daemons file (`/etc/config/daemons`) to enable device accounting (remember the device type name is a single argument and so it may need to be enclosed in double quotation marks if it contains shell separators):

```
SYS1 devacct YES - /usr/lib/acct/devacct -b "device type name"
```

The `devacct` command with the `-l` option may be used to label file systems (file systems may be labeled only while mounted). The names of device types are defined in the `BLOCK_DEVICE`*x* and `CHAR_DEVICE`*x* variables located in `/etc/config/acct_config`. Some of the default names include spaces; such names must be enclosed in double quotation marks on the command line.

For example, to label the device `/dev/dsk/root` with the label `"dd49 with ldcache"`, the command would be as follows:

```
/usr/lib/acct/devacct -l "dd49 with ldcache" /dev/dsk/root
```

Device accounting for any device type may be turned on at any time by invoking the `devacct` command with the `-b` option. While device accounting is on, no records are written unless per-process accounting is enabled.

For example, to enable accounting for the devices labeled `"dd49 with ldcache"`, the command is as follows:

```
/usr/lib/acct/devacct -b "dd49 with ldcache"
```

You can turn on performance accounting using the following command:

```
/usr/lib/acct/devacct -b perf01
```

Device accounting for any device type may be turned off at any time by invoking the `devacct` command with the `-t` option. While accounting is disabled, those processes that have already accumulated data will report that data at termination if per-process accounting is enabled. For example, to disable accounting for the devices labeled `"dd49 with ldcache"`, the command is as follows:

```
/usr/lib/acct/devacct -t "dd49 with ldcache"
```

To determine the current label for a device, use the `devacct` command with the `-L` option. For example, to list the current label of `/dev/dsk/root`, you would execute the following command:

```
/usr/lib/acct/devacct -L /dev/dsk/root
```

2.1.14.5.1 Implications of Device Accounting

The system overhead for device accounting is fairly low. However, the amount of accounting data produced in the worst cases is more than double that produced by standard accounting. The more device accounting data kept, the more file system space that is required. If one device is accounted for, processes that use that device generate twice as much accounting data as a process that did not use the device or the same process without device accounting. However, for 1 to NODEVACCT device types, the maximum size of the accounting data does not increase, except that more processes may use one of the devices.

2.1.14.5.2 Tape Device Accounting

To enable or disable tape device accounting, use the *device type name* associated with the CHAR_DEVICE15 parameter in `/etc/config/acct_config`. By default, this device name is `"bmx daemon"`.

The device name associated with CHAR_DEVICE11 (the default is `"bmx tape"`) controls device accounting only for tape diagnostics.

To enable device accounting for the tapes, execute the following command:

```
/usr/lib/acct/devacct -b "bmx daemon"
```

2.1.15 Switching `/` and `/usr` File Systems

Occasionally, sites run on numerous `/` and `/usr` file systems and want to maintain the same accounting files throughout. The easiest way to accomplish this is to put `/usr/adm` or `/usr/adm/acct` on a separate file system and mount this file system along with each different system.

In addition, two other files, `/etc/csainfo` and `/etc/wtmp`, must be copied from the previously booted `/`. These files must be copied to the new root file system before it is brought up. Failure to correctly copy `/etc/csainfo` to the new `/` can cause `csarun` to abort abnormally. Incorrect connect time data is reported when `/etc/wtmp` is not copied.

2.1.16 Logging Information

The following sections describe log files found in the UNICOS operating system.

2.1.16.1 Boot Log

The boot log contains the date and time the system was booted. It is located in `/etc/boot.log` and can be accessed through normal file manipulations such as `tail`(1), `cat`(1), `pg`(1), and `more`(1). The `/etc/rc` (see `brc`(8)) script appends the record to the `boot.log`. The format is as follows:

date, `uname -a`
yy/*mm*/*dd* *hh*:*mm* *system* *node* *release* *version* *hardware*

Example:

```
93/05/10 08:07 sn1703c cool 8.0.0tk dev.6 CRAY Y-MP
```

See `date`(1) and `uname`(1) for further information. See also `who`(1), and the sample `wtmp` and `utmp` files in this chapter.

2.1.16.2 `cron`(8) Log

The `cron` log contains the history of all actions taken by the `cron`(8) command. It is located in `/usr/lib/cron/log` and can be accessed by using normal file manipulations such as `tail`(1), `cat`(1), `pg`(1), and `more`(1). The format of this file is as follows:

```
CMD: command_executed username  process_id  job_type
     start_time username  process_id    job_type
     end_timerc=error return code
```

job_type can have one of the following values:

`a`	`at` job

`b`	Batch job
`c`	`cron` job

Example:

```
>  CMD: 645827040.a
>  user1 7191 a Tue Jun 19 15:24:00 1990
>  CMD: /usr/lib/sa/sa1 120 1
>  root 7192 c Tue Jun 19 15:24:00 1990
<  root 7192 c Tue Jun 19 15:24:00 1990
<  user1 7191 a Tue Jun 19 15:24:00 1990
>  CMD: 645827059.b
>  user1 7273 b Tue Jun 19 15:24:19 1990
<  user1 7273 b Tue Jun 19 15:24:20 1990 rc=1
```

2.1.16.3 dump Log

The dump log contains the time and a reason for each dump. The system supplies the time and the user supplies the reason. By default, the dump is located in `/etc/dump.log` and can be accessed using the normal file manipulations such as `tail`(1), `cat`(1), `pg`(1), and `more`(1). When the system is changing out of single-user mode, `brc`(8) calls `coredd`(8) to copy a dump file to a file system. The location of the dump can be reconfigured by using the install tool. Note that the user may also change the location of the log file by using the `-l` option with the `cpdmp` command.

Example of `/etc/dump.log`:

```
cpdmp: 035120 blocks on dump device - waiting to be copied
04/26/93 07:27:09   coredd: Copying system dump into /core//04260727.
Unicos-E dump copied to=/core//04260727/dump
        dump taken: 04/26/93 at 07:18:51
        reason: PANIC: master.s: EEX interrupt in UNICOS kernel
```

2.1.16.4 New User Log

The new user log contains information on new users given logins on the system; this data includes who added the users, the times at which they were added, and information about their environment defaults and IDs. This log is located in `/usr/adm/nu.log` and can be accessed using normal file manipulations such as `tail`(1), `cat`(1), `pg`(1), and `more`(1). It is created by the `nu`(8) command. An example of the format follows:

```
user1:co:user login #1
user1:ui:10702:di:/j/user1:sh:/bin/csh:dr:/:pw:qQfHS6B8XYdzg
user1:gi:128,129,130,131,132
user1:ai:0
user1:dl:0:mx:0:mn:0:lk:0:tp:0
user1:dc:default:cm:default:pm:default
        added by adm1 on Wed Jul 20 08:43:20 1988
```

2.1.16.5 su(8) Log

The `su` log records `su`(1) attempts for the current day. It is located in the `/usr/adm/sulog` file and can be accessed using normal file manipulations such as `tail`(1), `cat`(1), `pg`(1), and `more`(1). It is written by the `su`(1) command. The format of the log is as follows:

`SU` *MM/DD hh:mm flag tty olduser-newuser*

flag can have the following values:

`+` `su` was successful.

`-` `su` was not successful.

olduser is the login name of the user executing `su`, and *newuser* is the name of the user the executing user is becoming. For example:

```
SU 06/19 15:13 + console operator-root SU 06/19 15:13 + ttyp025 \n
user1-root SU 06/19 15:14 + ttyp021 user2-adm SU 06/19 15:19 - ttyp026 \n
user3-root SU 06/19 15:19 - ttyp022 user4-root
```

2.1.16.6 OLDsu Log

The `OLDsu` log is a directory containing all files of daily `su`(1) attempts. It is located in `/usr/adm/OLDsu/*` and can be accessed using normal file manipulations such as `tail`(1), `cat`(1), `pg`(1), and `more`(1). The `/etc/rc` script moved the `/usr/adm/sulog` file to the `/usr/adm/OLDsu` directory at system startup. An example of the format follows:

```
$ ls -al OLDsu

-rw-rw-rw-  1 root    2864 Oct 31 19:02 Oct31
-rw-rw-rw-  1 root   20211 Sep 12 09:15 Sep01
-rw-rw-rw-  1 root     938 Sep 12 09:15 Sep02

$ cat Sep01

SU 09/01 16:29 + tty?? root-root
SU 09/01 16:30 + tty?? root-sys
SU 09/01 16:32 + tty?? root-sys
SU 09/01 16:32 + tty?? root-root
SU 09/01 16:34 + tty?? root-sys
SU 09/01 16:35 + tty?? root-root
SU 09/01 16:36 + tty?? root-sys
```

2.1.16.7 System Logs

The system logs are files into which the `syslogd`(8) command has logged messages. They are located in the `/usr/adm/syslog/*` directory. Note that these files are described by the configuration file `/etc/syslog.conf`. They can be accessed using normal file manipulations such as `tail`(1), `cat`(1), `page`(1), and `more`(1). They are written by the `/etc/syslogd` command; the `logger`(1B) command also makes entries in the system logs.

These logs consist of ASCII messages, which may include debug messages, kernel messages, and so on.

The following example is the configuration file for `/etc/syslogd` (these fields are described on the `syslogd`(8) and `syslog`(3) man pages):

```
$ cat /etc/syslog.conf

# USMID @(#)man/2302/02.accounting      92.2    02/05/96 13:26:44
#
#       This is a configuration file for /etc/syslogd
#
#*.debug                                /usr/adm/syslog/debug
#
mail.debug                              /usr/spool/mqueue/syslog
#
kern.debug                              /usr/adm/syslog/kern
#
daemon,auth.debug                       /usr/adm/syslog/auth
#
#*.err;kern.debug;auth.notice           /dev/console
#
*.err;kern.debug;daemon,auth.notice;    /usr/adm/syslog/daylog
#
#*.alert;kern.err;daemon.err            operator
*.alert                                 root
```

Note: The `/etc/syslogd.conf` file does not work if spaces are in it; only tabs can be used to separate items in this file.

The following example shows a listing of the files in the `/usr/adm/syslog` directory:

```
$ ls -l /usr/adm/syslog
total 10
-rw-r--r--   1 root     root         168 Jun 19 15:35 auth
-rw-r--r--   1 root     root        5164 Jun 19 15:45 daylog
-rw-r--r--   1 root     root        4107 Jun 19 15:45 kern
drwxr-xr-x   2 root     root       16864 Jun 19 15:09 oldlogs
```

2.1.16.8 Error Log

The error log is a file containing error records from the operating system. The default error file is `/usr/adm/errfile`. There are two facilities available for generating reports from the data collected by the error-logging mechanism. The first is `errpt`(8), which processes error reports from the data, and the second is `olhpa`, a hardware performance analyzer that reports the hardware errors and statuses recorded in the system error log.

Note: The `olhpa` facility is only available on IOS-E based systems. It is not available on GigaRing based systems.

The `/etc/errdemon` command (see `errdemon`(8)) reads `/dev/error` and places the error records from the operating system into either the specified file, or `errfile`, by default. The `/etc/rc` (see `brc`(8)) script starts `/etc/errdemon`, and `/etc/mverr` is used to start a new `errfile`.

The following example shows sample `errpt` output:

```
Tue Jun 7 12:01:49 1988
       Error reported from IOS 0 for device S49-0-21
       Major:0  Minor:6         Block:140868   status: Recovered
       Iop:0  Channel:21        Unit:0
       Cylinder:1156  Head:5    Sector:0
       Function:Read  Requested:344064 bytes   Received:344064 bytes

       IOS 0 ERROR LOGGING ENABLED
```

See `errpt`(8) for further information. See the *Online Maintenance Tools Guide for Cray PVP Systems*, or the `olhpa`(8) man page for information concerning `olhpa`. [2]

2.1.16.9 Multilevel Security (MLS) Log

The multilevel security (MLS) log is a file containing security-relevant event loggings. The security log, `/usr/adm/sl/slogfile`, can be analyzed by using the `reduce` command. `reduce` extracts, formats, and outputs entries from UNICOS security event files. The security event logging daemon, `slogdemon`(8), collects security-relevant records from the operating system by reading the character special pseudo device `/dev/slog`. For more information regarding the format of the security log and on the UNICOS MLS feature, see the `reduce`(8) man page and *General UNICOS System Administration*.

2.1.16.10 System Activity Log

The system activity report facility provides commands for generating various system activity reports. Two reporting capabilities exist (one automatic and one user-driven); however, the actual reports are created by the `sar`(8) program in either case. The system activity log is located in `/usr/adm/sa/sa`*DD* and can be accessed with `sar`.

[2] CRAY PRIVATE. This document contains information private to Cray Inc. It can be distributed to non-Cray personnel only with approval of the appropriate Cray manager.

Warning: The log files located in /usr/adm/sa/sa*DD* on a Cray ML-Safe configuration of the UNICOS system are considered to be covert channels. You may want to consider restricting access to these files to the adm group.

With this command, you can generate system activity reports in real time and save system activities in a file for later use. The sa1, sa2, and sadc commands (see sar(8)) generate system activity data on a routine basis, with sa2 calling sar to generate the report.

UNICOS counters are incremented as various system actions occur. These counters provide system-wide measurements. sadc accesses /dev/kmem to read these system activity counters.

Refer to the sar(8) man page for more information on the format of the system activity log.

2.1.16.11 Message Log

The message log contains messages and replies to the operator. It is located in /usr/spool/msg/msglog.log and can be accessed using normal file manipulations, such as tail(1), cat(1), pg(1), and more(1). All messages and replies to and from the operator console are put into this file by the console. An example of the file format follows:

```
Apr  1 07:11:06  Message daemon stopped
Apr  1 09:36:54  Message daemon started
Apr  1 08:09:49  Message   1: TM122 - mount tape WK1102(sl) on a CART
 device for user1 50,  () or reply cancel / device name
```

Warning: The msglog.log file is considered a covert channel on a Cray ML-Safe configuration of the UNICOS system. You may want to consider restricting access to this file to the adm group.

2.1.16.12 Accounting Logs

The accounting logs are files containing various accounting information, as follows:

Log	Description
`csainfo`	A file containing boot times. It can be accessed with the `od`(1) command (the `-d` option will give the seconds). Each time the system is booted, the boot time is written to `/etc/csainfo` by the `/etc/csaboots` (see `csaboots`(8)) command. `csaboots` is invoked by `/etc/rc` (see `brc`(8)). See also the description of the boot log in Section 2.1.16.1, page 71.
`utmp`	A file containing active system and terminal connection information. This log is used by `write`(1), `who`(1), `wall`(8), and `mail`(1) in getting user information. It is located in `/etc/utmp` and can be accessed using the `who`(1) and `last`(1B) commands. It is written to by `init`(8), `date`(1), `login`(1), and `getty`(8). For information on the format of `utmp`, see `utmp`(5). **Warning:** On a Cray ML-Safe configuration of the UNICOS system, `utmp` and `wtmp` are considered to be covert channels. You may want to consider restricting access to these files to the `adm` group.
`wtmp`	A file containing a system and terminal connection history record. This log includes usage statistics for each terminal, date change, time stamp, boot records, reboots, shutdowns, and state changes. `wtmp` must exist; programs that access it do not create it (the `/etc/rc` script creates `/etc/wtmp` by default). Records are in the form of `utmp`(5); `acctcon`(8) and `csaline`(8) convert `/etc/wtmp` into session and charging records. This data is merged into the system accounting reports. `wtmp` can also be accessed using the `who`(1) and `last`(1) commands. `wtmp` is written by `init`(8), `date`(1), `login`(1), and `getty`(8). For information on the format of `wtmp`, see `utmp`(5).
`pacct`	Files containing per-process accounting data; these are located in `/usr/adm/acct/day/pacct*` and can be accessed using the `acctcom`(1) command. Note that these files are read by system accounting programs, and the information appears in the accounting reports. `pacct` is written by the kernel, and its format is described in `/usr/include/sys/acct.h`.

Warning: On systems running a Cray ML-Safe configuration of the UNICOS system, access to `pacct*` files should be restricted to the `adm` group.

The following data files are accessed by system accounting programs, and their information appears in the accounting reports:

Log	Description
`disktacct`	A file containing disk accounting data, located in `/usr/adm/acct/nite/disktacct`. The `/usr/lib/acct/dodisk` (see `dodisk`(8)) command writes this file.
`fee`	A file containing user fees for accounting data, located in `/usr/adm/acct/day/fee`. This file is written by `/usr/lib/acct/chargefee` (see `chargefee`(8)).
`nqacct`	A file containing NQS daemon accounting data, located in `/usr/adm/acct/day/nqacct*`. This file is written by `/usr/lib/nqs/nqsdaemon`. See `/usr/include/acct/dacct.h` for the format.
`soacct`	A file containing socket accounting data, located in `/usr/adm/acct/day/soacct*`. This file is written by the kernel. See `/usr/include/sys/acct.h` for the format.
`tpacct`	A file containing tape daemon accounting data, located in `/usr/adm/acct/day/tpacct*`. This file is written by `/usr/lib/tp/tpdaemon` (see `tpdaemon`(8)). See `/usr/include/acct/dacct.h` for the format.

2.1.16.13 NQS Log

The NQS log contains NQS information. Its default location is the ASCII file `/usr/spool/nqs/log` (you can change the location of the log file with the `qmgr set log_file` command; to see where the current log file resides, use the `qmgr show parameters` command). Access to `/usr/spool/nqs` is restricted; however, if you have the correct permissions, you can access the NQS log file using normal file manipulations, such as `tail`(1), `cat`(1), `pg`(1), and `more`(1). This log is created by the NQS log daemon.

Warning: On systems running a Cray ML-Safe configuration of the UNICOS system, access to the NQS log should be restricted to the adm group.

An example of the log file's format is as follows:

```
05/13 08:00:00 I getpkt(): Received packet from local process: <89775>.
05/13 08:00:00 I getpkt(): Client process real UID=<900>.
05/13 08:00:00 I getpkt(): Packet type=<PKT_QUEREQVLPQ(30)>.
05/13 08:00:00 I getpkt(): Packet contents are as follows:
05/13 08:00:00 I getpkt(): Pkt_str[1] = <batnam1>.
05/13 08:00:00 I getpkt(): Pkt_int[1] = <40>.
05/13 08:00:00 I getpkt(): Pkt_int[2] = <119>.
05/13 08:00:00 T nqs_quereq(): Request <40.cool>: Attempting to read request.
05/13 08:00:00 T nqs_quereq(): Request <40.cool>: Request was read.
```

2.2 Standard UNIX Accounting

The standard UNIX accounting feature of the UNICOS system provides methods for collecting resource use data per process, recording connect sessions, monitoring disk usage, and charging fees to specific logins. A set of C language programs and shell procedures is provided to reduce this accounting data into summary files and reports. This section describes the structure, implementation, and management of the accounting system; it also describes the reports generated and the meaning of the columnar data.

The following list is a synopsis of the standard accounting actions:

- At process termination, the UNICOS system kernel writes one record per process in /usr/adm/acct/day/pacct.
- The login(1) and init(8) programs record connect sessions by writing records into /etc/wtmp. Date changes, reboots, and shutdowns are also recorded in this file. The wtmp file is described in utmp(5).
- The programs acctdusg(8) and diskusg(8) break down disk usage by login.
- Fees can be charged to specific logins with the chargefee(8) shell procedure.
- Each day the cron(8) shell procedure executes the runacct(8) shell procedure, which reduces accounting data and produces summary files and reports.

- The `monacct`(8) procedure can be executed on a monthly or fiscal period basis. It saves and restarts summary files, generates a report, and cleans up the `sum` directory. These saved summary files could be used to charge users for UNICOS system usage.

2.2.1 Files and Directories

The `/usr/lib/acct` directory contains all the C language programs and shell procedures necessary for running the accounting system. The `adm` login is used by the accounting system and has the login directory structure shown in Figure 3.

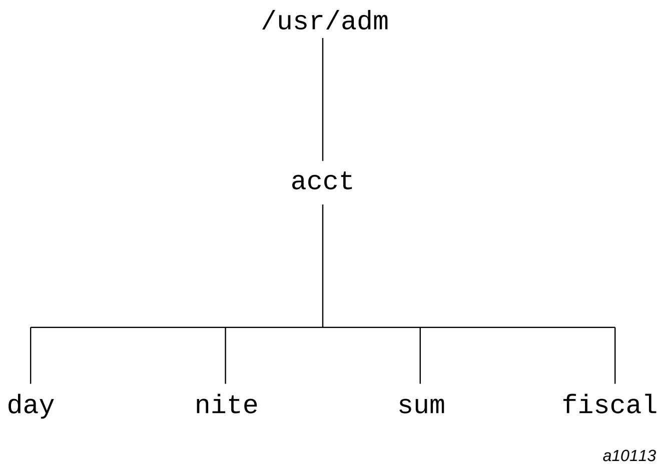


Figure 3. Directory Structure of the Adm Login

The `/usr/adm/acct/day` directory contains the active data collection files. The `nite` directory contains files that are reused daily by the `runacct`(8) procedure. The `sum` directory contains the cumulative summary files updated by `runacct`(8). The `fiscal` directory contains periodic summary files created by `monacct`(8).

In addition, configurable parameters are located in `/etc/config/acct_config`. You should modify these parameters to meet your site’s needs.

2.2.2 Daily Operation

When the UNICOS system is switched into multiuser mode, `/usr/lib/acct/startup` is executed, as follows:

1. The `acctwtmp`(8) program adds a boot record to `/etc/wtmp`. This record is signified by use of the system name as the login name in the `wtmp` record.
2. Process accounting is started with `turnacct`(8). The `turnacct` command specified with the `on` argument, as follows, executes the `accton`(8) program with the `/usr/adm/acct/day/pacct` argument:

   ```
   /usr/lib/acct/turnacct on
   ```
3. The `remove` shell procedure is executed to clean up the saved `pacct` and `wtmp` files left in the `sum` directory by `runacct`(8).

The `ckpacct`(8) procedure is run with `cron`(8) every hour to check if there is enough space on the current file system (the default is `/usr`). If there are fewer than `MIN_BLKS` free blocks (by default 500), accounting is stopped, and the system administrator is notified about the action. `MIN_BLKS` is defined in the configuration file `/etc/config/acct_config`. The `ACCT_FS` variable in `/etc/config/acct_config` must be set to the file system containing `/usr/adm/acct`. If the free space increases to 500 free blocks at a later time, accounting is restarted, again with notification to the system administrator.

You can use the `chargefee`(8) program to bill users. It adds to `/usr/adm/acct/day/fee` records that are picked up and processed by the next execution of `runacct` and merged into the total accounting records.

The `runacct` command is executed with `cron` each night. It processes the following active accounting files:

```
/usr/adm/acct/day/pacct
/etc/wtmp
/usr/adm/acct/day/fee
/usr/adm/acct/nite/disktacct
```

It produces command summaries and usage summaries by login. When the system is shut down with `shutdown`(8), the `shutacct`(8) shell script is executed. It writes a shutdown reason record into `/etc/wtmp` (see `utmp`(5)) and turns process accounting off.

2.2.3 Setting up the Accounting System

This section explains how to automate the operation of the accounting system. It also contains information on converting UNICOS 8.0, 8.3, 9.0, 9.1, 9.2, and 9.3 standard UNIX accounting files to UNICOS 10.0 CSA format.

To automate the operation of the accounting system, complete the following steps:

1. Modify any necessary parameters in the file `/etc/config/acct_config`, which contains configurable parameters for the accounting system. Ensure that the parameters, such as `MEMINT`, reflect the needs of your site. You can specify an alternate configuration file when running any of the accounting commands. See Section 2.1.10.8, page 55, for more information.

2. If you maintain startup options with the Installation Configuration Menu System (ICMS), configure `RC_ACCT` to have a value of `YES`. Otherwise, edit the `/etc/config/rcoptions` file to set `RC_ACCT` to a `YES` value.

3. Add an entry similar to the following to `/usr/spool/cron/crontabs/root` so that `cron` automatically runs `dodisk`:

   ```
   0 2 * * 4 /usr/lib/acct/dodisk
   ```

   `dodisk` must be executed by `root`, because no other user has the correct permissions to read `/dev/dsk/*`.

4. For most installations, you should make entries similar to the following in `/usr/spool/cron/crontabs/adm` so that `cron` will run the daily accounting automatically:

   ```
   0 4 * * 1-6 /usr/lib/acct/runacct 2>/usr/adm/acct/nite/fd2log
   50 * * * * /usr/lib/acct/ckpacct
   ```

   The `runacct`(8) command should be run at a time when the `dodisk`(8) routine has had sufficient time to complete. If `dodisk` has not completed before `runacct` executes, disk information may be missing.

5. To facilitate monthly merging of accounting data, make an entry similar to the following in `/usr/spool/cron/crontabs/adm`:

   ```
   15 5 1 * * /usr/lib/acct/monacct
   ```

   This entry allows the `monacct`(8) procedure to clean up all daily reports and daily total accounting files and to deposit one monthly total report and one monthly total accounting file in the `fiscal` directory. It takes advantage of the default action of monacct, which uses the current month's date as the suffix for the file names. The entry is executed when the runacct(8) procedure has sufficient time to complete. This results in the creation of monthly accounting files on the first day of each month containing the entire previous month's data.

6. Set the PATH shell variable in /usr/adm/.profile to the following:

```
PATH=/usr/lib/acct:/bin:/usr/bin
```

2.2.3.1 Setting up a User Exit

Daily accounting provides one user exit, /usr/lib/acct/run.user, that you can call from the runacct command. This user exit allows you to tailor the runacct procedure to your site's needs by creating a shell script to perform any additional processing during the daily run of accounting. You do not have to modify the runacct script.

While executing, runacct checks in the USEREXIT state for a shell script named /usr/lib/acct/run.user. If the script exists, it is executed via the shell . (dot) command. If the script does not exist, the user exit is ignored. The . (dot) command will not execute a compiled program, but the user exit script can. runacct variables are available, without being exported, to the user exit script. runacct checks the return status from the user exit and, if it is nonzero, the execution of csarun is terminated.

2.2.3.2 Converting Standard UNIX Accounting to CSA Accounting

If your site decides to run CSA instead of standard UNIX accounting, you should wait until the start of an accounting period before implementing CSA. (An accounting period usually begins on the first day of a month.) Before switching to CSA, use the standard UNIX accounting package to process the previous month's accounting data.

Follow these steps to convert from standard UNIX accounting to CSA:

1. Run the current version of UNICOS standard UNIX accounting programs until the first day of the next month. Use the runacct(8) command to process the daily accounting data.
2. On the first day of the month, use the monacct(8) command to generate an accounting report for the previous month.
3. On the first day of the month, switch from running the standard UNIX accounting package to CSA.

4. (Optional step) The daily `tacct` files must be converted to `cacct` format if you later want to summarize this data by using `csaperiod`(8). The conversion should be done by using the `csaconvert`(8) command. Refer to the `csaconvert`(8) man page and the UNICOS Installation Guide, publication SG-2112, for more information on conversion.

For details on how to set up CSA, see Section 2.1.4, page 11.

2.2.4 The `runacct` Command

The `runacct`(8) command is the main daily accounting shell procedure. It processes connect, fee, disk, and process accounting files and prepares daily and cumulative summary files for use by `prdaily`(8) or for billing purposes. `runacct` also contains one user exit point that allows you to tailor the daily accounting run to your site's needs. It is normally initiated with the `cron`(8) command during nonprime hours.

The following files in `/usr/adm/acct`, which are produced by `runacct`, are of particular interest:

File	Description
`nite/daytacct`	The total accounting file for the previous day in `tacct.h` format.
`nite/lineuse`	Produced by `acctcon`(8). It reads the `wtmp` file and produces usage statistics for each terminal line on the system. This report is not especially useful, but is a carryover from traditional UNIX systems.
`sum/cms`	The accumulation of each day's command summaries. It is restarted by the execution of `monacct`(8). The ASCII version of this file is `nite/cms`.
`sum/daycms`	Produced by the `acctcms`(8) program. It contains the daily command summary. The ASCII version of this file is `nite/daycms`.
`sum/loginlog`	Produced by the `lastlogin`(8) shell procedure. This file contains a record of the last time each login was used.
`sum/rprt`*MMDD*	Each execution of `runacct`(8) saves a copy of the daily report as produced by `prdaily`(8).

`sum/tacct`	The accumulation of each day's `nite/daytacct`. It can be used for billing purposes and is restarted each month or fiscal period by the `monacct`(8) procedure.

The `runacct` command does not damage files in the event of errors. It contains a series of protection mechanisms that attempt to recognize an error, provide intelligent diagnostics, and terminate processing in such a way that `runacct` can be restarted with minimal intervention.

The `runacct` command records its progress by writing descriptive messages into the file `active`. (Files used by `runacct` are assumed to be in the `/usr/adm/acct/nite` directory unless otherwise noted.) All diagnostic output during the execution of `runacct` is written into `fd2log`. `runacct` terminates execution if the `lock` and `lock1` files exist when it is invoked. The `lastdate` file contains the month and day `runacct` was last invoked and is used to prevent more than one execution per day. If `runacct` detects an error, it writes a message to `/dev/console`, sends mail to `root` and `adm`, removes locks, saves diagnostic files, and terminates execution.

Processing is broken down into separate reentrant states so that `runacct` can be restarted. The last state completed is recorded in a file. As each state completes, `statefile` is updated to reflect the next state. When `runacct` reaches the `CLEANUP` state, it removes the locks and terminates. States are executed as follows:

State	Description
`SETUP`	The `turnacct`(8) command switch is executed. The process accounting files, `/usr/adm/acct/day/pacct*`, are moved to `/usr/adm/acct/day/Spacct*`.*MMDD*. The `/etc/wtmp` file is moved to `/usr/adm/acct/nite/wtmp.`*MMDD*, with the current date added at the end.
`WTMPFIX`	The `wtmpfix` (see `fwtmp`(8)) program checks the `wtmp` file in the `nite` directory for accuracy. Some date changes cause `acctcon1` (see `acctcon`(8)) to fail, so `wtmpfix` attempts to adjust the time stamps in the `wtmp` file if a date change record appears.

	If `wtmpfix` is unable to fix the `wtmp` file, the `wtmp` file must be manually repaired. Refer to Section 2.2.5.1, page 90.
`CONNECT1`	Connect session records are written to `ctmp` in the form of `ctmp.h`. The `lineuse` file and the `reboots` file are created, showing all of the boot records found in the `wtmp` file.
`CONNECT2`	The `ctmp` file is converted to `ctacct.`*MMDD*, which is comprised of connect accounting records. (Accounting records are in `tacct.h` format.)
`PROCESS`	The `acctprc1` and `acctprc2` programs (see `acctprc`(8)) are used to convert the process accounting files, `/usr/adm/acct/day/Spacct*.`*MMDD*, into total accounting records in `ptacct*.`*MMDD*. The `Spacct` and `ptacct` files are correlated by number so that, if `runacct` fails, the `Spacct` files are not reprocessed. One precaution should be noted: when restarting `runacct` in this state, remove the last `ptacct` file, because it will not be complete.
`MERGE`	The process accounting records are merged with the connect accounting records, the output going to `daytacct`.
`FEES`	Any ASCII `tacct` records from the file `fee` are merged into `daytacct`.
`DISK`	On the day after the `dodisk`(8) procedure runs, `disktacct` is merged with `daytacct`.
`MERGETACCT`	The `daytacct` file is merged with `sum/tacct`, the cumulative total accounting file. Each day, `daytacct` is saved in `sum/tacct.`*MMDD* so that `sum/tacct` can be recreated if it becomes corrupted or lost.
`CMS`	Today’s command summary is merged with the cumulative command summary file `sum/cms`. ASCII and internal format command summary files are produced.

`USEREXIT`	User exit point. If a script named `/usr/lib/acct/run.user` exists, it will be executed via the shell . (dot) command. The . (dot) command will not execute a compiled program, but the user exit script can. `runacct` variables are available, without being exported, to the user exit script. You might use this user exit to run local accounting programs.
`CLEANUP`	Clean up temporary files, run `prdaily`(8) and save its output in `sum/rprt`*MMDD*, remove the locks, and then exit.

2.2.4.1 Failure Recovery for `runacct`(8)

The `runacct`(8) program can fail for a variety of reasons; the most common reasons are a system crash, a lack of space in the file system containing `/usr/adm/acct`, and a corrupted `wtmp` file. If the `active` *MMDD* file exists, check it first for error messages. If the `active` file and lock files exist, check `fd2log` for messages.

The following are error messages produced by `runacct` and the recommended recovery actions:

```
ERROR: locks found, run aborted
```

The `lock` and `lock1` files were found. These files must be removed before `runacct` can restart.

```
ERROR: acctg already run for date:  check
/usr/adm/acct/nite/lastdate
```

The date in `lastdate` and today's date are the same. Remove `lastdate`.

```
ERROR: turnacct switch returned rc=?
```

Check the integrity of `turnacct`(8) and `accton`(8). The `accton` program must be owned by `root`, and the `setuid` bit must be set.

```
ERROR: Spacct?.MMDD already exists
```

File setups have probably already been run. Check status of files, then run setups manually.

```
ERROR: /usr/adm/acct/nite/wtmp.MMDD already exists, run
setup manually.
```

This message is self-explanatory.

```
ERROR: wtmpfix errors see /usr/adm/acct/nite/wtmperror
```

The `wtmpfix`(8) program detected a corrupted `wtmp` file. Use `fwtmp`(8) to correct the corrupted file.

```
ERROR: Connect acctg failed:  check /usr/adm/acct/nite/log
```

The `acctcon1`(8) program encountered a bad `wtmp` file. Use `fwtmp` to correct the bad file.

```
ERROR: Invalid state, check /usr/adm/acct/nite/active
```

The `statefile` file is probably corrupted. Check `statefile` and read the `active` file before restarting.

2.2.4.2 Restarting `runacct`(8)

If you invoke `runacct`(8) without arguments, the invocation is assumed to be the first one of the day. The *MMDD* argument is necessary if `runacct` is being restarted. It specifies the month and day for which `runacct` is to rerun the accounting. The entry point for processing is based on the contents of `statefile`. To override `statefile`, include the desired state on the command line. For each case, see the appropriate example, as follows:

To start `runacct`:

```
nohup runacct 2> /usr/adm/acct/nite/fd2log&
```

To restart `runacct` using the state specified in `statefile`:

```
nohup runacct 0601 2> /usr/adm/acct/nite/fd2log&
```

To restart `runacct` at a specific state, overriding `statefile`:

```
nohup runacct 0601 WTMPFIX 2> /usr/adm/acct/nite/fd2log&
```

2.2.5 Fixing Corrupted Files

When file corruption occurs, some files can be ignored or restored from the file save backup. Certain files, however, must be fixed in order to maintain the integrity of the accounting system.

2.2.5.1 Fixing `wtmp` Errors

The `wtmp` files generally cause the highest number of errors in the day-to-day operation of the accounting system. When the date is changed, and the UNICOS system is in multiuser mode, a set of date change records is written into the `/etc/wtmp` file. The `wtmpfix` program (see `fwtmp`(8)) is designed to adjust the time stamps in the `wtmp` records when a date change is encountered.

Some combinations of date changes and reboots, however, slip through `wtmpfix` and cause `acctcon1` (see `acctcon`(8)) to fail.

The following example shows how to repair a `wtmp` file:

```
$ cd /usr/adm/acct/nite
$ /usr/lib/acct/fwtmp < wtmp.MMDD > xwtmp
$ ed xwtmp
  (Delete corrupted records)
$ /usr/lib/acct/fwtmp -ic < xwtmp > wtmp.MMDD
  (Restartrunacct  at the WTMPFIX state)
```

If the `wtmp` file is beyond repair, create a null `wtmp` file, which prevents any charging of connect time. The `acctprc1` program (see `acctprc`(8)) cannot determine which login owned a particular process, but the process is charged to the first login in the `/etc/udb` file for that user ID.

2.2.5.2 Fixing `tacct` Errors

If your installation is using the accounting system to charge users for system resources, the integrity of `sum/tacct` is quite important. Occasionally, `tacct` records appear with negative numbers, duplicate user IDs, or a user ID of `65535`. First, check the `sum/tacctprev` file with `prtacct`(8). If it looks correct, the latest `sum/tacct.` *MMDD* should be corrected; `sum/tacct` must then be recreated. A correctional procedure is as follows:

```
$ cd /usr/adm/acct/sum
$ /usr/lib/acct/acctmerg -v tacct.MMDD xtacct
$ ed xtacct
  (Remove the bad records, write duplicate user ID records to another file)
$ /usr/lib/acct/acctmerg -i xtacct tacct.MMDD
$ /usr/lib/acct/acctmerg tacctprev tacct.MMDD tacct
```

The `monacct`(8) procedure removes all `tacct.` *MMDD* files; therefore, you can recreate `sum/tacct` by merging these files.

2.2.6 Updating Holidays

The `/usr/lib/acct/holidays` file contains the `prime/nonprime` time table for the accounting system. You should edit the table to reflect your site's holiday schedule for the year. By default, the `holidays` file is located in the `/usr/lib/acct` directory. You can change the location of this file by modifying the `HOLIDAY_FILE` variable in `/etc/config/acct_config`. If necessary, you should modify the `NUM_HOLIDAYS` variable (also located in `acct_config`), which sets the upper limit on the number of holidays that can be defined in `HOLIDAY_FILE`.

The format is composed of three types of entries:

1. Comment lines: These lines may appear anywhere in the file as long as the first character in the line is an asterisk.

2. Year and time designation line: This line should be the first data line (noncomment line) in the file and must appear only once. The line consists of three fields of 4 digits each (leading white space is ignored). For example, to specify the year as 1982, prime time at 9:00 A.M., and nonprime time at 4:30 P.M., the following entry would be appropriate:

   ```
   1982 0900 1630
   ```

   As a special condition for the time field, the time `2400` is automatically converted to `0000`.

3. Company holidays lines: These entries follow the year designation line and have the following general format:

   day-of-year Month Day Description of Holiday

   The day-of-year field is a number in the range of 1 through 366, indicating the day for a given holiday (leading white space is ignored). The other three fields are commentary and are not currently used by other programs.

2.2.7 Reports

The `runacct`(8) program generates five basic reports upon each invocation. These reports cover the areas of connect accounting, usage by user on a daily basis, command usage reported by daily totals, command usage reported by monthly totals, and last login time by user. The `diskusg` command can be configured at your site; see Section 2.1.10.9, page 56, for a description of how to customize a report for your site.

The following sections describe the reports and interpretation of their tabulated data.

2.2.7.1 Daily Report

In the first part of the report, the from / to banner alerts you to the time period being reported. The specified times are the time the last accounting report was generated until the time the current accounting report was generated. This banner is followed by a log of system reboots, shutdowns, power failure recoveries, and any other record dumped into the `/etc/wtmp` file by the `acctwtmp`(8) program.

The second part of the report is a breakdown of line usage. The `TOTAL DURATION` value is the difference between the time stamps of the first and the last record found in the `wtmp` file. The columns are as follows:

Column	Description
`LINE`	The terminal line or access port
`MINUTES`	The total number of minutes the line was in use during the accounting period
`PERCENT`	The total number of `MINUTES` the line was in use, divided into the `TOTAL DURATION`
`#SESS`	The number of times this port was accessed for a `login`(1) session

2.2.7.2 Daily Usage Report

The daily usage report gives a breakdown of system resource usage by user. Its data consists of the following:

Heading	Description
`ACCOUNT NAME`	If the UNICOS user-information database is enabled, this field contains the account name; otherwise, it contains `default`.
`UID`	User ID.
`LOGIN NAME`	Login name of the user; there can be more than one login name for a single user ID (although this is not recommended); this identifies the user.
`CPU SECS`	The amount of time in seconds the user’s process used the CPU.

`KCORE-MINS`	A cumulative measure of the amount of memory a process used while running. The amount shown reflects kiloword segments multiplied by minutes used.
`CONNECT (MINS)`	The real time used. Real time is the amount of time that a user was logged in to the system. If this time is rather high, and column `# OF PROCS` is low, this person probably logs in first thing in the morning and rarely uses the terminal the rest of the day. This type of user can be a system resource problem. If this user is logged in and is not using the system at all, he or she may be using a line to the system that someone else needs.
`DISK BLOCKS`	Output from the disk accounting programs after that output has been merged into the total accounting record (`tacct.h`). Disk accounting is accomplished by the `acctdusg`(8) program.
`# OF PROCS`	The number of processes invoked by the user. Large numbers indicate an uncontrolled user shell procedure.
`# OF JOBS`	Number of times the user logged in to the system (interactive or batch).
`# DISK SAMPLES`	Number of times disk accounting was run to obtain the average number of `DISK BLOCKS` listed earlier.
`FEE`	The total accumulation of billing units charged against the user by the `chargefee`(8) shell procedure. The `chargefee` procedure is used to levy charges against a user for special services (such as file restores) performed. This field is often unused.
`SBU`	A site-specific system billing unit (SBU); default is 0. You can modify the SBU calculation for your

site by editing the source and recompiling the accounting software (see Section 2.1.10.1, page 40).

2.2.7.3 Daily Command and Monthly Total Command Summaries

The daily command and monthly total command summaries are virtually the same, except that the daily command summary reports only on the current accounting period, while the monthly total command summary reports on the time from the start of the fiscal period to the current date. That is, the monthly report reflects the data accumulated since the last invocation of the `monacct`(8) procedure.

The data included in these reports tells you which commands are used most often. Based on this information, you can identify areas of the system using a majority of system resources.

These two reports are sorted by `TOTAL CPU-MIN`. The following categories are used:

Heading	Description
`COMMAND NAME`	The name of the command. All shell procedures are under the name `sh`, because only object modules are reported by the process accounting system. The `acctcom`(1) program is a good tool to use for identifying a user who executed a suspiciously named command and also for determining whether super-user privileges were used.
`NUMBER CMDS`	The total number of invocations of this particular command.
`TOTAL KCOREMIN`	The total cumulative measurement of the number of kiloword segments of memory used by a process per run-time minute.
`TOTAL CPU-MIN`	The total processing time this program has accumulated.
`TOTAL REAL-MIN`	The total real-time (wall-clock) minutes this program has accumulated.
`MEAN SIZE-K`	The mean of the `TOTAL KCOREMIN` over the number of invocations reflected by `NUMBER CMDS`.

MEAN CPU-MIN	The mean derived between the NUMBER CMDS and TOTAL CPU-MIN.
HOG FACTOR	A relative measurement of the ratio of system availability to system usage. It is computed by the following formula: `(total CPU time) / (elapsed time)` This gives a relative measure of the total available CPU time consumed by the process during its execution.
K-CHARS TRNSFD	The total number of characters moved by the read(2) and write(2) system calls.
I/O BUFS RD/WR	The total number of physical reads and writes that a process performed.

2.2.7.4 Last Login Report

The last login report provides the date on which a particular login was last used. You can use this report as a source of likely candidates to be moved to the archives, or, of unused logins and login directories to be deleted.

2.2.8 Accounting Files

This section lists files relevant to the accounting system in the /usr/adm/acct/day, /usr/adm/acct/nite, /usr/adm/acct/sum, and /usr/adm/acct/fiscal directories.

Files in the /usr/adm/acct/day directory are as follows:

File	Description
dtmp	Output from the acctdusg(8) program.
fee	Output from the chargefee(8) program (ASCII tacct records).
pacct	Active process-accounting file.
pacct*	Process-accounting files switched using turnacct(8).

`Spacct*.`*MMDD*	Process-accounting files for *MMDD* during execution of `runacct`(8).

Files in the `/usr/adm/acct/nite` directory are as follows:

File	Description
`active`	Used by `runacct` to record progress and print warning and error messages. The `active`*MMDD* file is the same as `active` after `runacct` detects an error.
`cms`	ASCII total command summary used by `prdaily`(8).
`ctacct.`*MMDD*	Connect accounting records in `tacct.h` format.
`ctmp`	Output of `acctcon1` program (see `acctcon`(8)); connect session records in `ctmp.h` format.
`daycms`	ASCII daily command summary used by `prdaily`.
`daytacct`	Total accounting records for one day in `tacct.h` format.
`disktacct`	Disk accounting records in `tacct.h` format; created by `dodisk`(8) procedure.
`fd2log`	Diagnostic output during execution of `runacct`.
`lastdate`	Last day `runacct` executed in *date* +% *m%* `d` format.
`lineuse`	The tty line usage report used by `prdaily`.
`lock lock1`	Used to control serial use of `runacct`.
`log`	Diagnostic output from `acctcon1`.
`log` *MMDD*	Same as log after `runacct` detects an error.
`reboots`	The beginning and ending dates from `wtmp`, and a listing of reboots.
`statefile`	A record of the current state during execution of `runacct`.
`tmpwtmp`	The `wtmp` file corrected by `wtmpfix` (see `fwtmp`(8)).
`wtmperror`	Place for `wtmpfix` error messages.

`wtmperror`*MMDD*	Same as `wtmperror` after `runacct` detects an error.
`wtmp.`*MMDD*	Previous day's `wtmp` file.

Files in the `/usr/adm/acct/sum` directory are as follows:

File	Description
`cms`	Total command summary file for current fiscal year in internal summary format.
`cmsprev`	Command summary file without latest update.
`daycms`	Command summary file for yesterday in internal summary format.
`loginlog`	Login record file created by `lastlogin`(8).
`pacct.`*MMDD*	Concatenated version of all `pacct` files for *MMDD*; removed after reboot by `remove`(8) procedure.
`rprt`*MMDD*	Saved output of `prdaily`(8) program.
`tacct`	Cumulative total accounting file for current fiscal period.
`tacctprev`	Same as `tacct` without latest update.
`tacct`*MMDD*	Total accounting file for *MMDD*.
`wtmp.`*MMDD*	Saved copy of `wtmp` file for *MMDD*, removed after reboot by `remove`(8) procedure.

Files in the `/usr/adm/acct/fiscal` directory are as follows:

File	Description
`cms`	Total command summary file for the fiscal period in internal summary format.
`fiscrpt`	Report similar to `prdaily`(8) for fiscal period.

`tacct*`	Total accounting file for fiscal period.

2.3 Front-end Formatting

Front-end formatting facilities let you customize accounting reports and generate output files that can be processed on a front-end computer system. The front-end formatting process consists of two main parts:

- Consolidating the accounting data you have collected to select useful information and to reduce it to a manageable amount of data for the front-end system.
- Formatting the consolidated data into meaningful reports and files for further processing on the front-end system.

Accounting data is consolidated using identifier keys. These keys may include user ID (`uid`), account ID (`acid`), job ID (`jid`), group ID (`gid`), and job class (`jclass`). The front-end formatters then can send the consolidated data output to either an ASCII report or to a binary file.

Note: Disk usage information is not available on a job basis in the UNICOS operating system; thus, it cannot be consolidated by job ID or job class. However, output from the `dodisk`(8) utility can be used for billing disk usage on a user ID or account ID basis.

2.3.1 Why Use Front-end Formatting

Sites may want to use a front-end formatter to customize Cray accounting data in the following situations:

- All billing is done on a single system. When accounting data from several systems are processed on a single system, the units of measure may need to be standardized. For example, all CPU time should be expressed in milliseconds.
- The front-end system is an IBM machine that requires character fields to be in EBCDIC format.
- Only a few fields are important to the billing system; these usually include CPU time, memory use, disk use, and swap use.

Cray accounting products let you choose from two types of front-end formatting:

- Cray system accounting (CSA) front-end formatters are templates of C programs that show you how to consolidate session file records and delivers output in VM, MVS, or ASCII format.
- The generic front-end formatter, `csagfef`(8), accepts as input a generic consolidated data file or multiple `pacct` (per-process accounting data) files. It delivers output as either an ASCII report or a Cray binary file. `csagfef` cannot convert output to VM or MVS format.

You should consider several factors when deciding which front-end formatter to use:

- The CSA front-end formatters require a source license, while the generic formatter does not.
- The generic front-end formatter delivers either ACSII or Cray binary data output, where binary numbers are always written as a 64-bit word. CSA formatters can be modified to write 32-bit numeric values or EBCDIC output.
- Both types of formatters process session record files, which are created by `csabuild`(8). However, the generic formatter is also capable of processing multiple `pacct` files.

2.3.2 Preparing to Use a Formatter

Before you attempt either to modify a CSA formatter or to execute the generic formatter, you must make several decisions based on what you want the final report or data file to contain. The issues you must decide upon include the following:

- Identifying the data that needs to be reported.

  A multitude of data can be extracted from a session or a `pacct` file. For efficiency and the conservation of disk space, only the necessary data should be consolidated by the CSA formatters or by `csagcon`(8).

- Selecting the consolidation keys.

  You can use various keys to consolidate the data. Both types of formatters support data consolidation by account ID, group ID, job ID, and user ID or some combination thereof. `csagcon` also supports data consolidation by job class, which is either interactive or batch through the Network Queuing System (NQS).

- Determining which sessions should be consolidated when the input is a session file.

You can consolidate data for only terminated sessions, only active sessions, or both terminated and active sessions.

- Selecting the format of the ASCII report or binary data file.

  Among the things to be decided are the units of the various fields, the precision, the order of the data, the character set, the length of character strings, and the size and format of binary integer and floating point numbers.

After making these decisions, you should modify or set up the front-end formatter to generate reports or data files based on these specifications. Normally, front-end formatters are executed by `csarun` in either the `FEF` or the `USEREXIT` state. See Section 2.1.10.3, page 51, for more information on these user exits.

2.3.3 CSA Front-end Formatting

All CSA front-end formatters contain code both to consolidate session record data and to send consolidated data to a report or file. You must modify one of these templates in order to consolidate and send the data output specifically needed by your site.

> **Note:** `csafef`(8), `csafef2`(8), and `csaibm`(8) are templates; if you execute them as released, they produce a message stating that they are templates. If your site wants to use one of these programs, you must have a source license and you must make modifications to the code. Any local changes made to these templates are not supported by Cray.

2.3.4 Generic Front-end Formatting

The generic accounting data consolidator `csagcon`(8) and the generic front-end formatter `csagfef`(8) are more flexible versions of the `csacon`(8) and `csacrep`(8) utilities. They let you do the following tasks:

- Consolidate a session file
- Consolidate one or more `pacct` accounting files
- Generate an ASCII report or a binary file based on a file created by `csagcon`

The `csagcon` and `csagfef` utilities let you specify the fields to be consolidated and the format of the report. In contrast, `csacon` and `csacrep` have hardcoded data specifications and formats that cannot be changed without source code and local modifications.

Administrators who execute `csagcon` may need privilege to access the the `/dev/kmem` file. If this privilege is needed and you do not possess it, `csagcon` will terminate with an error.

The `csagcon` and `csagfef` utilities can be executed from the `csarun` user exit scripts. Both commands can be invoked from either the `FEF` or `USEREXIT` state of `csarun`. See Section 2.1.10.3, page 51, for more information on user exits.

To invoke `csagcon` and `csagfef` from the `FEF` state, put these or similar commands in the file `/usr/lib/acct/csa.fef`:

```
csagcon -S ${SESSION_FILE} -s username -o ${SESSION_DIR}/gacct
csagfef -f ${SESSION_DIR}/gacct source_file > ${CRPT_DIR}/site.rpt
```

Alternately, the same two commands can be placed into the `/usr/lib/acct/csa.user` file; then, `csagcon` and `csagfef` will execute from the `csarun` `USEREXIT` state.

2.3.4.1 Data Consolidation

The `csagcon` command consolidates data either from a session file, which is created by `csabuild`(8), or from `pacct` files. You can choose the data that is to be consolidated by using the `csagcon -R` option. If a data list is not specified, a set of default variables is selected. In addition, some variables are always selected.

The variable names listed throughout this section are used by both `csagcon` and `csagfef`.

2.3.4.2 Required Data Variables

The following table lists the required variables that are always included in the consolidated data file. You must not include any of these variables in a `csagcon` request file (`-R` option). If you do, `csagcon` will terminate with an error.

Table 4. Required Data Variables

Variable	Type or Value	Description	
`acid` *	Integer	Account ID.	
`con_key`	Integer	`csagcon` consolidation option(s) you specify. If you specify multiple options, the values are added together.	
		Value	`csagcon` consolidation option
		0001	`-a` (consolidate by the account ID (`acid`) variable)
		0002	`-c` (consolidate by the job class (`jclass`) variable; job class is either interactive or NQS)
		0004	`-g` (consolidate by the group ID (`gid`) variable)
		0010	`-j` (consolidate by the job ID (`jid`) variable)
		0020	`-u` (consolidate by the user ID (`uid`) variable)
		0040	`-N` (consolidate NQS requests strictly by job ID)
		0100	`-A` (consolidate active and terminated sessions)
		0200	`-C` (consolidate only active sessions)
`creatime`	Integer	Creation time of the file in seconds since 00:00:00 GMT, 1 January 1970.	
`file_end`	Integer	If the input was a `pacct` file, this is the latest process end time found in the file. If the input was a session file, this is the end time of the last uptime period. Measured in seconds.	
`file_start`	Integer	If the input was a `pacct` file, this is the earliest process end time found in the file. If the input was a session file, this is the start time of the first uptime period. Measured in seconds.	
`gid` *	Integer	Group ID.	
`ios`	Integer	I/O subsystem type.	
		Value	I/O subsystem type
		1	Model E
`jclass` *	Integer	Job class.	
		Value	Job class
		1	Interactive job

Variable	Type or Value	Description	
		2	NQS job
jid *	Integer	Job ID.	
ncpus	Integer	Number of CPUs started.	
njobs	Integer	Number of jobs. Calculated as the number of pacct end-of-job records found.	
nproc	Integer	Number of processes.	
nsess	Integer	Number of sessions. This is meaningful only when the input was a session file.	
num_datarec	Integer	Number of data records in the file.	
sort_opt	Integer	csagcon sort option used.	
		Value	csagcon sort option
		0	None (unsorted)
		1	-s acid (sorted by account ID then user ID)
		2	-s acname (sorted by account name then user name)
		3	-s jclass (sorted by job class then job ID)
		4	-s uid (sorted by user ID then account ID)
		5	-s uname (sorted by user name then account name)
tp_devgrp	String	An array that is indexed by 0 through (tp_ndevgrp -1) and contains the names of the tape device groups. The names are prefixed with tp_. If there are fewer than tp_ndevgrp tape device groups, the unused entries have values of tp_null0, tp_null1, and so on. This field is reported when the input was a session file and tape information was requested.	
tp_ndevgrp	Integer	Number of tape device groups. This field is reported when the input was a session file and tape information was requested.	
uid *	Integer	User ID.	
us_nttype	Integer	Number of UNICOS station call processor (USCP) transfer types. This field is reported when the input was a session file and USCP information was requested.	

Variable	Type or Value	Description
us_tname	String	An array that is indexed by 0 through (us_nttype -1) and contains the names of the USCP transfer types. The names are prefixed with us_. This field is reported when the input was a session file and USCP information was requested.
BYTE_CLICK	Integer	Number of bytes per click.
BYTE_WORD	Integer	Number of bytes per word.
CLK_TCK	Integer	Number of clocks per second.
FPTYPE	String	Floating point type: Cray or IEEE.
HARDWARE	String	Machine identification. Includes serial number and mainframe type.
MACHINE	String	Machine name.
MAXBDEVNO	Integer	Maximum number of block devices.
MAXCDEVNO	Integer	Maximum number of character devices.
MAXCPUS	Integer	Maximum number of CPUs for this mainframe type and subtype.
MEMORY	String	Memory configuration.
MEMORY_NWORD	Integer	Total system and user memory in words.
NODENAME	String	Network node name.
OS_HZ	Integer	Clock rate (the frequency per second with which the clock routine is called); usually 60 or 100.
RELEASE	String	Release of the operating system.
SDS_WGHT	Integer	Number of clicks per SDS allocation unit.
SOFTWARE	String	Software release information.
SYSNAME	String	Operating system name.
VERSION	String	Release version of the operating system.
WORD_CLICK	Integer	Number of words per click.

* If this variable is not selected as a consolidation key, its value is -2. For example, the following command consolidates session record file by job ID:

```
csagcon -jN -S Session-Record.0928 -o gacct.0928
```

In the file `gacct.0928` the values for the `acid`, `gid`, `jclass`, and `uid` variables will be -2 for records. This is because these variables were not selected as consolidation keys on the command line.

2.3.4.3 Default and Optional Data Variables

The following sections describe the data that you can specify in a `csagcon` request file (`-R` option). The request file contains a list of variables that will be consolidated by `csagcon`. By default, `csagcon` consolidates the same data as `csacon`(8).

The `csagcon` utility gets the default and optional data variables from the file `/usr/lib/acct/table_init`. Specifying a different file using the `-T` option is not recommended because `csagcon` expects the data variable names given in this file. Use caution in specifying the `-T` option; normally it is used only for debugging source code.

The column headings are defined as follows:

Heading	Meaning
Variable	The name that `csagcon` and `csagfef` use for the data item. This name, except where noted, should appear in the request file when you use the `csagcon -R` option.
Type	The data type of the variable. Valid types are `integer`, `float`, and `string`.
Unit	The unit, if any, of the data item. The item can be converted to another unit by `csagfef` (see Section 2.3.4.5.7, page 127).
Default	Specifies whether a data item is consolidated when the `csagcon -R` option is omitted. Yes — The data item is consolidated by default. No — The data item is not consolidated by default.
Job	Specifies whether the `csagcon -j` option must be used when the data item is consolidated. Yes — The `csagcon -j` option must be used. For this data item to be consolidated when `-j` is specified, either the `-R` option is not used and this is a default item, or the `-R` option is used and this item is listed in the request file.

No	The `csagcon -j` option does not have to be used.

2.3.4.3.1 `pacct` Record Variables

This section describes the variables that contain `pacct` process information. These variables are available when the `csagcon` input is either a session file or one or more `pacct` files.

Note: The values in Table 5 are only available when using the `csagcon -I` option.

Table 5. `pacct` Base Record Variables — Per-process Values

Variable	Type	Unit	Default	Job	Description
`pp_p_cmd`	String	-	No	No	Command name (first 8 characters).
`pp_p_flag`	Integer	-	No	No	Record flags (See `ac_flag` in `/user/include/sys/acct.h`).
`pp_p_nice`	Integer	-	No	No	Nice value.
`pp_p_pid`	Integer	-	No	No	Process ID.
`pp_p_ppid`	Integer	-	No	No	Parent process ID.
`pp_p_stat`	Integer	-	No	No	Exit status.
`ps_p_tty`	String	-	No	No	Controlling tty device (maximum of 8 characters).

Table 6. `pacct` Base Record Variables - Total Values

Variable	Type	Unit	Default	Job	Description
`pb_t_btime`	Integer	Seconds	No	Yes	Process start time.
`pb_t_ctime`	Integer	Clocks	No	No	Process connect time.
`pb_t_etime`	Integer	Clocks	No	No	Elapsed time.

Variable	Type	Unit	Default	Job	Description
`pb_t_io`	Integer	Bytes	No	No	Number of characters transferred.
`pb_t_iobtime`	Integer	Clocks	No	No	I/O wait time.
`pb_t_iosw`	Integer		No	No	I/O swap count.
`pb_t_iowmem`	Integer	Click-ticks	No	No	I/O wait time memory integral while locked in memory.
`pb_t_iowtime`	Integer	Clocks	No	No	I/O wait while locked in memory.
`pb_t_kcore`	Float	Kiloword-minute	No	No	Kcore-minutes.
`pb_t_lio`	Integer		No	No	Number of logical I/O requests.
`pb_t_mem`	Integer	Click-ticks	No	No	Memory integral.
`pb_t_phimem_max`	Integer	Words	No	No	Maximum process highwater memory mark.
`pb_t_phimem_min`	Integer	Words	No	No	Minimum process highwater memory mark.
`pb_t_rw`	Integer		No	No	Number of physical I/O requests.
`pb_t_sctime`	Integer	Clocks	No	No	System call time.
`pb_t_stime`	Integer	Clocks	No	No	System CPU time.
`pb_t_utime`	Integer	Clocks	No	No	User CPU time.

Table 7. `pacct` Base Record Variables - Prime Time Values

Variable	Type	Unit	Default	Job	Description
`pb_p_ctime`	Float	Clocks	No	No	Process connect time.
`pb_p_etime`	Float	Clocks	No	No	Elapsed time.
`pb_p_io`	Float	Bytes	Yes	No	Number of characters transferred.
`pb_p_iobtime`	Float	Clocks	Yes	No	I/O wait time.
`pb_p_iosw`	Float		No	No	I/O swap count.
`pb_p_iowmem`	Float	Click-ticks	Yes	No	I/O wait time memory integral while locked in memory.
`pb_p_iowtime`	Float	Clocks	Yes	No	I/O wait while locked in memory.
`pb_p_kcore`	Float	Kiloword-minute	Yes	No	Kcore-minutes.
`pb_p_lio`	Float		Yes	No	Number of logical I/O requests.
`pb_p_mem`	Float	Click-ticks	No	No	Memory integral.
`pb_p_rw`	Float		Yes	No	Number of physical I/O requests.
`pb_p_sctime`	Float	Clocks	Yes	No	System call time.
`pb_p_stime`	Float	Clocks	Yes	No	System CPU time.
`pb_p_utime`	Float	Clocks	Yes	No	User CPU time.

Table 8. `pacct` Base Record Variables - Nonprime Time Values

Variable	Type	Unit	Default	Job	Description
`pb_n_ctime`	Foat	Clocks	No	No	Process connect time.
`pb_n_etime`	Float	Clocks	No	No	Elapsed time.
`pb_n_io`	Float	Bytes	Yes	No	Number of characters transferred.

Variable	Type	Unit	Default	Job	Description
`pb_n_iobtime`	Float	Clocks	Yes	No	I/O wait time.
`pb_n_iosw`	Float		No	No	I/O swap count.
`pb_n_iowmem`	Float	Click-ticks	Yes	No	I/O wait time memory integral while locked in memory.
`pb_n_iowtime`	Float	Clocks	Yes	No	I/O wait while locked in memory.
`pb_n_kcore`	Float	Kiloword-minute	Yes	No	Kcore-minutes.
`pb_n_lio`	Float		Yes	No	Number of logical I/O requests.
`pb_n_mem`	Float	Click-ticks	No	No	Memory integral.
`pb_n_rw`	Float		Yes	No	Number of physical I/O requests.
`pb_n_sctime`	Float	Clocks	Yes	No	System call time.
`pb_n_stime`	Float	Clocks	Yes	No	System CPU time.
`pb_n_utime`	Float	Clocks	Yes	No	User CPU time.

Table 9. `pacct` Secondary Data Storage (SDS) Record Variables - Total Values

Variable	Type	Unit	Default	Job	Description
`ps_t_memsw`	Integer	Click-ticks	No	No	SDS execution memory integral.
`ps_t_sdioch`	Integer	Bytes	No	No	Number of bytes transferred to or from SDS.
`ps_t_sdlio`	Integer		No	No	Number of logical SDS I/O requests.
`ps_t_sdsmem`	Integer	Click-ticks	No	No	SDS residency memory integral.

Table 10. `pacct` SDS Record Variables - Prime Time Values

Variable	Type	Unit	Default	Job	Description
`ps_p_memsw`	Float	Click-ticks	No	No	SDS execution memory integral.
`ps_p_sdioch`	Float	Bytes	Yes	No	Number of bytes transferred to or from SDS.
`ps_p_sdlio`	Float		Yes	No	Number of logical SDS I/O requests.
`ps_p_sdsmem`	Float	Click-ticks	No	No	SDS residency memory integral.

Table 11. `pacct` SDS Record Variables - Nonprime Time Values

Variable	Type	Unit	Default	Job	Description
`ps_n_memsw`	Float	Click-ticks	No	No	SDS execution memory integral.
`ps_n_sdioch`	Float	Bytes	Yes	No	Number of bytes transferred to or from SDS.
`ps_n_sdlio`	Float		Yes	No	Number of logical SDS I/O requests.
`ps_n_sdsmem`	Float	Click-ticks	No	No	SDS residency memory integral.

Note: All of the variables in Table 12 are available when `-E` is specified. Job-specific variables (the `Job` value is `Yes`) are also accessible when the `csagcon -j` option is used.

Table 12. `pacct` End-of-job Record Variables

Variable	Type	Unit	Default	Job	Description
`pe_t_fsblkused`	Integer		No	Yes	Number of file system blocks used.
`pe_t_jetime`	Integer	Seconds	No	Yes	Time the job ended.
`pe_t_jhimem`	Integer	Clicks	No	Yes	Job highwater memory mark.
`pe_t_jnice`	Integer	-	No	No	Nice value at job termination.
`pe_t_sdshiwat`	Integer	SDS allocation units	No	Yes	Job SDS highwater mark.

In Table 13, the `pd_t_b`*xxxx* variables are arrays that are indexed by 0 through (`MAXBDEVNO` - 1). The `pd_t_c`*xxxx* variables are arrays that are indexed by 0 through (`MAXCDEVNO` - 1).

Table 13. `pacct` Device I/O Record Variables - Total Values

Variable	Type	Unit	Default	Job	Description
`pd_t_bioch`	Integer	Bytes	No	No	Number of bytes transferred to or from the block device.
`pd_t_blio`	Integer		No	No	Number of logical I/O requests for the block device.
`pd_t_btype`	Integer		No	No	Major device number for block devices. A device number of -1 indicates that there is no accounting information for the array index.
`pd_t_cioch`	Integer	Bytes	No	No	Number of bytes transferred to or from the character device.

Variable	Type	Unit	Default	Job	Description
`pd_t_clio`	Integer		No	No	Number of logical I/O requests for the character device.
`pd_t_ctype`	Integer		No	No	Major device number for character devices. A device number of -1 indicates that there is no accounting information for this array index.

In Table 14, the `pd_p_b`*xxxx* variables are arrays that are indexed by 0 through (`MAXBDEVNO` - 1). The `pd_p_c`*xxxx* variables are arrays that are indexed by 0 through (`MAXCDEVNO` - 1).

Table 14. `pacct` Device I/O Record Variables - Prime Time Values

Variable	Type	Unit	Default	Job	Description
`pd_p_bioch`	Float	Bytes	Yes	No	Number of bytes transferred to or from the block device.
`pd_p_blio`	Float		Yes	No	Number of logical I/O requests for the block device.
`pd_p_btype`	Integer		Yes	no	Major device number for block devices. A device number of -1 indicates that there is no accounting information for the array index.
`pd_p_cioch`	Float	Bytes	Yes	No	Number of bytes transferred to or from the character device.

Variable	Type	Unit	Default	Job	Description
`pd_p_clio`	Float		Yes	No	Number of logical I/O requests for the character device.
`pd_p_ctype`	Integer		Yes	No	Major device number for character devices. A device number of -1 indicates that there is no accounting information for this array index.

In Table 15, the `pd_n_b`*xxxx* variables are arrays that are indexed by 0 through (`MAXBDEVNO` - 1). The `pd_n_c`*xxxx* variables are arrays that are indexed by 0 through (`MAXCDEVNO` - 1).

Table 15. `pacct` Device I/O Record Variables - Non-prime Time Values

Variable	Type	Unit	Default	Job	Description
`pd_n_bioch`	Float	Bytes	Yes	No	Number of bytes transferred to or from the block device.
`pd_n_blio`	Float		Yes	No	Number of logical I/O requests for the block device.
`pd_n_cioch`	Float	Bytes	Yes	No	Number of bytes transferred to or from the character device.
`pd_n_clio`	Float		Yes	No	Number of logical I/O requests for the character device.

Table 16. pacct Massively Parallel Processing (MPP) Record Variables - Total Values

Variable	Type	Unit	Default	Job	Description
`pm_t_pe`	Integer		No	No	Number of MPP processing elements.
`pm_t_pe_max`	Integer		No	No	Largest number of MPP processing elements used by a single process.
`pm_t_pe_time`	Integer	Clocks	No	No	Sum of (number of PEs used multiplied by time used).
`pm_t_time`	Integer	Clocks	No	No	MPP time used.
`pm_t_time_max`	Integer	Clocks	No	No	Greatest amount of MPP time used by a single process.

Table 17. pacct MPP Record Variables - Prime Time Values

Variable	Type	Unit	Default	Job	Description
`pm_p_pe`	Float		Yes	No	Number of MPP processing elements.
`pm_p_pe_time`	Float	Clocks	Yes	No	Sum of (number of PEs used multiplied by time used).
`pm_p_time`	Float	Clocks	Yes	No	MPP time used.

Table 18. `pacct` MPP Record Variables - Nonprime Time Values

Variable	Type	Unit	Default	Job	Description
`pm_n_pe`	Float		Yes	No	Number of MPP processing elements.
`pm_n_pe_time`	Float	Clocks	Yes	No	Sum of (number of PEs multiplied by time used).
`pm_n_time`	Float	Clocks	Yes	No	MPP time used.

Note: Each item in the following multitasking tables is an array that is indexed by 0 through (`MAXCPUS` - 1).

Table 19. `pacct` Multitasking Record Variables - Total Values

Variable	Type	Unit	Default	Job	Description
`pu_t_mutime`	Integer	Clocks	No	No	Time connected to [i+1] CPUs.
`pu_t_smwtime`	Integer	Clocks	No	No	Semaphore wait time.

Table 20. `pacct` Multitasking Record Variables - Prime Time Values

Variable	Type	Unit	Default	Job	Description
`pu_p_mutime`	Float	Clocks	Yes	No	Time connected to [i+1] CPUs.
`pu_p_smwtime`	Float	Clocks	No	No	Semaphore wait time.

Table 21. `pacct` Multitasking Record Variables - Nonprime Time Values

Variable	Type	Unit	Default	Job	Description
`pu_n_mutime`	Float	Clocks	Yes	No	Time connected to [i+1] CPUs.
`pu_n_smwtime`	Float	Clocks	No	No	Semaphore wait time.

Table 22. pacct Performance Record Variables - Total Values

Variable	Type	Unit	Default	Job	Description
pp_t_phrwblks	Integer		No	No	Number of raw physical blocks moved.
pp_t_rwblks	Integer		No	No	Number of buffered physical blocks moved.
pp_t_rtime	Integer	Clocks	No	No	Process start time past pb_t_btime.
pp_t_srunwtime	Integer	Seconds	No	No	SRUN wait time.
pp_t_swapclocks	Integer	Clocks	No	No	Swapped time.
pp_t_tiowtime	Integer	Clocks	No	No	Terminal I/O wait time.

Table 23. pacct Performance Record Variables - Prime Time Values

Variable	Type	Unit	Default	Job	Description
pp_p_phrwblks	Float		No	No	Number of raw physical blocks moved.
pp_p_rwblks	Float		No	No	Number of buffered physical blocks moved.
pp_p_rtime	Float	Clocks	No	No	Process start time past pb_t_btime.
pp_p_srunwtime	Float	Seconds	No	No	SRUN wait time.
pp_p_swapclocks	Float	Clocks	No	No	Swapped time.
pp_p_tiowtime	Float	Clocks	No	No	Terminal I/O wait time.

Table 24. `pacct` Performance Record Variables - Nonprime Time Values

Variable	Type	Unit	Default	Job	Description
`pp_n_phrwblks`	Float		No	No	Number of raw physical blocks moved.
`pp_n_rwblks`	Float		No	No	Number of buffered physical blocks moved.
`pp_n_rtime`	Float	Clocks	No	No	Process start time past `pb_t_btime`.
`pp_n_srunwtime`	Float	Seconds	No	No	SRUN wait time.
`pp_n_swapclocks`	Float	Clocks	No	No	Swapped time.
`pp_n_tiowtime`	Float	Clocks	No	No	Terminal I/O wait time.

2.3.4.3.2 Daemon Accounting Variables

The accounting variables that contain daemon usage information are available only when the `csagcon` input file is a session file.

In Table 25 each item is an array that is indexed by the tape device group names prefixed by `tp_` (see the `tp_devgrp` array in Table 14, page 112) or by 0 through (`tp_ndevgrp - `1).

Table 25. Tape Accounting Variables

Variable	Type	Unit	Default	Job	Description
`tp_nmount`	Integer		Yes	No	Number of volumes mounted.
`tp_nread`	Integer	Bytes	Yes	No	Number of bytes read.
`tp_nwrite`	Integer	Bytes	Yes	No	Number of bytes written.
`tp_rtime`	Integer	Seconds	Yes	No	Reservation time.

Variable	Type	Unit	Default	Job	Description
tp_stime	Integer	Clocks	Yes	No	System CPU time.
tp_utime	Integer	Clocks	Yes	No	User CPU time.

In Table 26, the values for nq_init, nq_disp, and nq_term are found in /usr/include/acct/dacct.h.

Table 26. NQS Accounting Variables

Variable	Type	Unit	Default	Job	Description
nq_btime *	Integer	Seconds	No	Yes	Start time of the request.
nq_disp *	Integer		No	Yes	Dispose subtype (NQ_DISP).
nq_elapse **	Integer	Seconds	No	Yes	Wall-clock time used while the request was running.
nq_init *	Integer		No	Yes	Initiation subtype (NQ_INIT).
nq_machname	String		No	Yes	Originating machine name (16 characters).
nq_mid *	Integer		No	Yes	Originating machine ID.
nq_nreq	Integer		Yes	No	Number of NQS requests.
nq_quename	String		No	Yes	Name of the last queue in which the request was located (16 characters).
nq_qwtime	Integer	Seconds	No	No	Queue wait time.
nq_reqname	String		No	Yes	Request name (16 characters).
nq_seqno *	Integer		No	Yes	Sequence number.

Variable	Type	Unit	Default	Job	Description
`nq_stime`	Integer	Clocks	Yes	No	Shepherd system CPU time.
`nq_term` *	Integer		No	Yes	Termination subtype (`NQ_TERM`).
`nq_utime`	Integer	Clocks	Yes	No	Shepherd user CPU time.
`nq_wallclock` ***	Integer	Seconds	No	Yes	Total wall-clock time for the request to complete.

* If the value for this field is unknown, or if this is an interactive session, this field is set automatically to -9.

** `nq_elapse` is the amount of wall-clock time which elapsed while the request was running on a CPU. This does not include queue wait time, system down time, or the period when the request was suspended, checkpointed, or held.

*** `nq_wallclock` is the total amount of wall-clock time it took the request to complete. This includes queue wait time and system down time. This value is reported only once for a request. It is possible that the amount of CPU time the request uses is greater than the wall-clock time, because the request could have created additional processes, been multitasked, or done work in the background.

Table 27. Connect Time Accounting Variables

Variable	Type	Unit	Default	Job	Description
`ct_con_n`	Integer	Seconds	Yes	No	Nonprime time connect time.
`ct_con_p`	Integer	Seconds	Yes	No	Prime time connect time.
`ct_nlogin`	Integer		Yes	No	Number of interactive logins.

2.3.4.3.3 System Billing Units (SBU) Variables

The following table describes the variables that contain information about the system billing units (SBUs). If the input to `csagcon` is a session file, all the

SBUs are multiplied by the appropriate NQS weighting factor. The NQS weighting factors are defined in the accounting configuration file `/etc/config/acct_config`.

Table 28. System Billing Units (SBU) Variables

Variable	Type	Unit	Default	Job	Description
`sb_pacct`	Float	Billing units	No	No	`pacct` SBUs.
`sb_tape`	Float	Billing units	No	No	Tape SBUs.
`sb_uscp`	Float	Billing units	No	No	USCP SBUs.
`sb_ctime`	Float	Billing units	No	No	Connect time SBUs.
`sb_total`	Float	Billing units	Yes	No	Total SBU value.

2.3.4.4 Data File Format

The `csagcon` consolidated data file consists of header and data records. The header records describe both the machine on which the data was collected and the data records.

The `csagfef -h` option displays some of the information found in the header records.

The file is organized as follows:

Record Type	Description
Header word	File identifier that is defined in `/usr/include/sys/accthdr.h`.
`gc-defs`	Definitions record.
`gc-imeta`	Meta record for integer data.
`gc-fmeta`	Meta record for floating point data.
`gc-cmeta`	Meta record for character string data.

`gc-data`	Indicator for the start of data record 1.
`gc-int`	Data record 1 containing integer data.
`gc-float`	Data record 1 containing floating point data.
`gc-char`	Data record 1 containing character string data.
`gc-data`	Indicator for the start of data record 2.
`gc-int`	Data record 2 containing integer data.
`gc-float`	Data record 2 containing floating point data.
`gc-char`	Data record 2 containing character string data.

(Additional `gc-data`, `gc-int`, `gc-float`, and `gc-char` records for each data record.)

2.3.4.4.1 Header Records

Header records appear only once, at the beginning of the consolidated data file. There are three types of header records:

Header Record Type	Description
Header word	Identifies the file according to the format specified in the file `/usr/include/sys/accthdr.h`. This word allows other accounting programs to check for a valid input file type before attempting to process the file.
Definitions record	Contains constants and character strings that describe the machine on which the data was consolidated and array element names. These variables can be accessed by `csagfef`(8) and are listed in Section 2.3.4.2, page 101.

Meta record	Describes the data in the data records. A meta record lists the name, type, and size of each item or array in the data records and the order of the data found in the data records. There is a separate meta record for integer data, floating point data, and character string data.

2.3.4.4.2 Data Records

Data records follow the header records in a file. The `gc-data` record denotes the start of the data for a unique consolidation identifier.

2.3.4.5 `csagfef`(8) Source Scripts

The `csagfef`(8) command is a translator that formats `csagcon`(8) output into an ASCII report or a binary file according to the directives found in a source script.

The `csagfef` scripts are based on four sections including the body, any of which may be empty or missing. Scripts can contain any of the following sections in any order:

`BEGIN` *{ statements }*
`END` *{ statements }*
`function` *name (arglist) { statements }*
statements

The `csagfef` command can process multiple source scripts, and one script can contain multiple `BEGIN`, `END`, or body sections. In these cases, `csagfef` executes the statements for all like sections in the order that they appear in the scripts or script.

For example, all statements in the various `BEGIN` sections will be combined into one `BEGIN` section. The statements will be in the same order as they appear in the scripts or script.

2.3.4.5.1 BEGIN Section

The statements associated with BEGIN comprise the preamble. The preamble is executed once after the definition and meta-data records are read. The preamble can be used to print report headings and to initialize variables used in the body

2.3.4.5.2 END Section

The statements associated with END comprise the postamble. The postamble is executed once after all the records in the data file have been read. You can instruct csagfef in this section to process and print summary data.

2.3.4.5.3 function Section

The statements in the function section of csagfef define functions as specified by you. Functions always begin with the word function followed by the function name and the argument list. The arglist consists of names separated by commas. These argument names are the formal parameters of the function and the variables that are local to the function. Function calls may be nested and recursive. The return statement can be used to return a value.

2.3.4.5.4 Body

Statements that are not in any of the above sections form the body of the csagfef source script. Typically, these statements print out information from the data records. This section is executed once for each data record encountered.

2.3.4.5.5 Example Source Scripts

Examples of csagfef source scripts can be found in the /usr/src/cmd/acct/src/csa/csagfef/examples directory.

2.3.4.5.6 csagfef Language Description

The csagfef language is the action language of nawk without the string processing operations. If you are familiar with nawk, you will have little difficulty writing and understanding csagfef scripts. The pattern part of nawk is unnecessary in csagfef, because the data format is defined in the data file. You merely select the data items to process by name.

csagfef implements a version of the awk language (new awk, or nawk) described in *The AWK Programming Language*, by Alfred Aho, Brian Kernighan, and Peter Weinberger (1988).

A `csagfef` script can include any of the following statements:

`if (` *expression* `)` *statement* `[ else` *statement* `]`
`while (` *expression* `)` *statement*
`do` *statement* `while (` *expression* `)`
`for (` *expression*`;` *expression*`;` *expression* `)` *statement*
`break`
`continue`
`{ [` *statements* `] }`
expression
`print` *expression-list* `[ >`*expression* `]`
`printf` *format*`[,` *expression-list* `] [ >`*expression* `]`
`next`
`exit [` *expression* `]`
`return [` *expression* `]`

The following describes further the contents of statements in a `csagfef` script:

- Statement terminators. Statements are terminated by semicolons, right braces, or newlines.
- Statement continuation. Statements can be continued on successive lines by using \ as the last character of the line. Statements can also be continued after the following symbols:

```
,       (comma)
{       (left brace)
&&      (logical AND)
||      (logical OR)

do
else
)       (right parenthesis in an "if" or "for" statement)
```

- Comments. Nonexecutable comments begin with # and end with a newline. They can appear anywhere in the source script.
- Expressions. Expressions include constants, variables, and operators. Parentheses can be used to control the grouping of the operations in an expression.
- Logical expressions. Logical expressions have a value of 1 (true) and 0 (false). As in the C language, any nonzero value is taken to be true.
- Numbers. Numbers can be integers or floating points. The format is the same as that recognized by `strtod`(3C) and `strtol`(3C): digits, decimal

point, digits, `e` or `E`, signed exponent. At least one digit or a decimal point must be present; the other components are optional. Octal integers begin with 0. Hexadecimal integers begin with 0 *x*.

- Variable names. Variable names consist of a letter followed by a string of letters, numbers, or the character _. Variables are used to name the data items found in the data records of the consolidated file.

  Some variables in the consolidated data file are arrays. The elements of these arrays can be referenced by indexing. For example, the variable, `pu_t_mutime`, is an array that contains the time a process was connected to (i+1) CPUs; see Table 19, page 115 (table: `pacct` multitasking record variables). The time a process was connected to one CPU is referenced by `pu_t_mutime` [0].

  You can also define additional variables within the `csagfef` source script; however, user-defined arrays are not supported.

A `csagfef` script can include prefix, infix, and suffix operators as follows:

Prefix operators

The `csagfef` command applies a prefix operator immediately preceding a term and any suffix operators. It then applies any prefix operators to the left of that operator, grouping them from right to left.

Operator	Action
++X	Preincrement
–X	Predecrement
+X	Plus
-X	Minus
!X	Logical NOT

Infix operators

The `csagfef` command applies infix operators, in descending order of precedence, as follows:

Operator	Action
X^Y	Exponentiation
X*Y	Multiplication
X/Y	Division
X%Y	Remainder

X+Y	Addition
X-Y	Subtraction
X<Y	Less than
X<=Y	Less than or equal
X>Y	Greater than
X>=Y	Greater than or equal
X==Y	Equals
X!=Y	Not equals
X&&Y	Logical AND
X||Y	Logical OR
Z?X:Y	Conditional
X=Y	Assignment
X*=Y	Multiply assign
X/=Y	Divide assign
X%=Y	Remainder assign
X+=Y	Add assign
X-=Y	Subtract assign
X,Y	Comma

Suffix operators

The `csagfef` command applies a suffix operator immediately following a term before it applies any other operator. It then applies any suffix operators to the right of that operator, grouping them from left to right. The following list shows the suffix operators:

Operator	Action
X++	Postincrement
X–	Postdecrement
X[Y]	Subscript

X(Y)	Function call

2.3.4.5.7 Built-in Functions

The `csagfef` command has the following built-in functions, with the function parameters (given in parentheses) defined at the end of the list:

Function name	Description
`abs(`*exp*`)`	Returns the absolute value of *exp*.
`acid2nam`(*num*)	Returns the character string associated with the account ID (*num*). If there is no associated string, return `Unknown`.
`bytes_to`(*num*[, *unit*])	Converts bytes to some other unit. If [, *unit*] is not specified, kilobytes are returned.
`clicks_to`(*num*[, *unit*])	Converts clicks to some other unit. If [, *unit*] is not specified, kilobytes are returned.
`clocks_to`(*num*[, *tunit*])	Converts clocks to some other unit. If [, *tunit*] is not specified, seconds are returned.
`close(`*str*`)`	Closes the file stream specified by *str*.
`frac(`*exp*`)`	Returns the fractional part of *exp*.
`gid2nam`(*num*)	Returns the character string associated with the group ID (*num*). If there is no associated string, return `Unknown`.
`imax(`*arr*`)`	Returns the index of the maximum element of array *arr*.

`imin(`*arr*)	Returns the index of the minimum element of array *arr*.
`int(`*exp*)	Returns the integer part of *exp*.
`isdefined(`*sym*)	Returns 1 if *sym* is defined. Otherwise, returns 0.
`nam2acid`(*str*)	Returns the numeric account ID associated with the account name (*str*). If there is no account ID, return -1.
`nam2gid`(*str*)	Returns the numeric group ID associated with the group name (*str*). If there is no group ID, returns -1.
`nam2uid`(*str*)	Returns the numeric ID associated with the user name (*str*). If there is no user ID, returns -1.
`strcmp`(*str1*, *str2*)	Compares two strings. Returns a value that is greater than, equal to, or less than 0 according to whether *str1* is greater than, equal to, or less than *str2*.
`strftime`(*fmt*, [`calendar_time`])	Formats the time into a string according to (*fmt*). Time is an integer; for example, the required data variable `file_start`.
`strlen`(*str*)	Returns the number of characters in string (*str*).
`sum(`*arr*)	Returns the sum of the elements in array *arr*.
`system(`*str*)	Passes *str* to the shell for execution.
`ticks_to`(*num*[, *tunit*])	Converts ticks to some other unit. If [, *tunit*] is not

	specified, seconds are returned.
`uid2nam`(*num*)	Returns the character string associated with the user ID (*num*). If there is no associated string, returns `Unknown`.
`words_to`(*num*[, *unit*])	Converts words to some other unit. If [, *tunit*] is not specified, kilowords are returned.

The definitions of the function parameters are as follows:

Parameter	Definition
arr	An array name. For example: `imax` (`pd_t_cioch`)
exp	A variable name or a function invocation. For example: `abs` (`pb_t_rw`) Variable name
fmt	NULL or a valid `strftime`(3C) format that is enclosed in double quotes. For example: `strftime` () NULL format `strftime` (" `%X` ") `strftime` format
num	Either an integer value or the name of a variable that contains an integer value. For example: `bytes_to` (`pb_t_io`) Variable name `uid2nam` (*uid*) Variable name

	`words_to` (5125)
	Integer value
str, *str1*, *str2*	Either character strings enclosed in double quotation marks or the names of a variables whose values are character strings. For example:
	`close` (" `cpu_data` ")
	Character string
	command = `"date; uname -a"`
	`system` (*command*)
	Variable that contains a character string
sym	A variable name. Names of array elements are not valid symbols. *sym* can be defined by the `csagfef -D` option. For example:
	`if` (`isdefined` (`ios_e`))
	Variable
	`if` (`isdefined` (`us_stime` [`us_Dispose`]))
	Not valid
	`csagfef -DCPU`
	`if` (`isdefined` (`CPU`))
	Symbol defined by the `csagfef -D` option
unit	May be one of the following:

`B`	Converts to bytes
`KB`	Converts to kilobytes (2^{10} bytes)
`MB`	Converts to megabytes (2^{20} bytes)

`GB`	Converts to gigabytes (2^{30} bytes)
`W`	Converts to words
`KW`	Converts to kilowords (2^{10} words)
`MW`	Converts to megawords (2^{20} words)
`GW`	Converts to gigawords (2^{30} words)
`number`	Uses `number` as the divisor and divides the value by number
`variable_name`	Uses `variable_name` as the divisor

Examples of using the `unit` function parameter follow:

`bytes_to` (`tp_nread, MB`)

Converts bytes to megabytes

`words_to` (`pb_t_phimem_max`, `1000`)

Uses 1000 as the divisor and returns (`pb_t_phimem_max / 1000`)

tunit May be one of the following:

`SEC`	Converts to seconds
`MIN`	Converts to minutes
`HOUR`	Converts to hours
`DAY`	Converts to days

Example:

`clocks_to` (`pb_t_iowtime`, `MIN`)

Convert clocks to minutes

2.3.4.5.8 Built-in Variables

The `csagfef` command has the following built-in variables, as shown in Table 29:

Table 29. Built-in Variables

Variable	Default	Description
`FILENAME`	None	Name of the current input file
`NR`	None	Number of data records read so far
`OFMT`	`%.6g`	Output format for printing numbers
`OFS`	`" "`	Output field separator
`ORS`	`\n`	Output record separator
`RSIZE`	None	Size of the data records in bytes

2.3.4.5.9 Generic Front-end Formatting Example

The extended example presented here illustrates how you can consolidate and format data for NQS requests using `csagcon` and `csagfef`. It assumes input from a session file. The example follows the steps listed in Section 2.3, page 98.

1. Identify the data that needs to be reported.

   Determine the information that is useful to your site. In this case, for each NQS request the example will report the following fields:

   - User name
   - Account name
   - Request name
   - Request ID
   - Queue name

- CPU time
- Memory high-water value
- Queue wait time
- Locked I/O wait time
- Unlocked I/O wait time

Because some of these items are not default `csagcon` consolidation items, you must specify a request file when executing `csagcon`. The following variable names, which are described in Table 14, page 112, through Table 29, page 132, must be in the request file (`nqs.req`). You can find a copy of this file in the `/usr/src/cmd/acct/src/csa/csagfef/examples` directory.

```
nq_reqname
nq_seqno
nq_quename
pb_t_stime
pb_t_utime
pb_t_phimem_max
nq_qwtime
pb_t_iowtime
pb_t_iobtime
```

Pass the request file name (`nqs.req`) to `csagcon` by using the `-R` option (`-R nqs.req`).

2. Select the `csagcon` consolidation keys.

   To extract information for each NQS request, you must select consolidation keys: appropriate job ID (`-j` option) and job class (`-c` option). However, you must be certain that all portions of an NQS request are processed as though they have the same job ID, which is the default. (For this example, do not specify the `-N` option, which consolidates each portion of an NQS request according to its job ID).

   To report the username and account name that is associated with each request, you also must specify the `-u` and `-a` options. If these two keys are not specified, the username and account name will not be known.

Note: All consolidation keys (`acid`, `gid`, `jclass`, `jid`, and `uid`) that are not selected on the `csagcon` command line by the `-a`, `-g`, `-c`, `-j`, and `-u` options, will have a value of -9.

For example, if you do not specify the `-u` option, the `uid` variable will always have a value of -9.

If you want to sort the output, use the `-s` option. In this example, the output is sorted alphabetically by `username` (`-s username` option).

To summarize, the consolidation and sort options used in this example are the following: `-j -c -u -a -s username`.

3. Determine which sessions should be consolidated.

   This example will consolidate only terminated sessions (default option). You can use the `-A` or `-C` option to consolidate all sessions or only active sessions.

   The data to be consolidated now is identified and you are ready to execute `csagcon`. If you assume that the input comes from a session file named `Super-Record.1130` and the output is written to the file `gacct.1130`, you would execute the following command:

```
csagcon -S Super-Record.1130 -o gacct.1130 -R nqs.req -jcua -s username
```

4. Format the consolidated data into a report.

   You must decide the units and length of the various fields. In this example, memory highwater is reported in megawords and CPU time, queue wait time, locked I/O wait time, and unlocked I/O wait time is reported in seconds. Data that is not already in the correct units is converted by `csagfef`. Tables Table 14, page 112 through Table 29, page 132 list the default units of the various fields.

   After deciding on the format, you must write a `csagfef` source script that tells `csagfef` how to generate the report. The following script can be used as input to `csagfef` and is found in the following file:

   `/usr/src/cmd/acct/src/csa/csagfef/examples/nqs.ss`

   The script contains variables that control the writing of the header and summary lines. When `-D HEADER` is specified on the command line, `csagfef` outputs the header. When `-D SUMMARY` is specified, summary information is written.

If you assume that the consolidated data file is named `gacct.1130`, and the source script is named `nqs.ss`, the following command will generate a report without the header and summary lines:

```
csagfef -f gacct.1130 nqs.ss
```

If you want both, the header and summary information, you should execute the following command:

```
csagfef -f gacct.1130 -D HEADER -D SUMMARY nqs.ss
```

The `nqs.ss` source script listing follows.

```
BEGIN {
#
#
#       Figure out which sessions were consolidated.
#
if (con_key & 0100) {
        CONSOL = "ACTIVE AND COMPLETED SESSIONS"
} else if (con_key & 0200) {
        CONSOL = "ONLY ACTIVE SESSIONS"
} else {
        CONSOL = "ONLY COMPLETED SESSIONS"
}

#
#       Initialize counters.
ntot_sess = 0                   # Total number of sessions
nnqs = 0                        # Number of NQS sessions
#
#       Print the header if "-D HEADER" was specified on the command line.
#
if ( isdefined(HEADER) ) {
        printf("%s   DAILY REPORT FOR %s (Rel %s, %s)\n\n",
           strftime("%c", creatime), SYSNAME, RELEASE, VERSION)

        printf("INCLUDES DATA FOR %s BETWEEN\n", CONSOL)
        printf("    %s AND %s\n\n",
           strftime("%c", file_start), strftime("%c", file_end))

        printf("                                               REQUEST  ")
        printf("                 CPU TIME    MEM HIWAT QWAIT   LCK IO  ")
        printf("UNLCK  \n")
        printf("USER NAME    ACCOUNT NAME     REQUEST NAME     ID       ")
        printf("QUEUE NAME       [SECS]      [MW]      [SECS]  WAIT    ")
        printf("IO WAIT\n")

        printf("============ ================ ================ ======== ")
        printf("================ =========== ========= ======= ======= ")
        printf("=======\n")
}
}

ntot_sess++             # count the total number of sessions

if ( jclass == 2 ) {    # output information only about NQS requests
        nnqs++          # count the number of NQS requests

        username = uid2nam(uid)                   # user name
        acname = acid2nam(acid)                   # account name

        cputime = clocks_to(pb_t_stime, SEC) + \ clocks_to(pb_t_utime, SEC)
                                                  # CPU time in seconds
        memhiwat = words_to(pb_t_phimem_max, MW)  # memory high water in megwords

        lockio = clocks_to(pb_t_iowtime, SEC)     # locked I/O wait in seconds
        ulockio = clocks_to(pb_t_iobtime, SEC)    # unlocked I/O wait in seconds

        printf("%-12.12s %-16.16s %-16.16s %-8d %-16.16s ",
           username, acname, nq_reqname, nq_seqno, nq_quename)
        printf("%11.3f %8.0f %7d %7.1f %7.1f\n",
           cputime, memhiwat, nq_qwtime, lockio, ulockio)

}

#
#       Print summary information about the input file if "-D SUMMARY"
#       was specified on the command line.
#
END {
if ( isdefined(SUMMARY) ) {
        printf("\n\nInput file: %s\nTotal number of sessions: %d\n",
           FILENAME, ntot_sess)
        printf("Number of NQS requests: %d\n", nnqs)
        printf("Number of non-NQS requests: %d\n", ntot_sess - nnqs)
}
```

The script above produces the following output. Both the header and summary information are included.

```
Wed Nov 30 10:04:50 1994   DAILY REPORT FOR sn1703c (Rel 9.0.0ao, d90.50)

INCLUDES DATA FOR ONLY COMPLETED SESSIONS BETWEEN
    Wed Nov 30 07:58:09 1994 AND Wed Nov 30 09:51:45 1994

                                    REQ            CPU TIME MEM HIWAT QWAIT  LCK IO  UNLCK
USER NAME ACCOUNT NAME REQUEST NAME ID  QUEUE NAME [SECS]   [MW]      [SECS] WAIT    IO WAIT
========= ============ ============ ==  ========== ======== ========= ====== ======= =======
fe        Xydev        STDIN        3   b_30_1     0.411       1       4       0.1     0.4
fe        Xydev        STDIN        4   b_30_1     0.414       1       3       0.1     0.3
pds       SysAdm       STDIN        6   b_30_1     0.570       0       4       0.0     0.4
root      SysAdm       STDIN        6   b_30_1     0.544       0       0       0.0     0.5
root      SysAdm       SLSCRUB      7   b_1200_1   0.958       0       0       0.0     1.6
root      SysAdm       STDIN        5   b_30_1     0.531       0       0       0.0     0.1
root      Xydev        STDIN        4   b_30_1     0.558       0       0       0.0     0.3
root      Xydev        STDIN        3   b_30_1     0.552       0       0       0.1     0.4
user1     SysAdm       SLSCRUB      7   b_1200_1   2.079       0       9       0.1    14.8


Input file: gacct
Total number of sessions: 175
Number of NQS requests: 10
Number of non-NQS requests: 165
```

Automated Incident Reporting (AIR) [3]

The automated incident reporting (AIR) system allows you to measure overall system availability for the following products:

- Transmission Control Protocol / Internet Protocol (TCP / IP)
- Network Queuing System (NQS)
- Online tapes
- UNICOS kernel
- Unified Resource Manager (URM)

Warning: The AIR feature is not part of a Cray ML-Safe configuration. This section **does not** contain any further warnings or information pertaining to the use of a Cray ML-Safe configuration of the UNICOS system.

3.1 AIR Components Overview

The AIR system consists of four main components, as follows:

- Configuration file
- Coordinator daemon
- Monitoring functions
- Report generator

3.1.1 AIR Configuration File

The AIR configuration file, `/usr/air/config_file`, contains definitions for all the configurable aspects of the AIR system, written in a simple configuration language syntax. All AIR system components refer to this file at initiation for information. Scanning, printing, validation, and translation routines manage the processing of the data in the file.

Caution: AIR configuration file, `/usr/air/config_file`, can be maintained through the UNICOS Installation Menu System (installation tool). If the installation tool is used to maintain this file, it should never be edited manually.

3.1.2 AIR Coordinator Daemon

The AIR coordinator daemon, `aird`, executes configured functions at the specified rates and enacts the return code processing cues.

The coordinator translates the configuration file into a work list consisting of functions associated with each monitored product. The coordinator keeps a running clock, executing the functions with rates indicating that they should be executed. When the coordinator is not executing functions, it waits for function completion. If there are no functions for which to wait, the coordinator sleeps until the next time a function is configured to be executed.

3.1.3 AIR Monitoring Functions

The AIR monitoring functions are product verification processes; these functions can be either shell scripts or executable binaries. The implementation of the functions for each monitored product follows a hierarchical philosophy. Several functions are specified for each monitored product, and they are differentiated by the cost of the resources they use and the aspects of the product that they test. For example, a function that verifies that a process exists on the system would be low cost, and, thus, could be executed more frequently than a function that required a tape mount. However, the higher-cost function provides a better assurance of product verification. These issues must be balanced by the rate specification in the configuration file and be configured for each site’s specific needs.

3.1.4 AIR Report Generator

The AIR report generator collects information, such as the AIR configuration and monitoring function event records from the coordinator log file, and presents product availability and summary information in a text format.

The coordinator reads input from the configuration file and translates that data into a series of event functions for periodic execution. The results of each function’s executions are processed on completion, and pertinent information is written to the coordinator's log file. Periodically, reports can be generated by executing the report generation procedures, as follows:

Command	Description
`airprconf`	Prints AIR configuration file contents from the configuration headers in the `aird` binary log file (see `airprconf`(8)).
`airsum`	Generates availability summary reports based on the `aird`(8) binary log file (see `airsum`(8))
`airtsum`	Generates detailed AIR reports based on the coordinator binary log file (see `airtsum`(8))
`airdet`	Generates detailed AIR reports based on the `aird` binary log file (see `airdet`(8))

3.2 Initiation and Administration

The `aird`(8) process is initiated automatically at system boot time. `aird` is listed in the system `/etc/daemons` file and is started in the same manner as the other system daemons. The `/usr/air/bin/start_air` script contains the sequence used to start the `aird` process; use this script if you need to start `aird` manually. (See `start_air`(8) for more information.)

For systems running AIR, the following scripts are available (to be executed periodically by `cron`) to perform useful, daily functions:

Shell script	Description
`mvfiles`	Moves the AIR log files to data directories, resetting the log files. If you do not execute this script periodically in your system, the binary log file written to by `aird` will grow quite large.
`airdchk`	Monitors `aird`. This script uses the `airexist`(8) command to verify that `aird` exists on the system. Mail is sent to root if `aird` is not running.

These two scripts are located in the `/usr/air/bin` directory. The following example shows `crontab` entries for `mvfiles` and `airdchk`:

```
#
#    AIR utilities for periodic execution by cron.
#    Execute aird checking test every fifteen minutes.
#    Move the log files every Sunday.
#
15 * * * * /usr/air/bin/airdchk
0 0 * * 0 /usr/air/bin/mvfiles
```

3.3 AIR Configuration

The AIR configuration file contains definitions for all configurable aspects of the AIR system. The `aird` process reads the configuration file and translates the contents into monitoring functions that are executed periodically. In addition to initiating its internal processing worklist, `aird` also sets any environment variables specified in the configuration file.

AIR configuration file can be maintained by using the UNICOS Installation Menu System (installation tool). For completeness, the following sections describe both the installation tool menus for AIR and the configuration file itself.

The AIR configuration file is composed of statements written in the AIR configuration language. This section explains the configuration language in detail along with the corresponding installation tool menus, examines the default configuration file, `/usr/air/config_file`, and investigates tuning and validating the configuration file.

3.3.1 Basic Syntactic Rules

The AIR configuration language is composed of a defined set of keywords and their associated arguments. Each line of the configuration file is blank (white space or a new line), a comment (containing a #, text and/or white space), or a keyword and its associated arguments. A comment is permitted on a keyword line.

The following basic syntactic rules apply:

- The configuration language can contain only printable ASCII characters; the parser exits with an error if it finds an unprintable, non-ASCII character in the configuration file.
- Keywords are uppercase names, and user values are lowercase names.
- Noncomment lines begin with a keyword followed by the appropriate arguments.
- Comments in the file are designated like shell comments, beginning with the # character and continuing until an end-of-line is encountered.

- Only one keyword may be present on a line, and it must begin with the first nonwhite character on that line. The `CONFIG` keyword must be the first keyword, and the `ENDCONFIG` keyword must be the last keyword.
- The maximum line length is 4096 characters.
- Legal separators are white space, tabs, colons, commas, and semicolons.
- All path names specified as arguments to keywords must be the full path name of the file. (AIR validation routines ensure that all path names begin with a `/`.)
- All rates specified as arguments to keywords are interpreted as minutes by default. For example, a specification of 300 is interpreted as 5 hours. The `aird -C` option lets you change the conversion factor of the specified arguments (see `aird`(8)).

3.3.2 Configuration Keywords

This section contains lists of available keywords and associated arguments. Refer to the configuration file on your system or to the AIR configuration menu in the installation tool while examining these sections, noting the location and value of each keyword and its arguments. The keywords are discussed in the order they appear in the released configuration file.

3.3.2.1 File Delineation Keywords

The keywords described in the following sections define the beginning and end of the configuration file.

3.3.2.1.1 `CONFIG` *name*

The `CONFIG` *name* keyword marks the beginning of the configuration specification. Only comment or blank lines are allowed before a `CONFIG` line. The *name* argument is the name of the configuration, which is any string.

3.3.2.1.2 `ENDCONFIG` *name*

The `ENDCONFIG` *name* keyword marks the end of the configuration specification. Only comment or blank lines are allowed after an `ENDCONFIG` line. The *name* argument is the name of the configuration, which is any string, but must match the *name* specified with the `CONFIG` keyword in the file.

3.3.2.2 Basic Operational Keywords

In your configuration file, the keywords described in the following sections appear immediately after the `CONFIG` keyword and before a `PRODUCT` or `FUNCTION` keyword specification. These keywords define the basic operational configuration for the AIR system.

3.3.2.2.1 Installation Tool

The operational keywords correspond to the following installation tool menu:

```
M-> Configure system ->
        M-> AIR configuration ->
                M-> Coordinator setup ->

    AIR Coordinator setup

S->  AIR daemon binary log file name          /usr/spool/air/logs/blog
     Monitoring function execution directory  /usr/air/test
     AIR daemon heartbeat rate                15
     AIR daemon ASCII log file name           /usr/spool/air/logs/coord.log
     AIR daemon debugging level               0
     AIR daemon information log level         0
```

3.3.2.2.2 `COORD_LOG` *file*

The `COORD_LOG` *file* keyword specifies the absolute path name to the `aird` ASCII log file. This log file is used for debugging purposes only. Any information needed by the report generators is logged into `aird`'s binary log file. The `COORD_LOGLEV` keyword specifies the number of messages written to this ASCII log.

3.3.2.2.3 `COORD_TESTDIR` *dir*

The `COORD_TESTDIR` *dir* keyword specifies the absolute path name to the directory where the configured monitoring functions are executed.

3.3.2.2.4 `COORD_HBEAT` *rate*

The `COORD_HBEAT` *rate* keyword specifies the rate at which `aird` should log its own heartbeat record into its binary log file. The report generators use this heartbeat record and the configured rate when determining AIR system availability.

3.3.2.2.5 COORD_DEBUG *level*

The COORD_DEBUG *level* keyword specifies the number of diagnostic messages that should be logged to the aird ASCII log file (the location of which is set by using the COORD_LOG keyword). The *level* argument is a number 0 through 20; however, because this keyword is used for debugging purposes only, it is recommended that *level* usually be set to 0.

3.3.2.2.6 COORD_BLOG *file*

The COORD_BLOG *file* keyword specifies the absolute path name to aird 's binary log file. The report generators use this log file when determining the availability of the monitored products. Refer to Section 3.5, page 176, for more information on the contents and use of this file.

3.3.2.2.7 COORD_LOGLEV *level*

The COORD_LOGLEV *level* keyword specifies the number of general informational messages that should be logged to the aird ASCII log file. The *level* argument is a number 0 through 20; however, because this keyword is used for debugging purposes only, it is recommended that *level* usually be set to 0.

3.3.2.2.8 TYPE *tag types*

The TYPE *tag type* keywords are configured in the following installation tool menu:

```
M-> Configure system ->
     M-> AIR configuration ->
          M-> Return tags and types setup ->
```

```
             AIR Return tags and types setup

     Tag Name    Tag Type          Type 1      Type 2       Type 3
     ----------  ----------------  ----------  ----------   ------
E->  PASSED      PROD_AVAILABLE
     FAILED      PROD_UNAVAILABLE
     CHANGED     PROD_AVAILABLE    WARN_ADMIN
     NODEVICE    PROD_AVAILABLE    WARN_ADMIN  WARN_OPS
     NORESERVE   PROD_AVAILABLE    WARN_ADMIN
     TIMEOFDAY   PROD_AVAILABLE
     NOMOUNT     PROD_UNAVAILABLE  WARN_OPS    WARN_ADMIN
     BADWRITE    PROD_UNAVAILABLE
```

```
BADJOB      PROD_UNAVAILABLE
TIMEDOUT    PROD_AVAILABLE    WARN_ADMIN
QSUBFAILED  PROD_UNAVAILABLE
AUDITERROR  PROD_AVAILABLE    WARN_ADMIN
TCPFAILED   PROD_UNAVAILABLE
UDPFAILED   PROD_UNAVAILABLE
ICMPFAILED  PROD_UNAVAILABLE
```

The TYPE *tag types* keyword defines return tags and their associated types. The tags are set as environment variables. aird and the monitoring functions use the tags to communicate. The monitoring functions use the tags as return values. The report generators use the types to report various aspects of the system availability. The tags are also used in the MESSAGE and RETURN keyword arguments, and additional text and subsequent actions are assigned to the tag.

Other than two required tags, PASSED and FAILED, the tag argument assignment is arbitrary; however, the tags defined in the configuration file must match the expected return values for the configured monitoring functions. In other words, for every expected return value in the monitoring functions, a TYPE keyword definition with that return value listed as the tag argument must exist.

The report generators use the PROD_AVAILABLE and PROD_UNAVAILABLE types extensively when determining product availability.

The following types are allowed:

PROD_AVAILABLE	PROD_UNAVAILABLE	PROD_WARNING	SEND_MAIL
SPR	OIR	WARN_OPS	WARN_ADMIN
WARN_USERS	10%	20%	30%
40%	50%	60%	70%
80%	90%	91%	92%
93%	94%	95%	96%
97%	98%	99%	100%

3.3.2.3 Monitored Products Keywords

The keywords in the following sections appear immediately following the TYPES keyword definitions in your configuration file. These keywords specify the products to be monitored by the AIR system and configure the monitoring functions to be used for each specified product. An explanation of how products and functions are configured in the installation tool follows the description of the keywords.

3.3.2.3.1 `PRODUCT` *name status*

The `PRODUCT` *name status* keyword marks the beginning of a product definition and is always paired with the `ENDPRODUCT` keyword. You can define multiple products in a configuration file (within the `CONFIG` and `ENDCONFIG` keyword pair); however, you cannot nest product specifications within other product specifications. You can define single or multiple monitoring functions within each `PRODUCT` / `ENDPRODUCT` pair. The *name* argument, which is indicated in the report generators output, is an arbitrary string but it must be unique within the configuration. The *status* argument indicates whether a product is active (`ON`) or inactive (`OFF`). A product that is inactive is still part of the configuration, but no functions defined within that product are executed.

3.3.2.3.2 `ENDPRODUCT` *name*

The `ENDPRODUCT` *name* keyword marks the end of a specific product specification. This keyword is always paired with the `PRODUCT` keyword.

3.3.2.3.3 `MESSAGE` *tag message*

The `MESSAGE` *tag message* keyword specifies a text message to be associated with the specified tag. As described in the `TYPES` keyword definition, the return tag indicates a status returned by a monitoring function after executing. `aird` sets these return tags to environment variables prior to the execution of the monitoring functions. The report generators use the specified *message* text when reporting availability statistics. The *tag* argument must be one of the tags defined in a `TYPE` keyword specification. The *message* is any arbitrary string. The entire line is limited to 4096 characters.

3.3.2.3.4 `FUNCTION` *name status*

The `FUNCTION` *name status* keyword marks the beginning of a function definition and is always paired with an `ENDFUNCTION` keyword. You can define multiple functions for a product (within a `PRODUCT` / `ENDPRODUCT` keyword pair); however, you cannot nest function specifications within other function specifications. The *name* argument which is indicated in the report generators output, must be unique within a product specification, but does not need to be unique within the configuration. The *status* argument indicates whether a function is active (`ON`) or disabled (`OFF`).

3.3.2.3.5 `ENDFUNCTION` *name*

The `ENDFUNCTION` *name* keyword marks the end of a `FUNCTION` specification. This keyword is always paired with a `FUNCTION` keyword.

3.3.2.4 Monitoring Function Specification

The keywords described in the following sections appear within a `FUNCTION` / `ENDFUNCTION` keyword pair. These keywords are required for a complete monitoring function specification.

3.3.2.4.1 `RATE` *rate*

The `RATE` *rate* keyword specifies the frequency with which `aird` executes the monitoring function. The *rate* argument is interpreted as number of minutes. If the function is meant to be an action type of routine rather than a monitoring function (for example, a function to restart a daemon when a `FAILED` status has been returned to a monitoring function), the argument for `RATE` should be set to `NONE`. This value prevents the function from being executed periodically while allowing it to be executed from other functions. Refer to Section 3.4, page 162, for more information about rate specification on the monitoring functions.

3.3.2.4.2 `EXECUTE` *file*

The `EXECUTE` *file* keyword specifies the absolute path name of the monitoring function that is to be executed.

3.3.2.4.3 `LOGFILE` *file*

The `LOGFILE` *file* keyword specifies the absolute path name of the file in which the function output is placed. If there is no output of interest from the specific monitoring function, set this argument to `NONE`.

3.3.2.4.4 `TIMEOUT` *time*

The `TIMEOUT` *time* keyword specifies the length of time that `aird` should wait for the return of the function. If this time is exceeded, `aird` kills the monitoring function and logs the abnormal termination in its binary log file. The *time* argument can be set to `NONE` to indicate that the function should never be timed out by `aird`.

3.3.2.4.5 `RETURN` *tag value action*

The `RETURN` *tag value action* keyword specifies the return values for the monitoring function. The return *tag* is a tag previously defined in a `TYPE` keyword specification and associated with text from the `MESSAGE` keyword specification. The *value* argument, to which the tag set in the environment is assigned, is an integer between 0 and 200. The *action* argument specifies the name of another function to be executed on the return of the specified tag value from the monitoring function.

The function identified in the action argument must also be defined using the function specifications within the current product specification. The action can also be set to `NONE`, indicating that no further action should be taken on the return of that tag from the monitoring function.

3.3.2.5 Installation Tool Configuration

The configuration of products and their monitoring functions span three interconnected menus in the installation tool:

```
M-> Configure system ->
      M-> AIR configuration ->
          M-> Product enable ->
          M-> Product functions ->
          M-> Function return configuration ->
```

Each of these menus is described below.

3.3.2.5.1 Product Enable Menu

This first menu describes the products and whether they are to be monitored. The menu consists of two fields: the name of the product, and whether it is enabled.

An example of the product enable menu follows:

```
              AIR Product enable
      Product Name     Product Enabled
      ------------     ---------------
E->   disk-integ       YES
      nqs              YES
      tapes            YES
      msgdaemon        YES
      tcp              YES
      urm              YES
```

The menu is used to determine the PRODUCT and ENDPRODUCT keywords in the configuration file.

3.3.2.5.2 Product Functions Menu

This menu contains seven fields that are used to determine the FUNCTION, ENDFUNCTION, RATE, EXECUTE, LOGFILE, and TIMEOUT keywords in the configuration file.

Here is a short example of the product functions menu, focusing on the URM product:

```
             AIR Product functions

    Prod Name Funct Name Enabled Rate  Command Log File                Timeout
     --------- ---------- ------- ----   ----------------------------   -------
E-> urm       function   YES     25    /usr/air/test/urm/urm.funct     NONE
    urm       response   YES     20    /usr/air/test/urm/urm.response  NONE
    urm       existence  YES      10    /usr/air/test/urm/urm.exist     NONE
    urm       restart    NO      NONE  /usr/air/test/urm/urm.restart   NONE
```

The Prod Name field associates these functions under a given product (there must be a product with this name in the Product enable-> menu).

The Funct Name field (FUNCTION) names the function while the Enabled field indicates whether this monitoring function is on or off. The Rate field (RATE) indicates the time in minutes between executions of the monitoring command set in the Command field (EXECUTE). The Log File field (LOGFILE) is the name of a file where the output is to be placed (blank means no log file) and the Timeout field (TIMEOUT) indicates the maximum amount of time the monitoring function can take.

3.3.2.5.3 Function Return Configuration

This menu contains six fields that define the RETURN and MESSAGE keywords. An example showing the function return configuration for the URM product is as follows:

```
       AIR Function return configuration

    Product Name Function Name Return Name  Return Value  Action   Return Message
    ------------ ------------- -----------  ------------  ------   --------------
E-> urm          function      PASSED       0                      Test Passed
    urm          function      FAILED       1                      Test Failed
    urm         response      FAILED      1                       Test Failed
    urm         response      PASSED      0                       Test Passed
    urm         existence     FAILED      1              restart  Test Failed
    urm         existence     PASSED      0                       Test Passed
    urm         restart       FAILED      1                       Test Failed
    urm         1restart      PASSED      0                       Test Passed
```

The `Product Name` and `Function Name` fields are used to associate the return information with a given product (must be a product by this name in the `Product enable->` menu) and function (must be a function by this name for this product defined in the `Product functions ->` menu).

Each function under a product must have a minimum of two return values, named `PASSED` and `FAILED`. These names correspond to an exit status (in this case `0` and `1` respectively) and to a return message text.

Depending on the return value for a function, an action can be performed. This action is specified by the `Action` field and corresponds to the name of another function defined for this product. In the example above, if a `FAILED` status is returned by the `existence` function of the `urm` product (exit status of `1` from `/usr/air/test/urm/urm.exist`), the `restart` function is then started by the `aird` process.

The `restart` function is a special function, since it is not used for monitoring purposes but rather for restarting the URM daemon. Since the restart function should only be started when the `existence` test fails, the `restart` function is configured with a `Rate` (`RATE`) of `NONE`. This indicates that this product is not to be started periodically. Note also that in the above example, the `restart` function is turned off. Therefore, if the `existence` function were to fail, the `restart` function would not be run because it is disabled.

3.3.3 Return Tags

This section contains an overview and summary of the `TYPE`, `MESSAGE`, and `RETURN` keywords and the associated arguments.

The `TYPE` keyword defines return tags and their associated types. The `TYPE` *tag* argument defines a variable to be used as an exit status by a monitoring function. `aird` sets this variable in the environment when it initially processes the configuration file. The *types* associated with each `TYPE` tag are associated with the environment variables that serve as exit statuses for the monitoring functions. The report generators use these types, associated with the tags, in their processing. Only the `PROD_AVAILABLE` and `PROD_UNAVAILABLE` types are used by the report generators at this time for availability determination.

The report generators also use the MESSAGE keyword, which associates a text message with a tag.

The RETURN keyword assigns the tag an environment variable and indicates whether further action should be taken following the return of a monitoring function.

3.3.4 Sample Configuration File

This section contains a sample configuration file and an interpretation of the contents. Read this section if you are unsure of the correct interpretation or if you want to check your understanding. Refer to this sample file as you read the text following the file.

```
# Start of Configuration File Generated by printcf on Thu Feb 21 15:31:40 1991
#
#       :%s./usr./sn1101/soft/os/crs
#
CONFIG  kernel_test_version
        #
        #       Define Coordinator logfile
        #
        COORD_LOG       /usr/spool/air/logs/coord.log
        #
        #       test directory
        #
        COORD_TESTDIR       /usr/air/test
        #
        #       Define Coordinator Heart Beat
        #
        COORD_HBEAT     10
        #
        #       Define Debug level
        #
        COORD_DEBUG     0
        #
        #       Define Binary output file name
        #
        COORD_BLOG      /usr/spool/air/logs/blog
        #
        #       Define ASCII logging level
        #
        COORD_LOGLEV    0
```

```
#
#       Define TYPES
#
TYPE    PASSED      PROD_AVAILABLE
TYPE    FAILED      PROD_UNAVAILABLE
TYPE    CHANGED     PROD_AVAILABLE WARN_ADMIN
TYPE    NODEVICE    PROD_AVAILABLE WARN_ADMIN WARN_OPS
TYPE    NORESERVE   PROD_AVAILABLE WARN_ADMIN
TYPE    NOMOUNT     PROD_UNAVAILABLE WARN_OPS WARN_ADMIN
TYPE    BADWRITE    PROD_UNAVAILABLE
TYPE    BADJOB      PROD_UNAVAILABLE
TYPE    QSUBFAILED  PROD_UNAVAILABLE
TYPE    AUDITERROR  PROD_AVAILABLE WARN_ADMIN
TYPE    TCPFAILED   PROD_UNAVAILABLE
TYPE    UDPFAILED   PROD_UNAVAILABLE
TYPE    ICMPFAILED  PROD_UNAVAILABLE
#
#       Define product disk-integ
#
PRODUCT disk-integ  ON
        MESSAGE FAILED  Test Failed
        MESSAGE PASSED  Test Passed
        #
        #       Define Function response of Product disk-integ
        #
        FUNCTION        response      ON
                RATE    1
                EXECUTE /usr/air/test/kern/kern.response
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     response
        #
        #       Define Function existence of Product disk-integ
        #
        FUNCTION        existence      ON
                RATE    15
                EXECUTE /usr/air/test/kern/kern.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
                ENDFUNCTION     existence
        ENDPRODUCT      disk-integ
       #
        #        #       Define product nqs
        #
        PRODUCT nqs  ON
                MESSAGE PASSED  Test Passed
                MESSAGE FAILED  Test Failed
                MESSAGE QSUBFAILED  Qsub failed during functional test.
                MESSAGE BADJOB  Returned job did not contain expected output
                #
                #       Define Function functional of Product nqs
                #
                FUNCTION        function      ON
                        RATE    15
                        EXECUTE /usr/air/test/nqs/nqs.funct
                        LOGFILE NONE
                        TIMEOUT                 10
                        RETURN  PASSED  0       NONE
                        RETURN  FAILED  1       NONE
                        RETURN  QSUBFAILED  2   NONE
                        RETURN  BADJOB      3   NONE
                ENDFUNCTION     function
                #
                #       Define Function response of Product nqs
                #
                FUNCTION        response      ON
                        RATE    10
                        EXECUTE /usr/air/test/nqs/nqs.response
                        LOGFILE NONE
                        TIMEOUT                 NONE
                        RETURN  FAILED  1       NONE
                        RETURN  PASSED  0       NONE
                ENDFUNCTION     response
                #
                #       Define Function existence of Product nqs
                #
                FUNCTION        existence      ON
                        RATE    5
                        EXECUTE /usr/air/test/nqs/nqs.exist
                        LOGFILE NONE
                        TIMEOUT                 NONE
                        RETURN  FAILED  1       NONE
                        RETURN  PASSED  0       NONE
                ENDFUNCTION     existence
                #
                #       Define Function netexist of Product nqs
                #
                FUNCTION        netexist      ON
                        RATE    5
                        EXECUTE /usr/air/test/nqs/nqsnet.exist
                        LOGFILE NONE
                        TIMEOUT                 NONE
                        RETURN  FAILED  1       NONE
                        RETURN  PASSED  0       NONE
                ENDFUNCTION     netexist
        ENDPRODUCT      nqs
        #
        #       Define product tapes
        #
        PRODUCT tapes  ON
                MESSAGE PASSED  Test Passed
                MESSAGE FAILED  Test Failed
                MESSAGE BADWRITE  Write to tape failed
                MESSAGE NOMOUNT   Mount of tape failed
                MESSAGE NORESERVE Reserve of tape failed
                MESSAGE NODEVICE  No devices available at start of test.
                #
                #       Define Function functional of Product tapes
                #
                FUNCTION        function      OFF
                        RATE    60
                        EXECUTE /usr/air/test/tapes/tape.funct
                        LOGFILE NONE
                        TIMEOUT                 10
                        RETURN  PASSED     0       NONE
                        RETURN  FAILED     1       NONE
                        RETURN  BADWRITE   2       NONE
                        RETURN  NOMOUNT    3       NONE
                        RETURN  NORESERVE  4       NONE
                        RETURN  NODEVICE   5       NONE
                ENDFUNCTION     function
                #
                #       Define Function response of Product tapes
                #
                FUNCTION        response      ON
```

```
                RATE    10
                EXECUTE /usr/air/test/tapes/tape.response
                LOGFILE NONE
                TIMEOUT                 1
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     response
        #
        #       Define Function existence of Product tapes
        #
        FUNCTION        existence      ON
                RATE    5
                EXECUTE /usr/air/test/tapes/tape.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     existence
        #
        #       Define Function avrexist of Product tapes
        #
        FUNCTION        avrexist      ON
                RATE    5
                EXECUTE /usr/air/test/tapes/tapeavr.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     avrexist
ENDPRODUCT      tapes
#
#       Define product msgdaemon
#
PRODUCT msgdaemon  ON
        MESSAGE PASSED  Test Passed
        MESSAGE FAILED  Test Failed
        #
        #       Define Function response of Product msgdaemon
        #
        FUNCTION        response      ON
                RATE    10
                EXECUTE /usr/air/test/msgd/msgd.response
                LOGFILE NONE
                TIMEOUT                 1
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     response
        #
        #       Define Function existence of Product msgdaemon
        #
        FUNCTION        existence      ON
                RATE    5
                EXECUTE /usr/air/test/msgd/msgd.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     existence
ENDPRODUCT      msgdaemon
#
#       Define product tcp
#
PRODUCT tcp  ON
        MESSAGE FAILED  Test Failed
        MESSAGE PASSED  Test Passed
        MESSAGE TCPFAILED Transmission Control Protocol failure
        MESSAGE UDPFAILED User Datagram Protocol failure
        MESSAGE ICMPFAILED Control Message Protocol failure
        #
        #       Define Function functional of Product tcp
        #
        FUNCTION        function      ON
                RATE    10
                EXECUTE /usr/air/test/tcp/tcp.funct
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  PASSED     0    NONE
                RETURN  FAILED     1    NONE
                RETURN  ICMPFAILED 2    NONE
                RETURN  UDPFAILED  3    NONE
                RETURN  TCPFAILED  4    NONE
        ENDFUNCTION     function
        #
        #       Define Function existence of Product tcp
        #
        FUNCTION        existence      ON
                RATE    5
                EXECUTE /usr/air/test/tcp/tcp.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     existence
        #
        #       Define Function gatedexist of Product tcp
        #
        FUNCTION        gatedexist      ON
                RATE    5
                EXECUTE /usr/air/test/tcp/tcpgated.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     gatedexist
        #
        #       Define Function lpdexist of Product tcp
        #
        FUNCTION        lpdexist      ON
                RATE    5
                EXECUTE /usr/air/test/tcp/tcplpd.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     lpdexist
        #
        #       Define Function namedexist of Product tcp
        #
        FUNCTION        namedexist      ON
                RATE    5
                EXECUTE /usr/air/test/tcp/tcpnamed.exist
                LOGFILE NONE
                TIMEOUT                 NONE
                RETURN  FAILED  1       NONE
                RETURN  PASSED  0       NONE
        ENDFUNCTION     namedexist
        #
        #       Define Function ntpdexist of Product tcp
        #
```

```
                FUNCTION        ntpdexist      ON
                        RATE    5
                        EXECUTE /usr/air/test/tcp/tcpntpd.exist
                        LOGFILE NONE
                        TIMEOUT                 NONE
                        RETURN  FAILED  1       NONE
                        RETURN  PASSED  0       NONE
                ENDFUNCTION     ntpdexist
                #
                #       Define Function smailexist of Product tcp
                #
                FUNCTION        smailexist      ON
                        RATE    5
                        EXECUTE /usr/air/test/tcp/tcpsmail.exist
                        LOGFILE NONE
                        TIMEOUT                 NONE
                        RETURN  FAILED  1       NONE
                        RETURN  PASSED  0       NONE
                ENDFUNCTION     smailexist
                #
                #       Define Function snmpdexist of Product tcp
                #
                FUNCTION        snmpdexist      ON
                        RATE    5
                        EXECUTE /usr/air/test/tcp/tcpsnmpd.exist
                        LOGFILE NONE
                        TIMEOUT                 NONE
                        RETURN  FAILED  1       NONE
                        RETURN  PASSED  0       NONE
                ENDFUNCTION     snmpdexist
        ENDPRODUCT      tcp
        #
        #       Define product urm
        #
        PRODUCT urm  ON
                MESSAGE PASSED  Test Passed
                MESSAGE FAILED  Test Failed
                #
                #       Define Function functional of Product urm
                #
                FUNCTION        function      ON
                        RATE    25
                        EXECUTE /usr/air/test/urm/urm.funct
                        LOGFILE NONE
                        TIMEOUT                 NONE
                        RETURN  PASSED  0       NONE
                        RETURN  FAILED  1       NONE
                ENDFUNCTION     function
                #
                #       Define Function response of Product urm
                #
                FUNCTION        response      ON
                        RATE    20
                        EXECUTE /usr/air/test/urm/urm.response
                        LOGFILE NONE
                        TIMEOUT                 NONE
                        RETURN  FAILED  1       NONE
                        RETURN  PASSED  0       NONE
                ENDFUNCTION     response
                #
                #       Define Function existence of Product urm
                #
                FUNCTION        existence      ON
                        RATE    10
                        EXECUTE /usr/air/test/urm/urm.exist
                        LOGFILE NONE
                        TIMEOUT                 NONE
                        RETURN  FAILED  1       restart
                        RETURN  PASSED  0       NONE
                ENDFUNCTION     existence
                #
                #       Define Function restart of Product urm
                #
                FUNCTION        restart        OFF
                        RATE    NONE
                        EXECUTE /usr/air/test/urm/urm.restart
                        LOGFILE NONE
                        TIMEOUT                 NONE
                        RETURN  FAILED  1       NONE
                        RETURN  PASSED  0       NONE
                ENDFUNCTION     restart
        ENDPRODUCT      urm
ENDCONFIG       kernel_test_version
#
# End of Configuration File Generated by printcf on Thu Feb 21 15:31:40 1991
#
```

At the top of the configuration file, the global operational keywords are defined. The ASCII log and binary log files for the `aird` process are set to `/usr/spool/air/logs/coord.log` and `/usr/spool/air/logs/blog`, respectively; however, the ASCII and debug logging levels are both set to 0, which means that the `aird` ASCII log file should remain empty. The rate that the `aird` process logs its own heartbeat record is set to every 10 minutes.

Next, the return tags and associated types are defined. The required `PASSED` and `FAILED` tags are at the top of the specification block. Also, the `PROD_AVAILABLE` and `PROD_UNAVAILABLE` types associated with the various tags are specified.

The next component of the file is the product's specification, the first one being the disk-integ. Messages are assigned to the two required tags, `PASSED` and `FAILED`, and two functions, response and existence, are defined. The response function is configured to execute every 90 minutes. No log file or time-out limit is specified, and no subsequent action is defined for the two required return tags. The existence function is configured to execute every 15 minutes and has no output, time-out, or subsequent actions specified. The actual monitoring functions are found in `/usr/air/test/kern/kern.response` and `/usr/air/test/kern/kern.exist`, respectively.

3.3.5 Configuration File Tuning and Validation

You can change the contents of the configuration file by using the UNICOS Installation Menu System (installation tool) validation. You can validate your configuration through the `airckconf`(8) command prior to putting that file into production on your system. The `airckconf` command uses the same validation routines that the `aird` process uses on the contents of the configuration file.

The AIR configuration menu in the installation tool allows you to verify your configuration before activating it. Simply select the `Verify air configuration ...` action in the AIR configuration menu. This action will generate a test configuration file based on your selections and then run the `airckconf`(8) command on it.

An example of an area that you may need to change is the execution rates for the monitoring functions.

Note: Before making changes to the execution rates of the functions, please read through Section 3.4, page 162, to determine the appropriate balance for your system.

Changing the rates is accomplished by editing the argument for the `RATE` keyword in a specific function (or the `Rate` field in the `Product functions ->` menu of the installation tool). For example, if you want the existence function for URM to run once every 10 minutes instead of the default of 5 minutes, you would change the `RATE` keyword argument from 5 to 10 in the existence function specification of the URM product specification.

Any associated keyword arguments can be changed in the same manner. To add functions or products, refer to Section 3.4.6, page 167.

3.4 Monitoring Functions

The *monitoring functions* are the actual product verification processes and are either shell scripts or executable binaries. The implementation of the functions for each monitored product follows a hierarchic philosophy. Several functions are specified for each monitored product, and they are differentiated by the cost of the resources they use and the aspects of the product that they test.

For example, a function that verifies that a process exists on the system would be low cost, and, thus, could be executed more frequently than a function that required a tape mount. However, the higher-cost function provides a better assurance of product verification. These issues must be balanced by the rate specification in the configuration file, and be configured for each site's specific needs.

This section discusses the product testing coverage provided by the monitoring functions. This section also describes functions that need to be configured on a site-by-site basis and the procedures for adding those functions and other products, to the AIR system.

Caution: Since the test scripts installed under `/usr/air/test` can be overwritten during a software update, local changes to the supplied monitoring functions should be made by copying the file to another directory (or renaming it) and then altering this copy. Be sure to update the configuration of AIR to point to the local copy of the monitor script.

Refer to the individual monitoring function man pages for more detailed information on the contents of the individual tests.

Monitoring functions are provided for TCP/IP, NQS, online tapes, URM, and disk-integ (general system checks). The functions for these products are divided into the following types determined by the aspect of the product they are testing:

- Existence
- Response
- Functional invocation

The *existence* functions are low-cost verification routines that check for processes on a system. The *response* functions are also lower-cost routines that cause a product to respond in some manner, but do not cause significant operational changes on a system. The *functional invocation* routines can be high-cost functions. These functions are responsible for causing the product to accomplish work in the same way a user would. The specified rates in the configuration file for each of these functions reflects the cost involved; the existence and response functions' rates are much higher than those for the functional invocation tests. Again, these rates are site-configurable.

The `airexist`(8) command verifies product component existence.

It is important to note that these functions are opaque objects in the AIR system. Although they are grouped as described previously, they are by no means restricted to those types. The basic structure of the `aird` process and monitoring functions allows maximum flexibility in monitoring the products.

The following sections describe the product testing coverage and the functions that can be configured for each product.

3.4.1 TCP/IP

Existence and functional monitoring functions are available for TCP/IP.

The existence functions use the `airexist` command for process verification. There are separate existence functions for each of the following processes:

- Internet services daemon (`inetd`)
- Gateway routing daemon (`gated`)
- Line printer daemon (`lpd`)
- Internet domain name server (`named`)
- Time synchronization daemon (`ntpd`)

- Mail daemon (`sendmail`)
- SNMP daemon (`snmpd`)

These daemons are checked separately so that a site can disable any of the functions that check processes not configured on their system.

The functional invocation test uses an enhanced version of `ping` to send `ECHO` packets to network hosts using the following protocols:

- Internet Control Message Protocol (ICMP)
- Internet Transmission Control Protocol (TCP/IP)
- Internet User Datagram Protocol (UDP)

Failure to receive the expected `REPLY` packet constitutes a failed status.

The existence functions for TCP/IP check for the following processes, which must be configured on the system:

Function	Process checked
`tcp.exist`	`inetd`
`tcpgated.exist`	`gated`
`tcplpd.exist`	`lpd`
`tcpnamed.exist`	`named`
`tcpntpd.exist`	`ntpd`
`tcpsmail.exist`	`sendmail`
`tcpsnmpd.exist`	`snmpd`

If you are not using one of the listed processes, you must disable the associated function in the configuration file. If you do not disable a function that checks for a process that does not exist, TCP/IP will always be reported as available 0%.

3.4.2 NQS

Existence, response, and functional monitoring functions are available for NQS.

The existence functions use the `airexist` command for process verification. There are separate existence functions for each of the following processes:

- Main NQS daemon (`nqsdaemon`)

- TCP/IP networking component (netdaemon)

These daemons are checked separately so that a site can disable any of the functions that check processes not configured on their system.

The response function uses the nqsresp command to verify that the nqsdaemon process is able to read its named pipe. The nqsresp command attempts to open the named pipe; if the open fails, this function returns a failed status (see nqsresp(8)).

The functional invocation test uses the NQS qsub command to submit a job. The test then verifies that the expected string is found in the job output file. This function must also be configured to use a valid enabled, started, and running batch queue.

The existence functions for NQS check for the following processes, which must be configured on the system:

Function	Process checked
nqs.exist	nqsdaemon
nqsnet.exist	netdaemon

If you are not using one of the listed processes, disable the associated function in the configuration file. If you do not disable a function that checks for a nonexistent process, NQS will always be reported as available 0%.

Because it invokes a generic qsub(1) command, the nqs.funct function monitoring test is dependent on an enabled, started, and running default batch queue. If you cannot ensure that a default batch queue is always available, edit nqs.funct so that the job is directed to an available queue by using the appropriate qsub options.

3.4.3 Online Tapes

Existence, response, and functional monitoring functions are available for online tapes.

The existence functions use the airexist command for process verification. There are separate existence functions for each of the following processes:

- Tape daemon (tpdaemon)
- Message daemon (msgd)
- AVR component (avrproc)

These daemons are checked separately so that a site can disable any of the functions that check processes not configured on their system.

The `tpstat`(1) command is a tape status command that forces a response from the `tpdaemon` process. The response function determines the ability of the `tpdaemon` to respond based on the return value from `tpstat`. The response function for the `msgd` process invokes the `msgd` command, and redetermines the ability for response based on the return value.

The function invocation test uses the `dd`(1) command to write a file to tape by using the tape daemon. This test serves as a template only. Due to the variance of device groups, label types, and volume serial numbers, each site needs to modify this script to reflect their configuration.

If you are not using AVR, disable the `tapeavr.exist` function in the AIR configuration file. If AVR is nonexistent and you do not disable `tapeavr.exist`, online tapes will always be reported as available 0%.

The `tape.funct` function monitoring test uses the `tpmnt`(1) command to mount a tape on a drive, and then uses the `dd` command to write data to that tape. Edit this test to reflect your local tape environment, paying particular attention to the `DEVGRP`, `LABEL`, and `VOLSER` variables and the time check at the top of the script. By default, this test executes between 10 A.M. and 4 P.M.; otherwise, it returns a passed exit status without execution.

3.4.4 Disk-integ

System monitoring functions that check existence and integrity are available for the kernel.

The existence script invokes various user commands, such as `cd`, `ls`, and `cat`, to verify that the kernel is working and to gain some idea of the interactive response time.

The integrity function checks the following:

- Unrecovered disk errors
- File system free space and inodes

This script is configured to send mail to the appropriate administrators with pertinent information.

3.4.5 URM

Existence, response, and functional monitoring functions are available for the Unified Resource Manager (URM).

The existence function uses the `airexist` command to determine if the `urmd` process exists in the system.

The `rmgr` command is a URM status command that forces a response from URM. The response function determines the ability of URM to respond based on the return value from `rmgr`.

The functional invocation test uses the URM `rmgr` command to send a `REGISTER` request to `urmd` to verify whether or not `urmd` is initialized and functioning. The URM product also has a special function called `restart`. The function is not a monitoring function, but rather, an action. The `existence` function defines it to be started in response to a `FAILED` exit status (the `urmd` process not running) in order to restart the `urmd` process. Since it is not a monitoring function, its `RATE` is set to `NONE` to indicate to `aird` that it should not be periodically started.

In the preceding example configuration, the `restart` function is turned off. In order for the `restart` function to be started by the `aird` process in response to the failure in `existence`, it must be turned on.

3.4.6 Adding Products and Functions

In the AIR system, you can easily extend product sets and enhance monitoring functions.

This section describes the addition of products and monitoring functions to the AIR system.

Follow these steps in order to add a product to the AIR system:

1. Create the appropriate monitoring functions for the product.
2. Create the appropriate directory in the test directory tree, and move the monitoring functions to that directory.
3. Add the product and monitoring function specifications to the configuration file.
4. Use the `airckconf` command on the newly edited configuration file to validate your changes.

5. Move the file into production; send a `SIGHUP` signal to the `aird` process.
6. Check to ensure that the added functions are working as expected.

3.4.6.1 Creating Functions

The first step in adding a product to the AIR system is creating the appropriate monitoring functions. This step is quite important because the accuracy and effectiveness of product monitoring depends on the quality of the monitoring functions. The creation method described in this section is recommended, but because the functions are opaque in the system, you may use any method that works for your site.

In this example, the data migration product and functions are added to the AIR system by using the same hierarchical implementation as the released functions. These functions are incomplete and meant only as examples; however, you can use them as the basis for monitoring the data migration facility.

The following example shows an existence function for the `dmdaemon` process:

```
#! /bin/sh
#
#       dmf.exist       Product DMF existence test.
#
#       Test Description:
#               This is an existence test for DMF.  It checks for the
#               existence of the following process:  dmdaemon.
#
#       Dependencies:
#               The environment variable PASSED must be set.
#               The environment variable FAILED must be set.
#               The airexist command is installed in /usr/air/bin.
#
PATH=$PATH:/usr/air/bin

airexist -u 0 dmdaemon
if [ $? -eq 1 ]
then
        EXITVAL=$PASSED
else
        EXITVAL=$FAILED
fi

exit ${EXITVAL}
```

```
#
#       End of product DMF existence test.
#
```

Functional monitoring of data migration could be accomplished in one of two ways:

- Attempting to migrate and restore a file to online tape media
- Attempting to migrate and restore a file to the MVS station

The following example shows the online tape function test; the MVS front-end function would be identical to this one except that the specified media would be the MVS front end:

```
#! /bin/sh
#
#       dmftape.funct   Product DMF online tape functional test.
#
#       Test Description:
#               This is a functional test for the online tape media
#               capability of the data migration facility.  Note that
#               this script, through the forced migration of a file,
#               calls for a tape mount and should be run at a rate
#               appropriate for the site operational personnel and tape
#               environment.
#
#       Dependencies:
#               The environment variable PASSED must be set.
#               The environment variable FAILED must be set.
#               The environment variable PUTFAIL must be set.
#               The environment variable NOMIGRATE must be set.
#               The environment variable GETFAIL must be set.
#               The MIGRATE_FILE must be defined and exist.
#
MIGRATE_FILE="site defined"
MEDIA_TYPE=1

#
#       Migrate the file.  Indicate that the tape media should be used.
#
dmput -p ${MEDIA_TYPE} MIGRATE_FILE
if [ $? -ne 0 ]
```

```
then
        exit ${PUTFAIL}
fi

#
#       Verify that the file has at least been premigrated.
#
if [ !(-M MIGRATE_FILE) ]
then
        exit ${NOMIGRATE}
fi

#
#       Restore the file.
#
dmget MIGRATE_FILE
if [ $? -ne 0 ]
then
        exit ${GETFAIL}
fi

#
#       Verify that the file has been returned.
#
if [ "‘ls -l MIGRATE_FILE | cut -c1‘" = "m" ]
then
        exit ${FAILED}
else
        exit ${PASSED}
fi

#
#       End of product DMF online tape functional test.
#
```

3.4.6.2 Integrating the Functions

After you have created the monitoring functions, you must create the appropriate directory in the test directory tree and place the new functions in that directory.

In this example, you would create the `/usr/air/test/dmf` directory. Then you would copy the `dmf.exist`, `dmftape.funct`, and `dmfmvs.funct` monitoring functions created in the previous section into that directory.

3.4.6.3 Configuring the Functions

After you create the monitoring functions and place them in the test directory, you must follow these steps to instruct the `aird` process what to do with these monitoring functions. (Refer to Section 3.3.2, page 143, for a discussion of the configuration file and information concerning the specific keywords and their associated arguments in the configuration language syntax. There is a section for users of the UNICOS Installation Menu System, as well as for those who manually update the configuration file.)

3.4.6.3.1 Manual Function Configuration

Use the following procedures to manually add new products/functions to the AIR configuration file.

1. This step is necessary only if you are not using the installation tool to configure AIR.

   Copy the configuration file, `/usr/air/config_file`, to a temporary file, as in the following example:

   ```
   $ cp /usr/air/config_file /tmp/tmp_config_file
   ```

2. Using a standard text editor, add the product and monitoring functions' specification to the file, in the following two-step process:

   a. Add the additional return types (`PUTFAIL`, `NOMIGRATE`, and `GETFAIL`) used in the functional tests to the `TYPES` definition area near the top of the configuration file, as in the following example:

   ```
   #
   #       Define TYPES
   #
   TYPE    PASSED      PROD_AVAILABLE
   TYPE    FAILED      PROD_UNAVAILABLE
   TYPE    CHANGED     PROD_AVAILABLE WARN_ADMIN
   TYPE    NODEVICE    PROD_AVAILABLE WARN_ADMIN WARN_OPS
   TYPE    NORESERVE   PROD_AVAILABLE WARN_ADMIN
   TYPE    NOMOUNT     PROD_UNAVAILABLE WARN_OPS WARN_ADMIN
   TYPE    BADWRITE    PROD_UNAVAILABLE
   TYPE    BADJOB      PROD_UNAVAILABLE
   ```

```
TYPE    QSUBFAILED  PROD_UNAVAILABLE
TYPE    AUDITERROR  PROD_AVAILABLE WARN_ADMIN
TYPE    TCPFAILED   PROD_UNAVAILABLE
TYPE    UDPFAILED   PROD_UNAVAILABLE
TYPE    ICMPFAILED  PROD_UNAVAILABLE
TYPE    PUTFAIL     PROD_UNAVAILABLE WARN_OPS WARN_ADM
TYPE    NOMIGRATE   PROD_UNAVAILABLE WARN_OPS WARN_ADM
TYPE    GETFAIL     PROD_UNAVAILABLE WARN_OPS WARN_ADM
```

Although the WARN_OPS and WARN_ADMIN return types are not implemented for use, they will be important pieces of information for the real-time evaluation of the monitored products.

b. Add the product and functions specification anywhere below the TYPES area, as in the following example:

```
ENDPRODUCT tcp
#
#       Define Product DMF
#
PRODUCT dmf     ON
   MESSAGE PASSED       Test Passed
   MESSAGE FAILED       Test Failed
   MESSAGE PUTFAIL      Migration of file failed
   MESSAGE NOMIGRATE    Expected migrated file not migrated
   MESSAGE GETFAIL      Restore of file failed
   #
   #    Define function existence of Product DMF
   #
   FUNCTION     existence       ON
        RATE    5
        EXECUTE /usr/air/test/dmf/dmf.exist
        LOGFILE NONE
        TIMEOUT NONE
        RETURN  PASSED 0        NONE
        RETURN  FAILED 1        NONE
   ENDFUNCTION  existence
   #
   #    Define function online functional of Product DMF
   #
   FUNCTION     tapefunct       ON
        RATE    60
        EXECUTE /usr/air/test/dmf/dmftape.funct
        LOGFILE NONE
```

```
             TIMEOUT 10
             RETURN  PASSED    0     NONE
             RETURN  FAILED    1     NONE
             RETURN  PUTFAIL   2     NONE
             RETURN  NOMIGRATE 3     NONE
             RETURN  GETFAIL   4     NONE
        ENDFUNCTION  tapefunct
        #
        #    Define function MVS functional of Product DMF
        #
        FUNCTION     mvsfunct        ON
             RATE    60
             EXECUTE /usr/air/test/dmf/dmfmvs.funct
             LOGFILE NONE
             TIMEOUT 10
             RETURN  PASSED    0     NONE
             RETURN  FAILED    1     NONE
             RETURN  PUTFAIL   2     NONE
             RETURN  NOMIGRATE 3     NONE
             RETURN  GETFAIL   4     NONE
        ENDFUNCTION  mvsfunct
     ENDPRODUCT dmf
```

3.4.6.4 Function Configuration through the Installation Tool

This section describes how to use the UNICOS Installation Menu System (installation tool) to add new products and functions to AIR.

1. Enter the `Return tags and types setup ->` submenu under the AIR configuration menu. Add the new return types (`PUTFAIL`, `NOMIGRATE`, and `GETFAIL`) used in the functional tests as follows:

```
                         AIR Return tags and types setup

     Tag Name     Tag Type           Type 1       Type 2       Type 3
     ----------   ----------------   ----------   ----------   ------
     ...
E->  PUTFAIL      PROD_UNAVAILABLE   WARN_OPS     WARN_ADM
     NOMIGRATE    PROD_UNAVAILABLE   WARN_OPS     WARN_ADM
     GETFAIL      PROD_UNAVAILABLE   WARN_OPS     WARN_ADM
     ...
```

2. Add the product to the product list in the `Product enable ->` menu:

```
                    AIR Product enable

              Product Name     Product Enabled
              ------------     ---------------
              disk-integ       YES
              nqs              YES
              tapes            YES
              msgdaemon        YES
              tcp              YES
              urm              YES
        E->   dmf              YES
```

3. Create the three functions in the `Product functions ->` menu:

```
       AIR Product functions

     Prod Name  Funct Name  Enabled  Rate  Command Log File               Timeout
     ---------  ----------  -------  ----  -----------------------------  -------
E->  dmf        mvsfunct    YES      60    /usr/air/test/dmf/dmfmvs.funct  10
     dmf        tapefunct   YES      60    /usr/air/test/dmf/dmftape.funct 10
     dmf        existence   YES      5     /usr/air/test/dmf/dmf.exist     NONE
```

4. Create the function return information through the `Function return configuration ->` menu:

```
                                AIR Function return configuration

    Product Name   Function Name   Return Name   Return Value   Action   Return Message
    ------------   -------------   -----------   ------------   ------   --------------
E-> dmf            tapefunct       FAILED        1                       Test Failed
    dmf            tapefunct       PASSE         0                       Test Passed
    dmf            tapefunct       PUTFAIL       2                       Migration of file failed
    dmf            tapefunct       NOMIGRATE     3                       Expected migrated file not migrated
    dmf            tapefunct       GETFAIL       4                       Restore of file failed
    dmf            mvsfunct        FAILED        1                       Test Failed
    dmf            mvsfunct        PASSED        0                       Test Passed
    dmf            mvsfunct        PUTFAIL       2                       Migration of file failed
    dmf            mvsfunct        NOMIGRATE     3                       Expected migrated file not migrated
    dmf            mvsfunct        GETFAIL       4                       Restore of file failed
    dmf            existence       FAILED        1                       Test Failed
    dmf            existence       PASSED        0                       Test Passed
```

3.4.6.5 Validating Configuration

All essential components are in place for monitoring a new product. Before placing the new configuration file into production, however, you should validate the changes and additions that have been made. The `airckconf`(8) command runs the configuration file through the same verification routines that the `aird` process uses during its internal processing initiation. The following sample command line verifies the temporary configuration file produced in the previous section:

```
$ airckconf tmp_config_file
```

If you are using the installation tool, perform the `Verify air configuration ...` action to verify that the changes made in the menu system are correct.

For more detailed information, see `airckconf`(8).

> **Note:** Do not proceed to the next step until `airckconf` is executing without error on the new configuration file.

3.4.6.6 Production

Before copying the new configuration file over the current one, make a back-up copy of `/usr/air/config_file`. If you are using the installation tool to update the AIR configuration, the backup is not necessary, since backups are made automatically. After creating a back-up copy and copying in the new configuration file (activating the configuration if using the installation tool), either start the `aird` process, using the `/usr/air/bin/start_air` script, or, if `aird` is already running, send the `aird` process the `SIGHUP` signal. Use the `ps`(1) command to determine the process ID of `aird` and use the `kill`(1) command to send the `SIGHUP` signal to that *pid*, as in the following example:

```
$  ps -el | grep aird

0 S     0 12954     1  0  39  24   26236     98     561  -    0:07 aird

$  kill -1 pid
```

By sending the `SIGHUP` signal, you cause the `aird` process to break out of its processing loop and reread the configuration file, thus adding the changes to its internal work list.

3.4.6.7 Final Verification

Verify that the new monitoring functions are operating as expected. Wait until all added functions have executed several times; the time to wait depends on the configured execution rates for the functions. Use the `airdet`(8) command to examine the records that have been logged pertaining to those functions, as in the following example:

```
airdet -p dmf -dmt /usr/spool/air/logs/blog
```

If you receive messages, final verification of the process is complete.

3.5 Using the Report Generators

This section examines the use of the four report generator commands, `airprconf`(8), `airdet`(8), `airtsum`(8), and `airsum`(8). The section includes a discussion of the record types used as input for each generator, an explanation of the output from each generator, and an analysis of a binary log file that highlights how you can use these commands and how they interact.

The availability numbers produced by the AIR report generators indicate the length of time a monitored product is available to a monitoring function. Whether the reported availability is the true availability depends on the quality of the monitoring functions and the AIR configuration. For example, if you configure your functional tests to run once a day, the numbers reported and the actual availability may be quite different for a product. Through careful crafting of the monitoring functions, and thorough configuring of the systems, accurate and detailed availability statistics can be obtained.

It is recommended that you read the man pages for the four report generators before reading this section, and that you keep the text of the man pages available for reference.

3.5.1 Record Types

The four report generator commands provide extensive and tunable selection criteria for the presentation of the availability statistics from the data gathered by the automated incident reporting system (AIR). This data consists of the binary records logged by the `aird`(8) process into its binary log file.

The following record types make up the contents of the `aird` binary log file:

Record type	Description
Configuration header	When initiated, the `aird` process reads the configuration file and translates the contents of that file into its own internal processing worklist. After completing the file translation, the `aird` process logs a configuration header record that delineates the contents of the configuration file just processed. If the `aird` process receives a `SIGHUP` signal, it breaks out of its internal processing loop and rereads the configuration file. After processing the configuration file, the `aird` process then logs another configuration header record into its binary log file.
Event	Upon the return of each configured monitoring functions, the `aird` process logs an event structure denoting the product and function names, the start and end times of the function, and the return type structure as specified in the configuration file.
Heartbeat	At a configured rate, the `aird` process logs a heartbeat record from which subsequent AIR system availability can be determined.
Time-out	If a monitoring function exceeds the specified `TIMEOUT` value, the `aird` process kills the test and logs a message indicating the abnormal termination.

The following list contains the AIR commands and descriptions of the reports they provide:

Command	Description
`airprconf`	Reads the configuration header records logged by the `aird` process and prints the contents of the configuration files that the records represent
`airdet`	A detailed reporting mechanism that reads and prints event, heartbeat, and time-out records
`airtsum`	Collates event records and prints a summary of statistics for the configured monitoring functions

`airsum`	Prints statistics on the availability of each monitored product by using event and heartbeat records

3.5.2 Output

This section discusses the output information printed by each generator command.

> **Note:** Because the `airprconf` and `airdet` commands merely log available statistics and do no collation or summarization, no interpretation of the output they produce is needed. However, the reports produced by the `airtsum` and `airsum` commands do require interpretation and this section contains the explicit derivation for the numbers found in the reports.

3.5.2.1 Using the `airprconf` Command

The default report generated by the `airprconf`(8) command contains the following information:

- Time the configuration header record was written to the binary log file by the `aird` process
- Number of types, messages, products, and functions defined
- Mapping of values and tags to the types
- Messages and functions defined for each product

For an example of the default report, see `airprconf`(8). Refer to Section 3.3.2, page 143, for a discussion of the configuration file and its contents, and for further information regarding the return types and their mappings.

If the `-P` option is specified, `airprconf` prints the configuration as it appears in the configuration file as it was in that period. All information found in the original configuration file is printed. Refer to Section 3.3.2, page 143, when interpreting this output.

3.5.2.2 Using the `airdet` Command

The default report generated by the `airdet` command consists of all event, heartbeat, and time-out records found in the `aird` binary log file. For each record, the product and function names and the message type are shown. For an example of a default report, see `airdet`(8).

The options for `airdet` allow you to change the selection of records and the information displayed for each selected record.

The selection criteria contain the following capabilities:

- The `-b` and `-e` options let you specify the range of time within which the records must fall in order to be selected. This option is helpful if you want to analyze only certain time periods.
- The `-p` and `-f` options let you limit the printed records to those from the given product or function, respectively.
- The `-n` option lets you specify a time so that the only records printed are those whose elapsed time exceeds the specified time. This option provides a back-end implementation of *noticing* for the AIR data. *Noticing* is the capability to indicate, or provide notice, when a particular record's elapsed time has exceeded a specified time. This capability can be useful if you want to highlight periods of time when a system may have been experiencing performance problems.
- The `-T` and `-O` options let you isolate the specified return type tags. These options are helpful if you are searching for specific as well as general product status. Refer to Section 3.3.2, page 143, for an explanation of the return type tags and what they indicate about their associated products.

The information criteria contain the following capabilities:

- The `-l` option outputs the elapsed time of the given record. If `-l` is specified with the `-n` option, `airdet` verifies the specified status.
- The `-t` option outputs the time stamps (beginning and end) for the selected records.
- The `-m` option outputs the text associated with the return type found in the record. This further denotes the status of the selected record.
- The `-h` option provides headers for the record delineation.

You can use the options previously described in a myriad of ways to assist you in analyzing the data collected by the AIR system. The `airdet`(8) man page describes basic concepts and provides examples to help you easily analyze a binary log file.

The following usage tips may be useful:

- When attempting to observe a particular range of time, use the `-b` and `-e` options.

- When attempting to observe particular products or functions, use the `-p` or `-f` options, respectively.
- When attempting to isolate product status, use the `-T` and `-O` options, keying off the desired status type.
- When attempting to identify abnormal elapsed times, use the `-n` option. You can use either the `airtsum` or `airsum` command with the `-E` option if you need to isolate a certain period of time. This option prints summary reports for each configuration header encountered. The default action is to display only the summary information over the range of files specified on the command line and not denote the smaller samples contained within.

Note: If the `-E` option is specified, both commands generate much more information, depending on the number of `SIGHUP` signals, machine boots, and AIR system startups contained within the specified binary log file.

3.5.2.3 Using the `airtsum` Command

The `airtsum` command prints summary statistics for the monitoring functions. This section discusses each column of information available within this report and the options associated with those columns. Note that the first four columns contain the default information printed in these reports.

Column	Description
`Product Name`	Name of the monitored product. Each of the configured products is displayed.
`Function Name`	Name of the monitoring tests. The monitoring functions specified for each of the monitored products are displayed.
`Total Executed`	Number of monitoring tests that returned. This number is really a count of the records logged by the `aird` process on return of each specific monitoring function. This number reflects the number of times that a specific monitoring function was executed during the given sample time of AIR data.

`Total Time Tested`

Time over which a specific function was monitoring its product. This time begins at the start time of the first record logged for the given function, and ends at the end time of the last record logged for the given function.

`From Time/Until Time`

(Displayed using the `-S` option) Beginning and end times that a specific function was monitoring its product. These times are used in the calculation of the total time tested.

`Percent Time`

(Displayed using the `-p` option) Percentage of time that the product was monitored by the specific function. This percentage is calculated by dividing the total time the specific function tested the product by the total time that UNICOS was running. The total time that UNICOS was running is calculated by subtracting the end time of the last record read from the boot time of the system. The equations for these calculations are as follows:

%_tested = *total_time_tested* / *total_system_time*
total_system_time = *system_boot_time* - *last_end_time*

`Return Type/Number Returned`

(Displayed using the `-r` option) Breakdown of each return type for each monitoring function. Each return type and the number of times that each type was returned are displayed for each configured monitoring function. The return numbers indicate the number of records logged by the `aird` process that contained the specific return type.

In the following columns, intervals are determined by subtracting the end time of the previous record of the same function type from the start time of the current record:

Column	Description
`Long Interval`	(Displayed using the `-l` option) Longest period of time between executions of the specific monitoring function

`Short Interval`

(Displayed using the `-s` option) Shortest period of time between executions of the specific monitoring function

`Average Interval`

(Displayed using the `-a` option) Average period of time between executions of the specific monitoring function

`Configured Interval`

(Displayed using the `-c` option) Period of time that should transpire between the execution of the specific monitoring function, as specified in the configuration file

The `airtsum` and `airsum` commands also accept `-b` and `-e` options, which specify sample times. They are used in the same manner as described previously in the `airdet` discussion.

3.5.2.4 Using the `airsum` Command

The `airsum` command reports summary statistics on the availability of each of the monitored products. The formats of report information are described in the following sections.

3.5.2.4.1 `Default Summary Information` Section

The `Default Summary Information` section contains default output from the `airsum` command. Specifically, this section contains the following basic availability information for each of the monitored products:

Column	Description
`Product Name`	Name of the monitored product. Each of the configured products are displayed.
`Total Time Available`	Entire time that the monitored product was available to its monitoring functions.

`Total Time Unavailable`

Entire time that the monitored product was unavailable to its monitoring functions.

`Relative Percentage Available`

Percentage of time that the monitored product was available with respect to the total time that the `aird` process was available.

`Real Percentage Available`

Percentage of time that the monitored product was available with respect to the total time that the system was available.

3.5.2.4.2 Product Availability Breakdown Section

When the records in the `aird` binary log file are processed, the final step is to determine the availability state of each product. A product's *state* indicates whether it is available or unavailable, and also the time during which this status is in effect. The state is determined not only by a change in availability, but also by configuration restarts either through `SIGHUP` signals or system boots. The *availability breakdown* for a product is the complete record of any availability changes; this section of the report contains availability breakdowns for each product and can be printed using the `-B` option. The report contains the following columns of information:

Column	Description
`Product Name`	Name of the monitored product. Each of the configured products is displayed.
`Product Status`	Status of each of the product's states. The status can be either `available` or `unavailable`.

`From Time/Until Time`

Range of time that the specific product was in the current state.

3.5.2.4.3 `Summary Information for Periods` Section

You can print each column of information in the `Summary Information for Periods` section by using the appropriate option, or you can print all columns by using the `-A` option. The `airsum` command collects the statistics from the product availability state breakdown. The report contains the following columns of information:

Column	Description
`Product Name`	Name of the monitored product. Each configured product is displayed.
`Total Time Available`	(Displayed by using the `-t` option) Entire time that the monitored product was available to its monitoring functions.
`Total Time Unavailable`	(Displayed by using the `-T` option) Entire time that the monitored product was unavailable to its monitoring functions.
`Longest Period Available`	(Displayed by using the `-l` option) Longest period of time that the specific product was in the available state.
`Longest Period Unavailable`	(Displayed by using the `-L` option) Longest period of time that the specific product was in the unavailable state.
`Shortest Period Available`	(Displayed by using the `-s` option) Shortest period of time that the specific product was in the available state.
`Shortest Period Unavailable`	(Displayed by using the `-S` option) Shortest period of time that the specific product was in the unavailable state.

`Average Period Available`

(Displayed by using the `-m` option) Average period of time that the specific product was in the available state.

`Average Period Unavailable`

(Displayed by using the `-M` option) Average period of time that the specific product was in the unavailable state.

The `-a` and `-u` options let you specify the return type tags used in determining the availability of the monitored products. By default, the `airsum` command uses the `PROD_AVAILABLE` and `PROD_UNAVAILABLE` return type tags for the availability determination. For examples of all options, see `airsum`(8).

3.5.2.5 Log File Analysis

This section analyzes an `aird` binary log file to provide you with the basic concepts involved in determining product availability from the data collected by the AIR system.

The following steps lead you through the analysis of an `aird` binary log file.

The most direct way to determine the availability of each monitored product on your system is to create the default report generated from the `airsum` command, as in the following example command line:

```
airsum -h /usr/spool/air/logs/blog
```

The `-h` option provides headers for the printed information and the default report contains the availability numbers for each monitored product on the system, as well as for the `aird` process itself. The following output is produced by the previous command line:

```
*** Total Availability Summary ***

Summary Information

Product          Total Time     Total Time     Rel. Perc. Real Perc.
Name             Available      Unavailable    Available  Available
---------------- -------------- -------------- ---------- ----------
aird                   04:22:43       00:00:00        100         99
tcp e                  00:00:00       04:20:33          0          0
nqs                    00:00:00       04:20:33          0          0
tapes                  04:20:33       00:00:00         99         99
```

```
msgdaemon              04:15:33       00:00:00          97          97
urm                    04:15:33       00:00:00          97          97
disk-integ             04:08:03       00:00:00          94          94
--------------------------------------------------------------------
```

By interpreting this information, you can tell that the example binary log file spans approximately 4 hours (most log files will be much longer than this).

This summary report indicates that the `aird` process was available the entire time the system was up; thus the relative and real percentages are identical for all products. If, for any reason, the `aird` process had not been running during the entire system time contained in the sample, the relative and real percentages would differ.

The availability times vary greatly among the products shown as never unavailable. Products that are always available do not necessarily have the same availability time values for a given sample for the following reasons:

1. Each monitoring function is executed at its individual, and commonly different, rate, as specified within the configuration file.
2. The availability of the product is keyed off the time marks found in the records logged by the `aird` process when each respective function returns from execution.

Thus, a product showing a shorter availability time indicates that the product's monitoring functions are configured to execute at a slower rate than a product showing a longer availability time. The disparity depends on the sample size and the configured execution rate for a function; the disparity increases as sample sets decrease in size and execution rates increase.

The following `airtsum` command line illustrates these concepts:

```
% airtsum -h /usr/spool/air/logs/blog

*** Total Test Summary ***

Function Summary Information

Product          Function         Total    Total Time
Name             Name             Executed Tested
---------------- ---------------- -------- --------------
tcp              gatedexist             14       01:05:00
                 namedexist             14       01:05:00
                 ntpdexist              14       01:05:00
                 existence              14       01:05:00
                 smailexist          14        01:05:00
                 snmpdexist          14        01:05:00
                 lpdexist            14        01:05:00
                 function             7        01:00:00
nqs              existence           14        01:05:00
                 netexist            14        01:05:00
                 response             7        01:00:00
                 function             5        01:00:16
tapes            existence           14        01:05:00
                 avrexist            14        01:05:00
                 response             7        01:00:00
msgdaemon        response             7        01:00:00
                 existence           14        01:05:00
urm              response             7        01:00:00
                 existence           14        01:05:00
                 function             5        01:02:00
disk-integ       existence           14        01:05:00
                 function             5        01:02:00
------------------------------------------------------------------------
```

The output from this command shows how varied testing can be for each product. The frequency of executions indicates how often the function has monitored the product. The kernel response test has been configured to execute at a very high frequency, as indicated by the high number of total executions. Also, because it has the highest frequency of execution, it also has the longest testing time accumulated. Remember that these times converge as the sample length is extended.

The testing times shown in the `airtsum` report and the availability times shown in the `airsum` output differ greatly (approximately 1 hour versus approximately 4 hours). The `airsum` report generator cause this variance by assigning a product's first state to the status returned by the first record pertaining to the product. This first state extends from the system boot time to the end time of the first record. In normal operations, the time between booting the system and logging the first record is negligible. In this example, approximately 1 hour previous to the time this snapshot of the binary log file was taken, the `mv_files` script was executed. Therefore, the binary log file contains information from only the last hour or so. The availabilities calculated by the `airsum` command, however, reflect the fact that the system was booted 4 hours ago. In cases such as this, where the time between the system boot and first record of the sample is no longer insignificant, you should gather all information by specifying the last binary log file, in addition to the current binary log file, on the report generator command lines.

The numbers in the `airtsum` report indicate the execution rate for each function and you verify the information by using the `airprconf` command, which prints the current configuration file (see Section 3.3.4, page 152 for a sample configuration file).

By observing the `airtsum` output and this file, you can see the results of the different execution rates specified in the file. For example, the kernel response function is set to a 1-minute execution rate while most of the other functions are 5 and 10 minutes. Refer to Section 3.3.2, page 143, for more information on using this file.

If you return to the summary report generated by the `airsum` command, you can see that, although NQS and TCP/IP are available, AIR marks them as unavailable. To understand this discrepancy, you could run `airsum` to create the product availability breakdown section, as follows:

```
% airsum -hB /usr/spool/air/logs/blog

*** Total Availability Summary ***

Product Availability Breakdown

Product    Product          From                 Until
Name       Status           Time                 Time
---------- ---------------- -------------------- ---------------------
aird  PROD_AVAILABLE   May  1 08:37:44 1991 May  1 13:00:27 1991
tcp        PROD_UNAVAILABLE May  1 08:37:44 1991 May  1 12:58:17 1991
nqs        PROD_UNAVAILABLE May  1 08:37:44 1991 May  1 12:58:17 1991
tapes      PROD_AVAILABLE   May  1 08:37:44 1991 May  1 12:58:17 1991
msgdaemon  PROD_AVAILABLE   May  1 08:37:44 1991 May  1 12:53:17 1991
urm        PROD_AVAILABLE   May  1 08:37:44 1991 May  1 12:53:17 1991
disk-integ PROD_AVAILABLE   May  1 08:37:44 1991 May  1 12:45:47 1991
```

In this example, the breakdown does not provide any additional information because the sample is too short for much change in states. However, for larger samples, this report allows you to view the availability breakdown for any products you may be examining. For example, if the `airsum` summary report indicates that the `aird` process is unavailable for some period of time, the product availability breakdown report would show the time that the product was available and unavailable.

Because this report has not revealed any additional clues, the next report to examine is that created by the `airtsum` command. In particular, it shows the functional breakdown in terms of return types or the status each function marked the product, as follows:

```
% airtsum -hr /usr/spool/air/logs/blog

*** Total Test Summary ***

Function Summary Information

Product    Function   Total    Return           Number   Total Time
Name       Name       Executed Type             Returned Tested
---------- ---------- -------- ---------------- -------- --------------
tcp        gatedexist       14 PROD_UNAVAILABLE       14       01:05:00
           namedexist       14 PROD_AVAILABLE         14       01:05:00
           ntpdexist        14 PROD_AVAILABLE         14       01:05:00
           existence        14 PROD_AVAILABLE         14       01:05:00
           smailexist       14 PROD_AVAILABLE         14       01:05:00
           snmpdexist       14 PROD_AVAILABLE         14       01:05:00
           lpdexist         14 PROD_AVAILABLE         14       01:05:00
           function          7 PROD_UNAVAILABLE        7       01:00:00
nqs        existence        14 PROD_AVAILABLE         14       01:05:00
           netexist         14 PROD_AVAILABLE         14       01:05:00
           response          7 PROD_AVAILABLE          7       01:00:00
           function          5 PROD_UNAVAILABLE        5       01:00:16
tapes      existence        14 PROD_AVAILABLE         14       01:05:00
           avrexist         14 PROD_AVAILABLE         14       01:05:00
           response          7 PROD_AVAILABLE          7       01:00:00
msgdaemon  response          7 PROD_AVAILABLE          7       01:00:00
           existence        14 PROD_AVAILABLE         14       01:05:00
urm        response          7 PROD_AVAILABLE          7       01:00:00
           existence        14 PROD_AVAILABLE         14       01:05:00
           function          5 PROD_AVAILABLE          5       01:02:00
disk-integ existence        14 PROD_AVAILABLE         14       01:05:00
           function          5 PROD_AVAILABLE          5       01:02:00
-----------------------------------------------------------------------
```

This output shows that only the `gatedexist` function (not all of TCP/IP) was returning a status of unavailable. For NQS, the function test was failing.

If an existence test continually returns an unavailable status, ensure that the particular process the function is verifying is actually configured to exist on the system. In the example, the running system does not have the routing daemon configured. Thus, the gatedexist function should be disabled in the configuration file.

If you do not properly configure monitoring functions for your system, the tests might report incorrect information about the monitored products. Ensure that all configured functions are appropriate for your system so that you may quickly diagnose and correct errors.

Although the cause of the existence functions failures in the example has been identified, the cause of the NQS functional test failure is still unknown. Use the airdet command to produce more details on system activities. Use the following command line to print all records for the NQS function test:

```
% airdet -hm -p nqs -f function /usr/spool/air/logs/blog

Product Function Type of Message  Message
Name    Name                      Text
------- -------- ---------------- ---------------------------------------------
nqs     function PROD_UNAVAILABLE Returned job did not contain expected output
nqs     function PROD_UNAVAILABLE Returned job did not contain expected output
nqs     function PROD_UNAVAILABLE Returned job did not contain expected output
nqs     function PROD_UNAVAILABLE Returned job did not contain expected output
nqs     function PROD_UNAVAILABLE Returned job did not contain expected output
```

Obviously, the batch job submitted to NQS was returned and does not contain the expected output. At this point, you must log into the system and try to determine why the output was not appearing correctly.

Note: If you do not properly configure monitoring functions for your system, the tests might report incorrect information about the monitored products. Ensure that all configured functions are appropriate for your system so that you may quickly diagnose and correct errors.

3.5.2.6 Summary

In the example in the previous section, a short binary log file is analyzed. The airsum command is used to first look at the global availability statistics for each of the monitored products. This section discussed the concepts of real versus relative percentages as they were displayed in the report, and examined the derivation of the net system statistics. We also touched upon the reasons that the availability numbers did not always match other products of similar states, and used the airtsum and airprconf commands to convey the results of the variable execution rates. As an aside, we talked about the assumption made by the airsum command in calculating the total test time starting from the system boot time. We then went on to troubleshoot why various products were marked unavailable, and used the breakdown report generated by `airsum`, as well as the return type breakdown report generated by `airtsum`, in our examination of the failing products. We pointed out the importance of properly configuring the AIR system, and the results of an incorrect configuration. And finally we went on to use the `airdet` command to isolate the truly failing function and to determine the reason for that failure.

Fair-share Scheduler [4]

The fair-share scheduler (also referred to as *fair-share*) controls CPU resources and allows the Cray system to be shared among groups in an organized fashion. To accomplish this, the fair-share scheduler assigns the system's CPU resource to the most deserving processes.

Other system resources, such as usage of memory, tape, SSD, and system calls, are monitored by appropriate system components. However, the fair-share scheduler can be configured to add a penalty for usage of these resources as well as the CPU resource.

This section includes information on the following topics:

4.1 Design Objectives

The fair-share scheduler is designed to allow users with similar *share allocations* (the number of shares allocated in the user database, or *UDB*) to utilize similar amounts of the CPU resource, regardless of the number of active processes they have executing. By contrast, traditional UNIX process schedulers allow users with more processes to have a larger percentage of the system than their priority might typically allow. The fair-share scheduler knows the aggregate consumption rate of each user, and it does not allow users with many active processes to utilize CPU resources at a higher rate than those users with only a few active processes.

Another goal of resource scheduling is to provide adequate and predictable response times. However, it is possible to have levels of system loading that create a long response time, regardless of the scheduling mechanism used. It is your responsibility to control the amount of work in process, using other available methods.

Before the UNICOS 8.0 release, the Network Queuing System (NQS) queue structures were the primary method of controlling the system workload. Beginning with 8.0, the Unified Resource Manager (URM) can be used to control the workload. Using URM ensures that the work the system is expected to do is reasonable, and predictable responsiveness is achievable. For more information on URM, see "Unified Resource Manager (URM)," Chapter 8, page 357.

Under the UNICOS operating system, users are allocated a portion of the CPU resource specified by their share. To improve the fair distribution of resources among users who are allocated equal shares, the concept of *usage history* has been introduced. Usage history allows the scheduler to allocate proportionally more resource to a user who has done less work in the recent past than one who has done more. This rewards users who distribute their work over time and can be valuable in environments where deadlines cause users to do work within a short span of time (which increases the possibility of overloading the system).

The length of time during which past work affects the priority calculation is determined by a *decay factor* (expressed as the half-life of usage history). This controls the rate at which past usage is reduced as a factor in scheduling the user's processes. It can be set anywhere in a range from seconds to many days.

4.2 Fair-share Feature Summary

The fair-share scheduler and the portion of resource control represented by it can be summarized as follows:

- Comprehensive user information is contained in the UDB. A number of utilities and library routines are provided to maintain and view this information and migrate from earlier mechanisms for user validation and control.

- Two system calls, `limits`(2) and `policy`(2), provide an interface between the kernel and user levels of the fair-share scheduler. A daemon, `shrdaemon`(8), updates usage information in the UDB and recovers user information from unplanned system halts. If user-level fair-share mode is enabled, the daemon also updates lnode information in the kernel. The `login`(1) command, the `cron`(8) command, and NQS access the new user information and pass it on to the kernel by using the `setshares`() routine to create *lnodes* (limits nodes) based on the UDB definitions.

- The system tracks usage of users or accounts with a user limits structure called an lnode in kernel memory for each active user name on the system. Users can access their system usage with the `limits`(2) system call.
- Both the user and administrator have displays of scheduling activity available through the `shrview`(1) command (as well as the command `shrmon`(8)). A user's profile can be viewed with the command `udbsee`(1), and the administrator may use features of this command to help generate reports or create source input, which, with further manipulation, can be used by `udbgen`(8) for UDB maintenance.
- At system startup or during operation, the administrator uses the `shradmin`(8) command to set or alter the behavior of the fair-share scheduler to tune the system or prepare for differences in operational emphasis. (The `shrdist`(8) and `shrsync`(8) commands can be used to adjust shares.)
- A fair-share hierarchy can be defined to proportion resources among and within organizations so that a predictable amount of system resources can be allocated to each organization. This method of allocation is dynamic and does not allow a portion of the resource to go unused if some of the organizations are not presently active or are unable to utilize their share. Users and administrators can display the fair-share hierarchy with the `shrtree`(8) utility.
- An optional *Share by Account* mode can be used to assign shares to account IDs rather than to users as in the default *Share by User* mode. With this feature, a user's share allocation and resulting scheduling priority are determined by the user's account ID, which is set initially to the default account ID for the user (in the UDB). Scheduling priorities are determined by the current account ID as users change from one account to another using the `newacct`(1) command.
- Usage history to the degree desired is available the administrator. It allows users or organizations who have equal shares but have utilized unequal amounts of resource to come into balance by encouraging the load to be spread over time rather than in a last-minute flurry of activity before a deadline.
- An optional CPU scheduling mode, *user-level fair-share*, provides a user exit and duplicates kernel functionality at the user level. In this mode, the fair-share scheduler's calculations are performed by the fair-share daemon, `shrdaemon`(8), instead of the kernel. `shrdaemon` replaces the kernel functions that apply the scheduling policy algorithms. This optional mode is enabled with the `USRLEVLFSS` flag in `shradmin`(8).

4.3 Components of Fair-share

Several components, both at the user level and in the kernel, work together to accomplish the objectives of the fair-share scheduler. These are described in the following sections and include the UDB, support functions, user and administrator displays, administrator controls, hierarchical share, share normalizing, and process scheduling.

4.3.1 User Database (UDB)

In resource control, all user profile information must be stored in a comprehensive way. The UDB contains all the user information (factors) used for the scheduler. The UDB factors used for the fair-share scheduler are as follows. (For more information on UDB fields, see the `udbgen`(8) man page.)

Factor	Description
Acids	The UDB stores account IDs (*acids*) as a list of up to 64 (set by `MAXVIDS`) numeric account IDs or account names separated by commas. If acids are used, they must be added to the `/etc/acid` file before `udbgen` is executed. The UDB `acids` field is maintained by the administrator; it has the following format: `acids =|+|- :`*n1*`,` *n2*`, ...,` *nn*`:`
Charges	The UDB stores the long-term accumulated costs from the fair-share scheduler. The UDB `shcharge` field is maintained by the fair-share scheduler; it has the following format: `shcharge :`*vv.vv*`:`
Exit-time	Records the time that the user last completely logged off the system (that is, the last time there were zero processes owned by this user running in the system). The UDB `shextime` field is maintained by the fair-share scheduler; it has the following format: `shextime :`*n*`:`
Resource-group	In a fair-share hierarchy, a user can be a member of an entity known as a *resource group*. Resource groups can be members of other resource groups. Four levels are enabled by default; the maximum

	number of levels is limited only by the configuration, but system overhead increases for each additional level. Resource groups can be further divided by account IDs, or *shareholders*, by using specific values in the UDB share-flags field (see the "Shareholders" entry in this list). Resource groups exist in the UDB with the character "*" as the password, ensuring that no user logs in as a resource group. The UDB `resgrp` field is maintained by the administrator; it has the following format: `resgrp:`*n*`:`
Shareholder	Subdivision of a resource group, used when *Share by Account* mode has been selected; also called an account ID. In Share by Account mode, there must be an entry in the UDB for each account ID that is in use. These entries must correspond with account ID (acid) numbers; that is, they must exist in the `/etc/acid` file before `udbgen` is executed (see the "Acids" entry in this list).
Shares	Each user of the system is allocated a number of shares. This number has meaning only as a proportional value; that is, a share represents the proportion of system resources relative to all other users or accounts within the same group. The actual share values in the UDB are not relevant in any other way. The UDB `shares` field is maintained by the administrator; it has the following format: `shares:`*n*`:`
Share-flags	Certain user entries in the UDB can represent an entity other than a direct user of the system (such as a resource group or a shareholder). The UDB `shflags` field is used to denote these special UDB entries. This field is maintained by the administrator; it has the following format: `shflags =|+|-:`*octal*`:` (The fair-share scheduler now enforces correct usage of the fair-share fields in the UDB.)
Usage	One principal factor that the fair-share scheduler uses for establishing priority is a user's previous usage of the system. When a user is completely

logged off the system, that user's decayed usage of the system is recorded in the UDB. The next time the user logs in to the system, the login process calculates a new decayed-usage value to be installed in the system, based upon the amount of time that has passed since the user last logged out (see the "Exit-time" entry in this list). The UDB `shusage` field is maintained by the fair-share scheduler; it has the following format: `shusage:`*vv.vv*`:`

In addition to the fair-share fields in the UDB, several special user accounts (UDB entries) are necessary for the proper operation of the system, including the `Idle` account, the `UnKnown` or `unknown` account, and resource group entries. (See "Setting up system UDB entries," Section 4.4.4, page 211, for a list of these special UDB entries.)

Idle processes are treated just like any other user of system resources in the fair-share system and, as such, need an entry in the UDB. User ID 11 is reserved in UNICOS for representing the idle usage of the system.

There must also be an entry in the UDB called `UnKnown` or `unknown`; this entry is used when no valid UDB entry for a user can be found.

Each resource group or shareholder (account ID) represented within the fair-share hierarchy must also have an entry in the UDB. Resource groups and shareholders are assigned shares, which are taken to be relative to the shares allocated to other resource groups at the same level, and are a subset of the shares in the next higher level in the fair-share hierarchy.

4.3.2 Support Functions

At the user level, the fair-share daemon `shrdaemon`(8) performs the following activities:

- At 1-minute intervals, `shrdaemon` records in the UDB usage for each user and resource group that has finished execution.
- At 5-minute intervals, `shrdaemon` checkpoints the kernel tables holding fair-share usage information about running users and resource groups.
- When the UNICOS operating system is restarted after an unscheduled shutdown, `shrdaemon` recovers user and group CPU consumption information from the checkpoint file.

At the kernel level, the fair-share scheduler performs the following activities:

- At configurable intervals (4 seconds by default), the fair-share scheduler accumulates charges and updates usage information in the lnodes.

  Note: In user-level fair-share mode, updating usage and lnode information is done by the fair-share daemon rather than by the kernel.

- Every 1/60th of a second, CPU usage is accumulated and stored in the lnodes of processes having had the CPU, and the `p_sharepri` values are adjusted based on usage.

- At 1-second intervals, the share priority of each process is decayed according to the rate determined by the nice value of the process.

The most significant factors affecting placement on the kernel run queue (high or low) are as follows:

- The process's resource usage during the last cycle, relative to other processes.

- The process's proportion of machine shares relative to other users or groups (active lnodes) at that time.

The scheduling calculations can lower, raise, or leave unchanged the position of the process in the run queue. When this evaluation has been accomplished, the processes at the top of the run queue are connected to a CPU.

In addition, the positions of all the processes in the queue are evaluated by decaying their resource consumption at a rate determined by their nice values (processes with smaller nice values move toward the top of the queue faster). This has the effect of gradually moving processes to the top of the queue and ensures that every process will be scheduled to run; note that only processes which have not received the CPU recently actually move up in the queue. It is necessary to balance the rate of migration to the top of the queue and the rate of resource consumption so that relative priorities are remembered for a long enough time period to prevent large numbers of processes from migrating to the top of the queue.

4.3.3 User and Administrator Displays

Fair-share displays are generated by the `shrview`(1) command, the `shrmon`(8) command, and the `shrtree`(8) command. In addition, users can view their share control values by using the `udbsee`(1) command, which displays the content of the UDB (with the exception of sensitive information).

Note: The `shrmon`(8) command will not be available in future releases of the UNICOS operating system. Its functionality has been replaced by the `shrview`(1) command.

4.3.4 Administrator Controls

The `shradmin`(8) command changes scheduling parameters, decay rates, flags, and other useful items. One of the intended uses of `shradmin` is to set appropriate control parameters at startup so that you can alter the behavior of the system to reflect current needs without resorting to recompiling modules or other inefficient mechanisms. It is also possible to change scheduling characteristics during system operation. This facility could be used to change scheduling emphasis during portions of the day when, for example, mostly interactive or batch work is encouraged.

Note: You must run `shradmin` before activating the fair-share daemon, `shrdaemon`(8). For more information, see "Activating the fair-share scheduler," Section 4.4.6, page 214.

You can use the `-n` option of `shradmin`(8) to lend more reliability to the fair-share scheduler information in the UDB. This option specifies the interval, in seconds, to be used for copying lnode information to the UDB. When this feature is enabled, the fair-share daemon writes accumulated usage and charge information from the lnodes to the UDB at the specified interval. For more information, see the `shradmin`(8) man page.

The `shrview`(8) command monitors the operation of the fair-share scheduler at a closer level of detail.

To turn off fair-share scheduling, see "Disabling the fair-share scheduler," Section 4.4.9, page 219.

4.3.5 Fair-share Hierarchy

A flat, or single-level, fair-share mechanism is inadequate for many environments because it does not provide a convenient way to allocate shares among organizations and then to the users within each organization. The *fair-share hierarchy* grants each organization a part of the system, determined by the proportional shares assigned to each. After this is accomplished, the members of each organization can be allocated shares as though that organization had exclusive use of the system. This makes the allocation of shares within an organization easier; however, individual user share values are not comparable across organization boundaries.

Whenever the active user population changes, the shares assigned through the fair-share hierarchy are converted to an internal value known as *machine share*. This is the proportion of the available resources to which a group or user is entitled. (You can see these values in the fair-share displays.) Because they are normalized across organizations with respect to each organization's share proportion, machine-share relative values can be directly compared; user shares cannot. Machine shares are recomputed whenever a user logs in or out or when an organization becomes active or inactive. For more information, see "Share normalizing," Section 4.3.6, page 201.

You can activate the optional Share by Account mode by using `shradmin`(8) to set the `SHAREBYACCT` scheduling flag. In this mode, fair-share determines the relative share based on the account ID rather than the user ID (UID). The initial association is based on the default account ID, which is the first ID in the list of valid account IDs for that user in the UDB entry. In Share by Account mode, the `newacct`(1) command reassociates the user with the new account ID, which allows the user to move within the fair-share hierarchy.

If Share by Account mode is enabled, the fair-share hierarchy must have resource-group lnodes at the head of the hierarchy chain; the hierarchy chain must terminate with shareholder (account ID) lnodes.

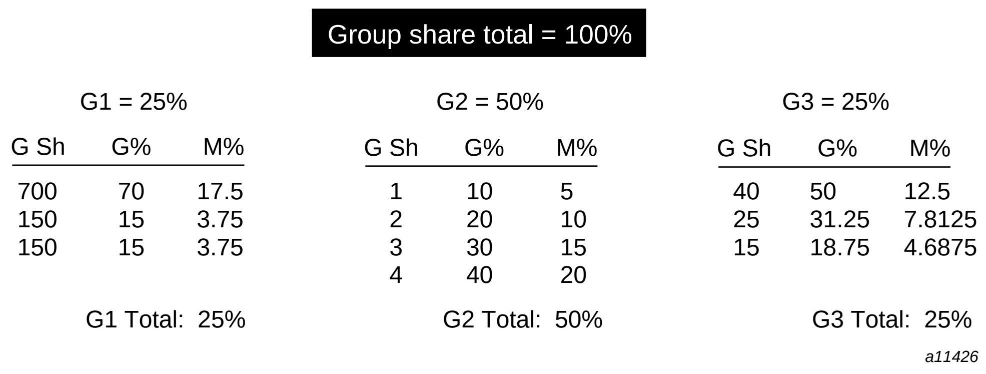
You can use the `shrtree`(8) command to display and verify the fair-share hierarchy; see "Using the `shrtree`(8) command," Section 4.4.11.3, page 224, for more information.

4.3.6 Share Normalizing

In Figure 4, page 202, three groups have been given portions of the system resources. Groups G1 and G3 have 25%, while Group G2 has 50%. This illustration is a snapshot of a particular instant in time when the active group membership is as shown. Within each group, shares have been allocated based on some arbitrary range of values. G1 is based on the range 0 through 1000, G2 on the range 0 through 10, and G3 on the range 0 through 100, to show that the normalizing function works in mixed-range situations.

The columns headed "G Sh" show the shares allocated to the currently active users from each group. Column "G %" shows the percentage of the group's resource to which each user is entitled, based on the allocated shares and the active group membership. (The percentage will vary as group members arrive and depart because the goal is to distribute the system proportion fairly according to each user's share among the active users at a given instant.)

The column headed "M %" shows the actual machine share each user should have, based on the group in which the user exists. The totals at the bottom of the figure show that the sum of machine shares from each group adds up to the share allocated to the group.



Group share total = 100%

G1 = 25%

G Sh	G%	M%
700	70	17.5
150	15	3.75
150	15	3.75

G1 Total: 25%

G2 = 50%

G Sh	G%	M%
1	10	5
2	20	10
3	30	15
4	40	20

G2 Total: 50%

G3 = 25%

G Sh	G%	M%
40	50	12.5
25	31.25	7.8125
15	18.75	4.6875

G3 Total: 25%

Figure 4. Share Normalizing

Normalizing is accomplished as follows:

1. Calculate the number of shares active, SA_g, within each group g for each member m:

$$SA_g \sum_{m=1}^{m=max} share_m \tag{4.1}$$

2. Calculate machine share, MS_m, for each member m of group g:

   $MS_m = Group\_proportion_g \times (share_m \div SA_g)$

This example considers only one hierarchy level. When more than one level is being supported on a given system, the method shown above is recursively applied down the hierarchy tree until the MS_m of each group and user node has been calculated. As more levels are added, however, there is a corresponding increase in the system overhead required to perform all the calculations for each cycle. See "Tuning the fair-share scheduler," Section 4.6, page 229, for more information on this overhead.

4.3.7 Process Scheduling

The purpose of process scheduling is to ensure that all system processes are running and to assign the remaining CPU resource to user processes. As discussed previously, this means that the CPU is assigned to the process at the top of the run queue. As resources are utilized, processes move down the queue, and as processes age, they move up the queue. This change in queue position is also influenced by the user's share of the system, the interaction between shares, resource consumption, and the passage of time. This process scheduling is the essence of the fair-share scheduling process.

4.3.8 Fair-share Limits Node (Lnode)

When a user enters the system (through NQS, `cron`, `rsh`, or interactive access) or exits the system, the system tracks that user's usage as follows:

1. The system creates a user limits structure called an *lnode* (limits node) in kernel memory. In Share by User mode, there is an lnode for each active user on the system; in Share by Account mode, there is an lnode for each active account ID (shareholder). System usage from the relevant UDB entry is passed to the kernel by the `setshares` () routine, which calls the `limits`(2) system call.
2. The time elapsed since the user last exited the system is determined, and the user's or shareholder's usage is aged by this last exit time. The fair-share scheduler uses this calculated decayed usage in the adjustment of process priorities. The decay rates can be configured by the system administrator; see `shradmin`(8) for more detail.
3. If necessary, the system creates lnode entries representing the ancestors (controlling levels in the fair-share hierarchy) of the user's lnode.
4. Once the lnode chain has been created and the user's login shell has been spawned (for both interactive and batch usage), all subsequent processes of the session reference (*are attached to*) the terminal lnode. However, in Share by Account mode the `newacct`(1) command can be used to attach a process to a different lnode.
5. The fair-share daemon, `shrdaemon`(8), updates the user database every 60 seconds (by default). Information from any lnode structures that have no attached process table entries is recorded; these lnodes are subsequently released. When the lnode structure is removed (if a user or group has no running processes), `shrdaemon` records the exit time in the `shextime` field of the user's UDB entry, and records the usage in the `shusage` field.

6. To prevent loss of current usage information stored in the lnode (for example, because of a system interrupt), `shrdaemon` writes all the active lnode structures to the checkpoint file `/etc/lnodes.chkpt` every 5 minutes.

7. In user-level fair-share mode, the scheduling policy algorithms are applied to lnode data by the fair-share daemon rather than by the kernel.

4.3.9 Fair-share and NQS

Fair-share data can also be used by the Network Queuing System (NQS) to select batch jobs for execution. This makes it possible for the fair-share scheduler to influence both queued and running jobs. The amount of influence fair-share has on NQS job scheduling is established by NQS scheduling parameters.

Sites running fair-share NQS should read the following section on fair-share setup and administration ("Using the fair-share scheduler," Section 4.4, page 204). For information on start-up and administration of fair-share NQS, see the `qmgr`(8) man page and the UNICOS NQS and NQE Administrator’s Guide, publication SG-2305.

4.3.10 Fair-share and URM

On a system using the fair-share scheduler, the Unified Resource Manager (URM) uses share values in its priority calculations for NQS job initiation recommendations. During the batch job ranking phase, if the fair-share weighting factors for machine share, usage, and share entitlement are nonzero values, URM computes a fair-share priority value based on effective share and past usage. This priority is combined with other scheduler weighting factors to determine the job selection order.

For more information on URM setup and administration, see "Unified Resource Manager (URM)," Chapter 8, page 357.

4.4 Using the Fair-share Scheduler

The following sections describe how to set up, activate, and monitor the fair-share scheduler, including the following tasks:

- Setting up a fair-share hierarchy
- Creating resource groups

- Allocating shares to users
- Setting up Share by Account mode
- Setting up system UDB entries
- Activating the fair-share scheduler
- Modifying fair-share scheduler settings
- Enabling resource group administrators
- Disabling the fair-share scheduler
- Monitoring the fair-share scheduler

The fair-share scheduler has two modes of operation: Share by User and Share by Account. In the default Share by User mode, the fair-share scheduler calculates priorities and costs for each active user. In the optional Share by Account mode, fair-share calculates priorities and costs for each active account ID. If a user works on different projects, Share by Account mode allows that user to switch between projects, which are set up as accounts, by using the `newacct`(1) command. This allows the user to work under a different set of fair-share priorities for each project.

The steps to set up and activate Share by Account mode are similar to those for Share by User mode, but there are some significant differences. The hierarchy of resource groups can be set up the same way. In addition, you must create UDB entries for each valid shareholder, or account ID, and assign share allocations to them. The following sections describe the procedures for setting up Share by User mode. See "Setting up Share by Account mode," Section 4.4.5, page 213, for specific information on setting up this optional share allocation method.

4.4.1 Setting up a Fair-share Hierarchy

This section describes how to establish a fair-share hierarchy. An example hierarchy is used throughout the section for demonstration purposes; note that actual division of resources on Cray systems will differ for each installation.

Although using a multilevel fair-share hierarchy is optional, the process of share allocation is simplified by organizing users in resource groups. In Share by User mode, the fair-share hierarchy has users at the end of the resource group chain; in Share by Account mode, the hierarchy has projects or groups, also known as *account IDs*, at the end of the chain.

Resource group chains are created for important divisions of users at the site (for example, different divisions, projects, or physical locations). As an example, site ACME divides UNICOS resources into three categories for three different projects: Projects 1, 2, and 3. Each of these projects must be assigned the appropriate proportion (share) of the system. In addition, system maintenance functions must receive enough resources to accomplish their tasks. The following figure shows the desired division of resources:

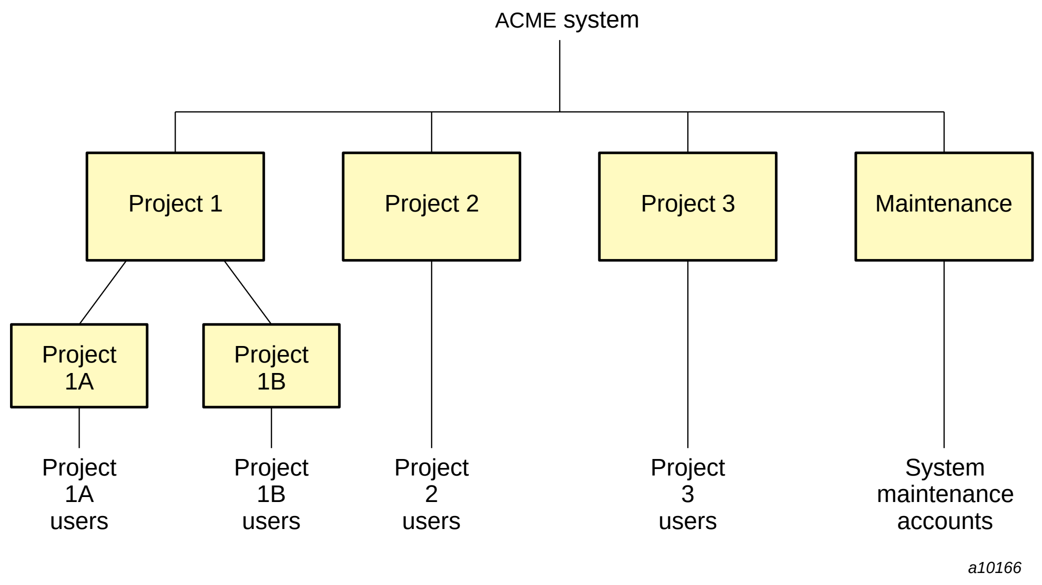


Figure 5. Example of System Resource Division for Fair-share

Four levels of hierarchy are available by default (the number of levels may be changed by using `shradmin -G`). The first level is permanently assigned to `root`. Additional levels of resource groups can be defined by creating a resource group that references its parent resource group in the UDB.

Note: Users or shareholders must not be connected directly to the root lnode. There must be at least one level of resource groups above the user or shareholder level. When this rule is not followed, the `shrtree`(8) command issues a warning message about referencing `ROOT` directly.

When designing a fair-share hierarchy, keep it as simple as possible. You can make later refinements to the hierarchy as needed without having to change other fair-share parameters. For information on how to make UDB or hierarchy changes on a running system, see "Modifying fair-share scheduler settings," Section 4.4.7, page 217. For information on how to display the fair-share hierarchy, see "Using the `shrtree`(8) command," Section 4.4.11.3, page 224.

4.4.2 Creating Resource Groups

The first step in creating a usable fair-share hierarchy is to create UDB entries for the resource groups. All resource groups must have an entry in the UDB. (Use the `udbgen`(8) command to create these entries.) The following guidelines apply to resource groups:

- Each resource group must have a unique user ID (UID). It is helpful to use a unique range of UIDs so the resource groups are easy to distinguish. To prevent accidental logins for the resource group's UID, do not specify a password (the default encrypted password is the character "*").
- Each resource group must be assigned shares relative to the shares allocated to other resource groups at the same level of the hierarchy. A common technique would be to apportion the shares among resource groups at the same level such that the total number of shares adds up to 100 or 1000 (for example, 50-50 or 700-200-100); however, any numbers can be used.
- Resource groups must have their `shflags` field in the UDB initialized to 040000. (The fair-share scheduler now enforces correct usage of the fair-share fields in the UDB.)
- For each resource group, the `resgrp` field indicates the UID of the controlling resource group (that is, the resource group above it in the hierarchy chain). Resource groups can belong to other resource groups, as explained on Section 4.3.1, page 196. Top-level resource groups (chain leaders) should have this field set to the `root` UID, which is always 0.

Because resource groups are defined in the UDB with the ordinary user entries and shareholders (account IDs), it is recommended that you establish a naming scheme to help distinguish resource group entries from user and shareholder entries and to avoid use of identical names. One convention that works well is to capitalize the first character of the names of the resource groups.

Note: A resource group entry for a system group is required for the system UDB entries such as `root`, `cron`, and system daemons. This group is often called `system` or `admin` (group `Maint` is used at example site ACME).

For the example site, the following project names are used as resource group chain leaders: `Proj1`, `Proj2`, and `Proj3`. Table 30 contains the project names, their desired proportion of the system, and their equivalent share value. A separate share of the system has also been reserved for maintenance (Maint). A total share value of 1000 is distributed among the projects in the desired proportion.

Table 30. Share Division Among Resource Groups

Project	Proportion	Share value
Maint	10%	100
Proj1	50%	500
Proj2	25%	250
Proj3	15%	150
Total	100%	1000

The UDB entries for these resource group chain leaders can be created by using udbgen(8) with the following directives:

```
create:Maint:uid:999:gid:10:passwd:*:shflags:040000:shares:100:resgrp:0:
create:Proj1:uid:111:gid:20:passwd:*:shflags:040000:shares:500:resgrp:0:
create:Proj2:uid:222:gid:30:passwd:*:shflags:040000:shares:250:resgrp:0:
create:Proj3:uid:333:gid:40:passwd:*:shflags:040000:shares:150:resgrp:0:
```

A comment field can be used to include an explanation for the UDB entries, but that was not done in this example. The value used in the gid field is a site-dependent group ID (GID); at least one GID must be specified, or warning messages will occur. The passwd field has been set to the character "*" to ensure that no one is able to log in to a resource group account. Notice the shflags value of 040000. This value marks these entries as resource groups. Also note that the shares field contains the share value numbers from Table 30.

Additional levels of resource groups can be defined by creating a resource group, as in the first example, including the name or user ID of the parent resource group in the resgrp field. (Four levels are available by default; use the shradmin -G command to increase this amount as desired.) For example, site ACME would subdivide resource group Proj1 into two resource groups, Proj1A and Proj1B, using the following udbgen directives:

```
create:Proj1A:uid:444:gid:20:passwd:*:shflags:040000:shares:500:resgrp:111:
create:Proj1B:uid:555:gid:20:passwd:*:shflags:040000:shares:500:resgrp:111:
```

Each new resource group now has 500 shares, or, half the resources of `Proj1` (1000 was used as the total share value for this level). The `resgrp` field is set to the UID of `Proj1` (111) to mark these resource groups as shareholders of `Proj1`.

4.4.3 Allocating Shares to Users

The next step in setting up fair-share allocations is to decide which users belong to each resource group and assign them to the appropriate group with an appropriate share value. Within each resource group, the share values are relative to each other. A convenient total share value, such as 100 or 1000, will make the job of assigning proportional shares easier. However, any number can be used for the total share value; only the relative values are important.

The allocation of shares in the UDB should occur as follows:

1. If Share by User mode will be enabled, set the resource group of each user (the `resgrp` field) to the name of that user's controlling resource group.
2. If Share by Account mode will be enabled, ensure that each user has a list of valid account IDs in the acids list (the `acids` field) of the UDB; at least one entry is required in this list. Fair-share uses the first account ID in the list as the user's default, or initial, account. (See "Setting up Share by Account mode," Section 4.4.5, page 213, for more information.)
3. Allocate each user and shareholder entry in the UDB a number of shares (in the `shares` field). The exact number is not critical, but it is convenient to pick a value that could later be adjusted up or down. (For example, each user could be allocated 100 shares.)

You can use the `udbgen` command to analyze share resource assignments in the UDB and report any problems. For Share by User mode, use the following command to analyze the default UDB in the `/etc` directory:

```
udbgen -a -R
```

For Share by Account mode, use the following command to analyze share resource groups based on the `acids` field instead of the `resgrp` field:

```
udbgen -a -A
```

Table 31 shows the division of shares for the example site.

Table 31. Share Division within Resource Groups

Group	User	Proportion	Share value
Maint	u1	25%	25
	u2	25%	25
	u3	25%	25
	u4	25%	25
Proj1			
Proj1A	u5	25%	50
	u6	25%	50
Proj1B	u7	25%	50
	u8	25%	50
Proj2	u9	50%	50
	u10	30%	30
	u11	10%	10
	u12	10%	10
Proj3	u13	25%	25
	u14	25%	25
	u15	25%	25
	u16	25%	25

These entries show, for each resource group, the users in the group ("User" column), the percentage of resources each user is to have within the group ("Proportion" column), and the individual share value equivalent to the proportion ("Share value" column). For each group, a total share value of 100 was used.

The following `udbgen` directives update the UDB to reflect this division of resources (only Share by User mode will be enabled):

```
update:u1:resgrp:999:shares:25:
update:u2:resgrp:999:shares:25:
update:u3:resgrp:999:shares:25:
update:u4:resgrp:999:shares:25:
update:u5:resgrp:444:shares:50:
```

```
update:u6:resgrp:444:shares:50:
update:u7:resgrp:555:shares:50:
update:u8:resgrp:555:shares:50:
update:u9:resgrp:222:shares:50:
update:u10:resgrp:222:shares:30:
update:u11:resgrp:222:shares:10:
update:u12:resgrp:222:shares:10:
update:u13:resgrp:333:shares:25:
update:u14:resgrp:333:shares:25:
update:u15:resgrp:333:shares:25:
update:u16:resgrp:333:shares:25:
```

The `resgrp` field for each entry is set to the UID of the resource group (`Maint` is `999`, `Proj1A` is `444`, `Proj1B` is `555`, `Proj2` is `222`, and `Proj3` is `333`). The specified resource group and user entries must already exist in the UDB before these `update` directives can be executed.

4.4.4 Setting up System UDB Entries

System UDB entries such as `Idle`, `root`, and system daemons have special requirements for the resource group and shares fields. This section describes the following accounts:

- `Idle` account
- `UnKnown` or `unknown` account
- Other system accounts (`root`, `cron`, system daemons, and so on)

4.4.4.1 Idle Account

A special account, called `Idle`, is required for the fair-share scheduler to work correctly. (During initial installation of the UNICOS operating system, the UDB initialization process sets up a correct `Idle` entry by default.) The following guidelines exist for the `Idle` account:

- The UID for the `Idle` account (the `uid` field in the UDB) must be 11.
- The `Idle` account must be assigned no shares (the `shares` field must be omitted or set to 0).
- The controlling resource group (the `resgrp` field) must be 0, which is the UID of `root`.
- The `acids` field must be set to a null value (that is, `acids::`).

- The `shflags` field must be 0. (The fair-share scheduler now enforces correct usage of the fair-share fields in the UDB.)

The following example shows a sample `Idle` account for site ACME:

```
Idle:uid:11:comment:System Idle:passwd:*:gids:0:acids::resgrp:0:shflags:0:
```

Note: It is critical to the proper operation of the system that only the `Idle` entry be allocated zero shares in the UDB.

4.4.4.2 UnKnown or Unknown Account

The UDB must contain an entry called `UnKnown` or `unknown`; this entry is used when no valid UDB entry for a user can be found. You must add this entry to the UDB before enabling the fair-share scheduler.

The following example shows a sample `unknown` account for site ACME:

```
unknown:uid:12:passwd:*:gids:0:acids::resgrp:999:shflags:0:shares:1:
```

4.4.4.3 Other System Resource Accounts

Each system account requires a resource group entry in the UDB. This includes entries for `root`, `cron`, the operator, NQS, system daemons, and other system user IDs. These system UDB entries must be assigned to the system or administrator resource group (often called `system` or `admin`; group `Maint` is used at example site ACME). In addition, if Share by Account mode will be enabled, the `acids` field of each system entry should be set to the account ID of the system or administrator shareholder.

For initial installations of the UNICOS operating system the UDB skeleton file creates a default list of resource group and user accounts in the UDB, and sets the `resgrp` field to 0 (`root`).

The UDB skeleton file is located in the file `/usr/src/skl/c1/etc/initudb`. It sets up the following accounts: `root`, `sync`, `bin`, `sys`, `adm`, `cron`, `nqs`, `daemon`, `operator`, `ce`, `Idle`, `unknown`, `osi`, and `nobody`.

Note: Upgrading sites should verify their UDB system account entries against the UDB skeleton file.

To enable a fair-share hierarchy, you must add the UDB entry for a system or administrator resource group and set the `resgrp` field of the system accounts to the UID of this resource group. The `resgrp` field is used to link the resource tree of the hierarchy.

If Share by Account mode will be enabled, the `acids` field of the `root` UDB entry should be set to the UID of the system maintenance or administration resource group (for example, 999 at site ACME specifies the `Maint` resource group).

4.4.5 Setting up Share by Account Mode

By default, the fair-share scheduler runs in Share by User mode, in which a user's relative priority is determined by the `resgrp` field in the UDB. However, you can activate the optional Share by Account mode by using the `shradmin`(8) command to set the `SHAREBYACCT` scheduling flag. In this mode, fair-share associates and sets priorities based on the shareholder, or account ID (`acids` field in the UDB). The initial association is based on the default account ID. This is the first ID in the list of valid account IDs for a user in the UDB entry.

Administrators have the option of assigning shares to users (Share by User mode) or accounts (Share by Account mode). If a user works on different projects, Share by Account mode allows that user to switch between projects, which are set up as accounts, by using the `newacct`(1) command. This allows the user to work under a different set of fair-share priorities for each project.

Enabling Share by Account mode is similar to enabling Share by User mode. The hierarchy of resource groups is set up the same way. However, the following additional administration tasks are necessary to set up Share by Account mode:

- Create a UDB entry for each valid account. It is helpful to use a unique range of UIDs so the account IDs are easy to distinguish. The UIDs for account ID entries (shareholders) must not overlap with resource group or user entries in the UDB.
- Set the `shflags` field for each account ID entry to 01000000. (The fair-share scheduler now enforces correct usage of the fair-share fields in the UDB.)
- Assign share allocations to each valid account ID entry.
- Assign a passwd of "*" to each account ID entry so no user can log in as a resource group.
- Verify that each account ID entry has been added to the `/etc/acid` file before `udbgen`(8) is executed.
- Set the `resgrp` field of each account ID entry in the UDB to the appropriate resource group entry. The `resgrp` field is used to link the resource tree of the hierarchy.

- Specify the first account ID in the acids field of each user entry as the name of the user's default account ID. Each user must have at least one account ID in this field; enter a comma-separated list of account IDs for users who can change to different accounts.
- Verify that the Idle UDB entry specifies a null account ID field; that is, the acids field should be omitted or set to acids::.
- Set the acids field for the root UDB entry to the UID of the system maintenance or administration resource group (for example, 999 at site ACME specifies the Maint resource group).
- If there are more than four levels in the fair-share hierarchy, including account ID entries, use the shradmin -G command to increase the number of hierarchy levels.

4.4.6 Activating the Fair-share Scheduler

To activate the fair-share scheduler during system startup, the system configuration script must be modified to execute the shradmin(8) command with the appropriate options and start up the fair-share daemon, shrdaemon(8). (To change and reactivate fair-share on a running system, see "Modifying fair-share scheduler settings," Section 4.4.7, page 217.)

There are two methods to modify the system configuration script: using the UNICOS Installation and Configuration Menu System, or editing the file /etc/config/daemons.

If you are using the menu system, access the following menu to make this change:

```
Configure System ->
        System daemons configuration ->
                System daemons table
```

If you are not using the menu system, access the file /etc/config/daemons to make this change. An example of /etc/config/daemons is as follows:

```
# /etc/config/daemons excerpt
# group tag        start   kill      pathname      arguments
  SYS1  share      YES     *        /etc/shradmin   options
  SYS1  share      YES     *        /etc/shrdaemon
```

These lines control initiation of `/etc/shrdaemon` (the fair-share daemon) and the administrator command `/etc/shradmin`, which activates the fair-share scheduler using the options described in the following paragraphs.

Note: The `/etc/shradmin` command must be run before the `/etc/shrdaemon` command.

4.4.6.1 Setting Scheduling Options and Flags

The following `shradmin` options are important for enabling fair-share functions. (See the `shradmin`(8) man page for a complete description of all options; see "Tuning the fair-share scheduler," Section 4.6, page 229, for information on performance impact of `shradmin` values.)

Option	Description
`-F`	(*flags*) Sets the fair-share control flags in the kernel `sh_consts` structure. The following flags are available:

`NOSHARE`	(001) Turns off the fair-share scheduler. Leaves accumulated charges in the UDB unless they are cleared by the system administrator.
`ADJGROUPS`	(002) Specifies adjustments by group IDs (group share allocation).
`LIMSHARE`	(004) Specifies limits on maximum share.
`SHAREBYACCT`	(010) Specifies Share by Account mode.
`NOSCHED`	(020) Gathers fair-share charges and usage information, but does not use these values for CPU scheduling.
`USRLEVLFSS`	(0100) Specifies *user-level fair-share mode*; in this mode, the fair-share daemon (`shrdaemon`(8)) performs the share calculations and updates the lnodes. This will be the default mode in future releases of the UNICOS operating system.

`-K`	(*half-life*) Establishes the length of time that past usage of resources are remembered. Longer decay rates cause the

	scheduler to distribute resources fairly over a longer period of time. This option can have a significant impact on interactive responsiveness; see "Tuning the fair-share scheduler," Section 4.6, page 229, for more information. **Note:** Never set the `-K` option to a value lower than 1 hour (3600 seconds).
`-R`	(*delta*) Sets the major fair-share adjustment cycle to the specified value. This means that once every cycle, the resource usage of all users and groups active on the system is evaluated, and, if the `ADJGROUPS` flag is set, the group hierarchy is evaluated to determine what adjustments are necessary to achieve proper group sharing.
`-t`	(*tick*) Sets the cost per tick. The usage field of the user owning that process is increased by this amount.
`-X`	(*maxushare*) Limits how much past usage can be accrued; it is used only when the `LIMSHARE` flag is set. See "Tuning the fair-share scheduler," Section 4.6, page 229, for more information.
`-Y`	(*mingshare*) Specifies the deviation between group share allocation and the actual rate of resource consumption allowed before the scheduling algorithm tries to compensate. It has an effect only when the `ADJGROUPS` flag is set. See "Tuning the fair-share scheduler," Section 4.6, page 229, for more information.
`-Z`	(*sharemin*) Sets a minimum allocation of machine shares for any active lnode (user or account ID). See "Tuning the fair-share scheduler," Section 4.6, page 229, for more information.

An example of ACME's `/etc/config/daemons` is as follows:

```
# /etc/config/daemons excerpt
# group tag    start  kill  pathname        arguments
  SYS1  share  YES    *     /etc/shradmin   -t100 -F06 -K3600s -X3.0 -Y.85 -Z .002
  SYS1  share  YES    *     /etc/shrdaemon
```

The `-t` option sets the cost per tick to 100. The `-F` option enables Share by User mode by setting the fair-share control flags; that is, the `ADJGROUPS` and `LIMSHARE` flags are set, while the `NOSHARE` flag is cleared. The `-K` option sets the usage decay rate to a half-life of 3600 seconds. The `-X` option sets *maxushare* to 3.0, which decreases the possibility that users with very low past usage can monopolize the CPU. The `-Y` option sets *mingshare* to .85 to decrease the deviation between group share allocation and the actual rate of resource consumption. The `-Z` option sets *sharemin* to .002, which guarantees each user or shareholder a minimum machine share of 0.2%.

4.4.7 Modifying Fair-share Scheduler Settings

Elements of fair-share configuration that can be changed on a running system include the following:

- Maximum levels in the fair-share hierarchy
- Distribution of shares within resource groups and account ID shareholders
- Existing UDB entries, such as the `acids`, `resgrp`, or `shares` fields
- Addition or deletion of UDB entries
- All `shradmin`(8) options (tuning parameters)

Use the `shradmin -G` command to change the maximum number of levels for the fair-share hierarchy (there are four levels by default). See the `shradmin`(8) man page for more information.

For information on allowing other users to administer share distribution within a resource group, see "Enabling resource group administrators," Section 4.4.8, page 218.

Use the `udbgen`(8) command to analyze share resource assignments in the UDB and report any problems. For Share by User mode, use the following command to analyze the default UDB in the `/etc` directory:

```
udbgen -a -R
```

For Share by Account mode, use the following command to analyze share resource groups based on the `acids` field instead of the `resgrp` field:

```
udbgen -a -A
```

See the `udbgen`(8) man page for more information.

To make and enable share allocation changes on a running system, use the `shrdist`(8) command to make changes. Users do not have to log out in order for these changes to take effect. (To make substantial changes, you can also use the `udbsee`(1), `udbgen`(8), and `shrdist`(8) commands; see the man pages for more information.)

Note: In order to change from Share by User mode to Share by Account mode (or vice versa), it is recommended that the system be brought to single-user mode, then restarted with the appropriate `shradmin` options. Changing between fair-share modes on a running system can cause unpredictable results for priority calculations.

If your site will alternate between fair-share modes, ensure that each user entry in the UDB sets both the `resgrp` field and the `acids` field to the appropriate values. See "Setting up Share by Account mode," Section 4.4.5, page 213, for more information on setting these fields.

4.4.8 Enabling Resource Group Administrators

System administrators may find it useful to set up separate administrators for each resource group. This capability allows the owner of a resource group (for example, the project manager) to modify share allocations within the resource group without requiring system administrator intervention to do so. For example, if project A changes priority over time, or project B needs more shares towards the end of the month, a resource group administrator could make the necessary changes without system administrator intervention. This is done by setting up the `shrdist.auth` file, which enables the listed resource group administrators to reallocate shares within the group.

To redistribute shares within a resource group, use the `shrdist`(8) command. For this command to work properly, you must create an authorization file called `/etc/shrdist.auth`. The `shrdist.auth` file consists of two fields: the login name of the resource group owner and the name of the resource group or groups.

The following lines from a `shrdist.auth` file allow user `mlb` to administer shares for two resource groups, `Mktg` and `tng`. If a resource group is not specified (with the `-g` option), the first match (in this case, `Mktg`) is used.

```
mlb     Mktg
mlb     tng
```

The `shrdist`(8) command can be used to adjust shares. The `shrdist` command operates in both batch and interactive mode. To reallocate share values in interactive mode, perform these steps:

1. Use the following command to access the share allocation database:

   ```
   /etc/shrdist
   ```

2. Position the cursor at the rightmost digit of the share allocation field for the desired account and enter the new values.

3. Enter `u` to update the share allocation database.

4. Enter `q` to exit the `shrdist` command.

You can also perform updates in batch mode by using the `shrdist -b` command. However, this method only allows you to update one account for every execution of the `shrdist` command. Test mode is also supported (`shrdist -p`). This capability is similar to test mode for the `udbgen`(8) and `udbsee`(1) commands.

4.4.9 Disabling the Fair-share Scheduler

To turn off the fair-share scheduler after it has been enabled, perform the following steps:

1. Execute the following `shradmin` command:

   ```
   shradmin -F 021
   ```

   This sets the `NOSHARE` flag, which specifies that no fair-share scheduling is done. It also sets the `NOSCHED` flag, which prevents the marooning of running processes by ignoring the `shcharge` and `shusage` information in the CPU scheduling algorithm.

2. Wait for at least 1 minute. This delay allows `shrdaemon` enough time to update the fair-share usage and charge information in the UDB.

3. Find the process ID (PID) of the fair-share daemon, `shrdaemon`, as follows:

   ```
   ps -efl | grep shrdaemo
   ```

   (The `ps` command truncates command names at 8 characters.)

4. Disable updating of fair-share information in the UDB by stopping the `shrdaemon` process with the `SIGTERM` signal, as follows:

   `kill -27` *PID_of_shrdaemon*

Turning off the fair-share scheduler in this manner leaves all fair-share usages and charges in the UDB. If a clean UDB is desired (for example, at the beginning of a new fiscal year), perform the following steps:

1. Use the `udbsee`(1) command to create an ASCII version of the UDB fair-share usage and charge fields, `shusage` and `shcharge`, and save this output to a file, as follows:

   ```
   udbsee -a -fupdate,uid,resgrp,shares,shusage,shcharge > tempfile
   ```

   The `uid`, `resgrp`, and `shares` fields will not be changed, but they may be useful as a reference during this operation.

2. Clear the fair-share information by editing *tempfile* as follows: replace the values in the `shrusage` and `shcharge` fields with 0.

3. Bring the system to single-user mode. It is important that the next step be performed when no other process is updating the UDB.

4. Use the `udbgen` command with the edited file, `tempfile`, to update the UDB with the changes removing the fair-share information.

5. Return the system to multiuser mode.

4.4.10 Costs, Usage, and Background Users

Watch the relationship between costs, usage, and the `MAXUSAGE` value. (`MAXUSAGE` is set by the `shradmin -U` command to the upper bound for reasonable usages; the default is a very large number.) If the cost factors are set high enough that a user's usage accumulation rises more quickly than the decay rate can bring it down, and the usage accumulation exceeds `MAXUSAGE`, then that user is put into the background user category by the fair-share scheduler. This situation can be controlled by either raising `MAXUSAGE` with the `shradmin -U` command, or by adjusting the following cost factors in a relatively uniform downward direction: `bio` (block I/O operations), `tio` (stream I/O operations), `click` (memory ticks), `syscall` (system calls), and `tick` (CPU ticks).

The tick cost, which is charged to the user up to 60 times per second, is usually the cost that causes users' usages to rise very rapidly. A background user can be recognized as running at priority 996 in a `ps`(1) display.

Processes that have been assigned a nice value of 39 are treated as special by the fair-share scheduler. They run at priority 997 and are not charged for CPU use. Processes attached to lnodes with 0 shares run at priority 998.

See "Tuning the fair-share scheduler," Section 4.6, page 229, for more information on fair-share costs and nice values, or for information on increasing the default `MAXUSAGE` value.

4.4.11 Monitoring the Fair-share Scheduler

Three commands display information about what is currently happening in a fair-share system: `shrview`(1), `shrtree`(8), and `shrmon`(8). The `shrview`(1) command provides the most functionality; it serves as a single interface for the functions of the other fair-share display commands. The `shrtree`(8) command displays the fair-share hierarchy. The `shrmon`(8) command is intended for use by the administrator; it is located in the `/etc` directory.

Note: The `shrmon`(8) command will not be available in future releases of the UNICOS operating system. Its functionality is replaced by the `shrview`(1) command.

All fair-share display commands can be executed in line mode, repeat mode, or continuous mode. In each mode, the time period used as a delay to collect information, or between successive display updates, is the share cycle time set by the `-R` option of the `shradmin`(8) command.

In line mode, the command collects information and prints its findings to standard output (`stdout`). In repeat mode, the command behaves as if in line mode and repeats the cycle as many times as specified with the `-r` option to `shrview`, separating the outputs with a form-feed character. In continuous mode, the command uses the `curses`(3) library capability and produces a full-screen display that it updates periodically. For displays involving more data than can be displayed (for example, 40 users on a 24-line terminal screen), only the first screen is displayed.

4.4.11.1 Using the `shriview`(1) Command

The `shrview` command is an integrated tool for displaying information about the behavior and current state of the fair-share scheduler. Many different display options and formats are available. Selection and configuration of displays may be done interactively when in curses mode.

The following sample display shows the `ADJGROUPS` display (`-da` option) of resource groups (`-so` option), organized by group name in alphabetical order (`-oi` option). This display helps quantify system use by resource group; the `Rate%`, `CPU%`, and `Rshr%` columns show that resource group `Xydev` in `SoftDev` is using the largest percentage of the system.

```
% shrview -da -so -oi

  SHRVIEW  Type:adjgroup  Select:only groups  Sort_opt:id

    Name      Rate%   CPU%  Rshr%   Nrun  Rate  Proj%    Adj_a   New%   Pri%
------------  -----  -----  -----  -----  ----  -----  -------  -----  -----
 CCN           0.00   0.00  26.67   0.00  0.00  20.92     1.00  16.63  15.67
  SysAdm       0.00   0.00  17.78   0.00  0.00  13.82     1.00  10.99  10.35
  Syssup       0.00   0.00   8.89   0.00  0.00   7.10     1.00   5.64   5.32
 Mktg          0.31   0.23  13.33   0.00  0.00  56.65     1.00  45.05   9.99
  Country      0.09   0.00   4.44   0.00  0.00  53.91     1.00  42.87   6.66
  Intl         0.22   0.23   4.44   0.00  0.00   0.83     1.00   0.66   1.66
  TechOps      0.00   0.00   4.44   0.00  0.00   1.91     1.00   1.52   1.66
 SoftDev      97.62  99.77  60.00  65.76 21.48  22.42     1.00  38.31  74.34
  Userint     12.37  21.14  10.00  14.00 12.31   3.73     1.00   2.97   9.05
  Users        3.29   7.95  10.00  11.76  1.94  11.30     1.00  22.09  18.66
   Netdev      1.59   3.86   2.00   2.00  1.00   0.00     1.00   0.00   0.75
  Xydev       81.96  70.68  40.00  40.00  7.22   7.39     1.00  13.25  46.63
```

The following example shows the monitor display (`-dm` option) on a system with two levels of hierarchies.

```
# shrview -dm

Shrview: sampling over 11 seconds starting at Tue Jul 25 12:27:33 1995

  SHRVIEW  Type:monitor  Select:all  Sort_opt:id

    Name      Rate%  Eshr%  Rshr%   Usage(k)    Rate   Muse  Ref  Chld   Flags
------------  -----  -----  -----  ----------  -----  -----  ---  ----  ------
 CCN          93.84  11.76  11.76  1000000000   1.00       0    0     2   50000
  SysAdm      93.84   7.84   7.84   199667501   0.84 1358396  12     0 1010000
  Syssup       0.00   3.92   3.92   570650974   0.84 489820    1     0 1010000
 Idle          0.00   0.00   0.10  1000000000  17.84      24    2     0   10000
 Mktg          6.16  11.76  11.76  1000000000   1.00       0    0     2   50000
  Country      0.00   5.88   5.88   119543509   0.84   29760    2     0 1010000
  TechOps      6.16   5.88   5.88    41308301   0.84    3652    1     0 1010000
 SoftDev       0.00  52.94  52.94  1000000000   1.11       0    0     4   50000
  Netdev       0.00  13.24  13.24    67308849   0.84   83308    6     0 1010000
  Userint      0.00  13.24  13.24   203251100   1.11 541828   23     0 1010000
  Users        0.00  13.24  13.24   200168203   0.84 1609728   8     0 1010000
 *Xydev        0.00  13.24  13.24   144301956   0.94 236500   19     0 1010000
 System        0.00  23.53  23.53  1000000000   1.00       0    0     1   50000
```

```
  Admin        0.00  23.53  23.53    10437769   0.84 428412    1     0 1010000
```

4.4.11.2 Using the `shrmon`(8) Command

The fair-share monitor, `shrmon`(8), produces a columnar formatted output that can be used to monitor most of the more important accumulators and feedback parameters found in the fair-share scheduling node within the kernel.

Note: The `shrmon`(8) command will not be available in future releases of the UNICOS operating system. Its functionality is replaced by the `shrview`(1) command.

The following display shows sample `shrmon` output:

```
$  shrmon -vv
Tue Sep 20 09:06:20 1988
 CPUs charge   rate nrun  %Rate %share  %CPU %rshare kids  ref
 11.2    0.0    1.3   13    0.0  100.0 100.0  100.0   12   59 System
  0.0    0.0    3.8    4    0.0    0.0   0.0    0.0    0    4  Idle
  3.4    0.2    1.3    2    0.0    8.3  30.5   33.3    2    9  Regions
  0.0    0.0    0.8    0    0.0    4.2   0.0    2.0    0    3   poulet
  3.4    0.2    1.4    2    0.0    4.2  30.5   33.3    0    6   ub
  0.0    0.0    0.0    0    0.0    8.3   0.0    2.0    2    2  LibProd
  0.0    0.0    0.8    0    0.0    4.2   0.0    2.0    0    1   saa
  0.0    0.0    0.8    0    0.0    4.2   0.0    2.0    0    1   jam
  7.7    0.3    5.0    5    0.0   16.7  69.4   66.7    3   22  OpSysDev
  0.0    0.0    0.8    0    0.0    5.6   0.0    2.0    0    1   dak
  0.0    0.0    1.0    1    0.0    5.6   0.0    2.0    0    3   sen
  7.7    0.4    5.0    5    0.0    5.6  69.4   66.7    0   20   dws
  0.0    0.0    0.0    0    0.0    8.3   0.1    2.2    4   11  Operatns
  0.0    0.0    0.8    0    0.0    2.1   0.1    2.0    0    3  operator
  0.0    0.0    0.8    0    0.0    2.1   0.0    2.0    0    3   tqa
  0.0    0.0    0.8    0    0.0    2.1   0.0    2.0    0    5   backup
  0.0    0.0    0.8    0    0.0    2.1   0.0    2.2    0    0   ce
  0.0    0.0    0.8    0    0.0    2.8   0.0    2.0    0    1   mab
  0.0    0.0    0.8    0    0.0    2.8   0.0    2.0    0    4   gop
  0.0    0.0    0.8    0    0.0    2.8   0.0    2.0    0    1   aaw
  0.0    0.0    0.0    0    0.0    8.3   0.0    2.0    1    1  Appl
  0.0    0.0    0.8    0    0.0    8.3   0.0    2.0    0    1   wgh
  0.0    0.0    0.0    0    0.0    8.3   0.0    2.0    1    0  IOS
  0.0    0.0    0.8    0    0.0    8.3   0.0    2.0    0    0   stt
  0.0    0.0    0.0    0    0.0    8.3   0.0    2.0    1    2  Services
  0.0    0.0    0.8    0    0.0    8.3   0.0    2.0    0    2   hhj
  0.0    0.0    0.0    0    0.0    8.3   0.0    2.0    1    1  CompDev
  0.0    0.0    0.8    0    0.0    8.3   0.0    2.0    0    1   plu
  0.0    0.0    0.0    0    0.0    8.3   0.0    2.0    1    1  Diag
  0.0    0.0    0.8    0    0.0    8.3   0.0    2.0    0    1   ljl
  0.0    0.0    0.0    0    0.0    8.3   0.0    2.0    1    0  Netwrkng
  0.0    0.0    0.8    0    0.0    8.3   0.0    1.9    0    0   jyj
```

The most important factors in this display are `%rshare` and `%CPU`. The `%rshare` factor is the machine share of the system, expressed as a percentage, to which this particular user or group is entitled based on the current active users and the fair-share hierarchy defined in the UDB. The `%CPU` factor is the percentage of CPU resources allocated to the lnode over the sampling interval.

Interactive users, who are relatively idle at the time of the sample, each hold a 2.0% `rshare`.

Note that the fair-share hierarchy in this example involves two levels: group levels have names beginning with a capital letter; and users are identified by lowercase login names.

4.4.11.3 Using the `shrtree`(8) Command

The `shrtree`(8) command displays the share tree, or fair-share hierarchy, that is defined in the UDB, and highlights problems that could prevent logging in to the system or submitting jobs. Any problems in the UDB that would affect the fair-share scheduler are marked in the `shrtree` output. (Most problems can prevent logging in and submitting jobs for the affected users.)

The `shrtree` command displays such useful statistics as the group share and usage value from the UDB. This command also displays a general approximation of share distribution by displaying a static analysis of what the relative entitlements would be if all the users in the UDB were logged on at the same time.

The following display shows sample output for `shrtree` with no options (short form report). The system is running Share by User mode, with a maximum of four levels in the fair-share hierarchy. In this example, the `Serv` entry has warning flag `Nc`; `Unknown` has warning flag `Zs`; and `Country`, `Region`, and `TechOps` have flags `Nc` and `Zs`.

```
$ shrtree

DISPLAY OF SHARE TREE
_____________________
UDB path:      DEFAULT
```

```
Analyzed:      By UID
Format:        Groups only
Maxgroups:     4
Node:          ALL
Group Count:   15
Account Count: 0
User Count:    1370
Warnings:      9
Errors:        0

Warning Count: 4     (Nc) Group has no references
Warning Count: 1     (Zs) User has zero shares
Warning Count: 4     (Zs) Group has zero shares

                                    System Group  System
Lv        Name                  ID     Share  Share  Usage  Status   Flags
__ __________________________ ________ ______ ______ ______ ________ ________
 0 _ROOT_                            0 100.0% 100.0% 100.0% G           40000
 1   Demos                        8367 100.0% 100.0%   0.0% G           40000
 2     Serv                       8001 100.0% 100.0%   0.0% G|Nc        40000
 1   System                       8389   0.0%   0.0%   0.0% G           40000
 2     Admin                      8306   0.0%   2.9%   0.0% G           40000
 1   Unknown                      8393   0.0%   0.0%   0.0% G|Zs        40000
 1   Users                        8395   0.0%   0.0%  28.8% G           40000
 2     CCN                        8354   0.0%   4.8%   0.2% G           40000
 2     Country                    8359   0.0%   0.0%   0.0% G|Nc|Zs     40000
 2     HardDev                    8372   0.0%   4.8%   0.0% G           40000
 2     Intl                       8374   0.0%  14.3%   0.0% G           40000
 2     Mktg                       8381   0.0%  28.6%   0.1% G           40000
 2     Region                     8385   0.0%   0.0%   0.0% G|Nc|Zs     40000
 2     SoftDev                    8386   0.0%  47.6%  28.5% G           40000
 2     TechOps                    8390   0.0%   0.0%   0.0% G|Nc|Zs     40000
```

The following example shows the error display for the same system (`shrtree` with the `-e` option). Only the entries with errors are displayed.

```
$ shrtree -e

DISPLAY OF SHARE TREE
_____________________
UDB path:      DEFAULT
Analyzed:      By UID
Format:        Groups only
```

```
Maxgroups:     4
Node:          ALL
Group Count:   15
Account Count: 0
User Count:    1370
Warnings:      9
Errors:        0

Warning Count: 4     (Nc) Group has no references
Warning Count: 1     (Zs) User has zero shares
Warning Count: 4     (Zs) Group has zero shares

  Type      Name    ID    Status  Description
_________ ________ _____ ________ ____________________________________
**WARN*** Serv      8001       10 Nc: Group has no references
**WARN*** Unknown   8393     1000 Zs: Group has zero shares
**WARN*** unknown     12     1001 Zs: User has zero shares
**WARN*** Country   8359     1010 Nc: Group has no references
**WARN*** Country   8359     1010 Zs: Group has zero shares
**WARN*** Region    8385     1010 Nc: Group has no references
**WARN*** Region    8385     1010 Zs: Group has zero shares
**WARN*** TechOps   8390     1010 Nc: Group has no references
**WARN*** TechOps   8390     1010 Zs: Group has zero shares
```

4.5 Customizing the Fair-share Scheduler (User Exit)

You can customize the fair-share scheduler's CPU scheduling policy at your site. This allows use of a different scheduling algorithm without modifying the UNICOS kernel.

To customize the scheduling policy, you provide a user exit routine and set the `USRLEVLFSS` flag with the `shradmin`(8) command. The fair-share daemon, `shrdaemon`(8), includes a stub routine (in the module `shrsiteux.c`) that has an entry point named `site_adjust_lnodes`. After reading the lnodes from the kernel and performing the calculations indicated by the share flags, this user exit is invoked before writing the lnodes back to the kernel.

The following calling sequence is defined in the include file `sys/lnode.h`:

```
site_adjust_lnodes( &Lnodes[0], count, Traceflag)
```

The arguments of `site_adjust_lnodes` have the following meanings:

Argument	Description
`Lnodes`	This argument is a pointer to the lnode table in the `shrdaemon`(8) memory; it is of type `struct kern_lnode *Lnodes`. All the address fields in the lnode table can be used for scanning, because `shrdaemon` converts them to `shrdaemon` memory addresses before performing any analysis.
`count`	This parameter specifies the number of lnodes in the lnode array; it is of type `int count`.
`Traceflag`	This parameter is of type `int Traceflag`; it can be used to specify logging, as in the following example: `if (Traceflag) fprintf(stderr, "Adjusting lnodes\n").`

4.5.1 Fair-share User Exit Example

As a simple example of a user exit, the following routine allows UID `104` to use as many CPU resources as necessary. This capability allows a system daemon to run under a different UID than `root` (some sites prefer to do this for accounting reasons), but removes the resource limitations that are normally enforced for all non- `root` UIDs.

```
static char USMID[] = "@(#)man/2302/04.fair.share       92.1    12/11/95 16:48:26";

/*
 *      (C) COPYRIGHT INC.
 *      UNPUBLISHED PROPRIETARY INFORMATION.
 *      ALL RIGHTS RESERVED.
 */

#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/param.h>
#include <sys/lnode.h>
#include <sys/share.h>
/*
```

```
 *      site_adjust_lnodes()
 *
 *      User exit stub to allow sites to perform any calculations
 *      desired upon lnode data before the shrdaemon writes them
 *      back to kernel memory.
 *
 *      A ptr to the lnode table in the shrdaemon memory is passed
 *      in; all the address linkages have been converted to shrdaemon
 *      memory addresses.
 */

void
site_adjust_lnodes(struct kern_lnode *Lnodes, int count, int Traceflag)
{
        int i;
        struct kern_lnode *lp;

        /*
         * Site can now perform any calculations desired on lnode data.
         */

        for (lp = Lnodes, i = 0; i < count; i++, lp++) {
            if (lp->kl.l_uid == 104) {
                /* This user is special, must never starve for cpu. */
                lp->kl.l_usage = 2.0;
                lp->kl_usage = 2.0;

            }
                if (Traceflag) {
                        fprintf(stderr, "Adjusting lnode %s.0, lp->kl.l_name);
                        fprintf(stderr, "l_uid %d0, lp->kl.l_uid);
                        fprintf(stderr, "l_usage %g0, lp->kl.l_usage);
                        fprintf(stderr, "l_charge %g0, lp->kl.l_charge);
                        fprintf(stderr, "kl_usage %g0, lp->kl_usage);
                        fprintf(stderr, "kl_totuse %g0, lp->kl_totuse);
                        fprintf(stderr, "kl_adj %g0, lp->kl_adj);
                        fprintf(stderr, "kl_rate %g0, lp->kl_rate);
                        fprintf(stderr, "kl_cost %d0, lp->kl_cost);
                        fprintf(stderr, "0);
                        fflush(stdout);
                }
        }
        return;
}
```

4.6 Tuning the Fair-share Scheduler

This section discusses tuning issues relating to the fair-share scheduler. If you run fair-share, it is important to consider other factors such as memory scheduling and NQS configuration to achieve your CPU scheduling goals.

This section discusses the following:

- Fair-share parameters
- Memory scheduling parameters and fair-share
- Priority-based scheduling with nice values (this item is not strictly about fair-share; but, it is a related topic)

4.6.1 Fair-share Parameters

Fair-share parameters provide you with considerable control over the behavior of the fair-share scheduler but do not allow you to directly affect system throughput. Before attempting to tune fair-share, it is important to establish a clear set of CPU scheduling goals. Tuning fair-share usually involves a compromise between accurate distribution of resources by share allocations and interactive responsiveness.

This section describes the `shradmin`(8) options most useful for tuning the fair-share scheduler and provides an example of possible parameter settings. For more information on fair-share parameters, see the `shradmin`(8) man page.

4.6.1.1 `shradmin`(8) Options Affecting Cost

The following options for the `shradmin`(8) command allow you to set the cost for various system resources:

Option	Description
`-b`	Cost for logical I/O request
`-m`	Cost for each memory click
`-s`	Cost for each system call
`-t`	Cost for a CPU tick

-y Cost for a tty read / write operation

The shradmin command allows you to apply a percent multiplier to all of these cost factors. For example, the command shradmin 20 specifies that all cost values should be multiplied by the factor 0.2. By applying the multiplier, you can uniformly scale the cost parameters.

Because fair-share schedules only the CPU resources, it is recommended that costs be weighted heavily, or exclusively, for CPU use.

4.6.1.2 -U Option (MAXUSAGE)

If the usage value for a process reaches MAXUSAGE, that process hangs if there is no idle time (the priority of the process is set to 998, which prevents the process from getting the CPU). Therefore, MAXUSAGE should be set such that it will only take effect under extreme circumstances.

The default setting of MAXUSAGE is 1.0e^{+12}. This is too low for systems with 8 or more CPUs and fast cycle times. For a system with 8 CPUs, a better starting value is 1.0e^{+15}; for a system with 16 CPUs, a better starting value is 1.0e^{+18}. For any site, if a very long usage decay rate (greater than 24 hours) or large cost values are used (greater than 1000), it is recommended that this value be increased.

Use the shrview(1) command with the -ds option to obtain statistics on process usage values. This command samples the average number of users logged on at one time, as in the following example:

```
% shrview -ds
Shrview: sampling over 5 seconds starting at Thu Aug  5 16:55:14 1993

  SHRVIEW  Type:statistics  Select:all  Sort_opt:id

                                               High         Max
                                            ----------  ----------
Active id's:   62/300           Usage:     5.9689e+09  1.0000e+12
Active groups:  6               Share_pri: 3.8694e+03  1.0000e+28

          syscall       bio         sio         tick       click
        -----------  ----------  ----------  ----------  ----------
Charge:          0%          0%          0%        100%          0%
Costs:            0           0           0         100           0
Counts:   501753913     6866002           0    15649624           0
```

If *clipping* is occurring on `MAXUSAGE` (that is, if the actual usage is consistently close to the `MAXUSAGE` setting), increase the value for `MAXUSAGE` in the `/etc/config/daemons` file.

4.6.1.3 `-D` Option (*priority Decay Rates*)

The `-D` option sets *priority decay rates* for processes in the following areas and alters the effect of nice:

- Normal nice values (20)
- Maximum nice values (39)

These two values are used to build a table of priority decay rates for each nice value (0 through 39). As the difference between the two rates increases, the effect of nice is more pronounced. This effect is not related directly to fair-share; it occurs even if fair-share is not enabled. Shorter decay rates cause the scheduler to act in more of a round-robin fashion. Longer values cause the scheduler to be more influenced by fair-share information.

The effect of this option is subtle and might have undesirable side effects. Long decay rates can cause *marooning*, where a compute-bound process can control the CPU(s) for long periods of time. The default values for the `-D` option (2,60) are recommended, and any changes should be carefully considered. Do not use values of less than (1,30) because this generates negative decay rates for some of the lower nice values.

The `shrinfo` (1) command used with the `-v` option shows the decay rates for the nice values of 0, 20, and 39, as in the following example:

```
Scheduling flags    = ADJGROUPS, LIMSHARE
Charging percentage = 100%
Usage decay rate    = 0.95484160 (half-life of  60.0 seconds)
Active users = 59/300, active groups = 6.

     ---syscall-- ----bio---- ----tio--------tick--- ---click--- (NULL)
Charge:         0%          0%          0%        100%          0%
Costs:           0           0           0         100           0
Counts:  494584243     6698323           0    15173200           0

Process priority decay rate biased by "nice":-
high priority (nice -20) 0.4044 (half-life of   0.8 seconds),
avg  priority (nice   0) 0.7039 (half-life of   2.0 seconds),
low  priority (nice  19) 0.9885 (half-life of  60.0 seconds).
Run rate decay rate      0.8409 (half-life of   4.0 seconds).
```

```
Max. value for normal usage        = 1.000000e+12,
Max. value for normal p_sharepri   = 1.000000e+28,
Max. value for idle   p_sharepri   = 1.000000e+38.
High value of current normal usage = 3.369134e+10,
High value of current p_sharepri   = 9.236291e+03.

User <anne> owns 100 shares.
```

The `shrview -dn` command shows a table of decay rates for each nice value, as in the following example:

```
Shrview: sampling over 5 seconds starting at Thu Aug5 16:41:36 1993

  SHRVIEW  Type:nice tables  Select:all  Sort_opt:id

             NiceDecays       Nice   Nice
 N  Nice   Decay  1/2life   Rates  Ticks
--  ----  ------  -------  ------  -----
 0   -20  0.4044    0.766  0.1591    150
 1   -19  0.4194    0.798  0.1591    147
 2   -18  0.4343    0.831  0.1591    145
 3   -17  0.4493    0.866  0.1591    142
 4   -16  0.4643    0.903  0.1591    140
 5   -15  0.4793    0.942  0.1591    137
 6   -14  0.4943    0.984  0.1591    135
 7   -13  0.5092    1.027  0.1591    132
 8   -12  0.5242    1.073  0.1591    130
 9   -11  0.5392    1.122  0.1591    127
10   -10  0.5542    1.174  0.1591    125
                .
                .
                .
17    -3  0.6590    1.662  0.1591    107
18    -2  0.6740    1.757  0.1591    105
19    -1  0.6890    1.860  0.1591    102
20     0  0.7039    1.974  0.1591    100
21     1  0.7189    2.100  0.1515     97
22     2  0.7339    2.240  0.1446     95
23     3  0.7489    2.397  0.1384     92
                .
                .
                .
```

```
37    17  0.9586   16.377  0.0860     57
38    18  0.9735   25.844  0.0837     55
39    19  0.9885   60.000  0.0000      1
```

4.6.1.4 `-X` Option (*maxushare*)

The `-X` option (*maxushare*) defines the maximum effective share for an individual user, when the global scheduling flag `LIMSHARE` is set (with the `shradmin -F` command). This option can be used to minimize marooning by preventing users with very low past usage from monopolizing the CPU or by preventing users with very high past usage from being locked out.

Values close to 1 ensure good interactive response regardless of past usage and provide a leveling effect so that all users will have scheduling priorities that correlate closely to their share allocations. Increasing this value allows scheduling priorities to deviate more from share allocations as determined by past usage.

The default value of 2.0 is a good initial value if interactive use is important at your site. Values of 8.0 and greater are good for a batch environment. If interactive response is not important, and consistent distribution of resources has the highest priority, you should turn off this option by setting the `LIMSHARE` flag equal to 0. This option must be set to a value greater than 1.0.

You can view the effect of the *maxushare* option with the `shrview -dl` command.

4.6.1.5 `-Y` Option (*mingshare*)

The `-Y` option (*mingshare*) specifies the deviation between group share allocation and the actual rate of resource consumption allowed before the scheduling algorithm tries to compensate. It has an effect only when the global scheduling flag `ADJGROUPS` is set (with the `shradmin -F` command). This option can diminish to some degree the effect of the `-X` (*maxushare*) option.

The default setting of .75 allows a 25% or less deviation. Values closer to 1.0 allow less deviation, but may interfere with *maxushare*. This option must be set to less than 1.0.

When accurate distribution of resources based on group share allocations is a high priority, this option is very important and should be set near to 1.0 (.90 to .95).

The effect of the *mingshare* option can be viewed with the `shrview -da` command, as in the following example:

```
% shrview -da

Shrview: sampling over 5 seconds starting at Thu Aug  5 16:47:12 1993

  SHRVIEW  Type:adjgroup  Select:all  Sort_opt:id

    Name      Rate%   CPU%  Rshr%   Nrun  Rate  Proj%    Adj_a   New%   Pri%
------------ ----- ----- -----  -----  ----  -----  -------  -----  -----
 Idle          0.00   3.11   0.02   0.00  7.62   0.00     1.00   0.00   0.00
 System        4.52   6.14  10.00   0.00  0.00   0.00     1.00   0.00   8.12
  operator     0.00   0.00   5.00   0.00  0.84   0.00     1.00   0.00   6.50
  jkl          4.52   6.14   5.00   5.00  1.00   0.00     1.00   0.00   1.62
 Users        63.65  45.37  90.00   0.00 19.48   0.00     1.00   0.00  29.24
  CCN          0.00   0.00   5.00   0.00  0.84   0.00     1.00   0.00   1.62
   abc         0.00   0.00   1.64   0.00  0.84   0.00     1.00   0.00   0.53
   zyx         0.00   0.00   1.64   0.00  0.84   0.00     1.00   0.00   2.13
   c1234       0.00   0.00   1.64   0.00  0.84   0.00     1.00   0.00   2.13
  HardDev      5.41   2.43   5.00   0.00  0.84   0.00     1.00   0.00   1.62
   paul        2.71   2.43   4.76   0.00  0.87   0.00     1.00   0.00   1.55
  Mktg         0.17   0.05  30.00   0.00  0.84   0.00     1.00   0.00   9.75
  *anne        0.08   0.05   3.61   0.00  0.84   0.00     1.00   0.00   1.17
   steven      0.00   0.00   3.61   0.00  0.84   0.00     1.00   0.00   1.17
   c0987       0.00   0.00   3.61   0.00  0.84   0.00     1.00   0.00   1.17
   gregj       0.00   0.00   3.61   0.00  0.84   0.00     1.00   0.00   4.70
  SoftDev     58.07  42.90  50.00   0.00 13.77   0.00     1.00   0.00  16.25
   ali         0.00   0.00   1.20   0.00  0.84   0.00     1.00   0.00   1.57
   birk        0.00   0.00   1.20   0.00  0.84 100.00     1.00 100.00   1.57
   bobo        0.17   0.18   1.20   1.20  1.19   0.00     1.00   0.00   0.39
   cde         0.00   0.00   1.20   0.00  0.84   0.00     1.00   0.00   1.57
                .
                .
                .
   zzzyzzy     0.08   0.60   1.20   0.00  0.84   0.00     1.00   0.00   0.39
```

4.6.1.6 -Z Option (*sharemin*)

The -Z option (*sharemin*) specifies the minimum machine share allocated to a user. This option, when combined with the *maxushare* option, sets a range for individual priorities to allow reasonable interactive response for all users. The default value of 0 sets no minimum share allocation for users. It is recommended that you select a nonzero value for this option to provide reasonable response for all users. Select a value based on the reciprocal of the average number of users logged on (1 *average_users*), rounded to three digits.

For example, use the value 0.013 on a system with an average of 80 users logged on at any one time.

The effect of the `-Z` option can be seen by comparing the `Eshr` and `Rshr` columns from the `shrview -dm` output, as in the following example. The values in these columns should be as close as possible.

```
Shrview: sampling over 5 seconds starting at Thu Aug  5 16:57:21 1993

  SHRVIEW  Type:monitor  Select:all  Sort_opt:id

    Name      Rate%  Eshr%  Rshr%   Usage(k)    Rate   Muse  Ref  Chld   Flags
------------  -----  -----  -----  ----------  -----  -----  ---  ----  ------
 Idle          0.00   0.00   0.02  1000000000   7.62     56    8     0   10000
 System        0.00  10.00  10.00  1000000000   0.00      0    6     2   50000
  operator     0.00   5.00   5.00           0   0.84 1210410   1     0   10000
  jkl          0.00   5.00   5.00           0   0.84  48068    5     0   10000
 Users        66.67  90.00  90.00  1000000000  15.93      0 1598474  4   50000
  CCN          0.00   5.00   5.00           0   0.84      0 60666    3   10000
   abc         0.00   1.64   1.64           0   0.84  32836    4     0   10000
   zyx         0.00   1.64   1.64           0   0.84   3475    3     0   10000
   c1234       0.00   1.64   1.64           0   0.84  14100    3     0   10000
  HardDev      1.44   5.00   5.00           0   0.84      0 14813    1   10000
   paul        0.72   4.76   4.76           0   0.94  81902    9     0   10000
  Mktg        10.41  30.00  30.00           0   1.02      0 211197   7   10000
  *anne        0.06   4.11   4.11           0   0.84   3536    3     0   10000
   steven      0.00   4.11   4.11           0   0.84   2626    2     0   10000
   c0987       0.00   4.11   4.11           0   0.84   3414    1     0   10000
   gregj       0.00   4.11   4.11           0   0.84   2848    1     0   10000
  SoftDev     54.81  50.00  50.00           0   8.46      0 1311798  41   10000
   ali         0.00   1.20   1.20           0   0.84   5720    1     0   10000
   birk        0.00   1.20   1.20           0   0.84   6638    1     0   10000
   bobo        0.23   1.20   1.20           0   1.82 131743   11     0   10000
   cde         0.00   1.20   1.20           0   0.84  14404    1     0   10000
                .
                .
                .
   zzzyzzy     0.00   1.20   1.20           0   0.84  89449    4     0   10000
```

4.6.1.7 `-R` Option (*delta*)

The `-R` option (*delta*) determines how often usage values are updated and is directly related to the overhead for fair-share processing. Shorter values increase the scheduler's responsiveness to changing conditions, but they increase overhead longer values decrease overhead but also decrease responsiveness.

The default of 4 seconds is a good compromise for interactive environments. For batch environments, the value can be increased; however, overhead is increased by the number of active lnodes and levels in the fair-share hierarchy.

4.6.1.8 `-K` Option (*usage Decay Rate*)

The `-K` option (*usage decay rate*) establishes the length of time that past usage of resources are remembered. Longer decay rates cause the scheduler to distribute resources fairly over a longer period of time. This option can have a significant impact on interactive responsiveness. On a heavily loaded machine with a large number of users, short decay rates provide uniformly poor response for all users with small share allocations. Longer rates will provide good interactive response for users that have used less than their allocated share and worse response for those who have used more. Very short usage decay rates (less than 3600 seconds, or 60 minutes) limit the effect of the *maxushare* and *mingshare* options. For these reasons, decay rates greater than 60 minutes will generally provide more accurate distribution of resources and better interactive response times.

Note: Do not set the decay rate to less than 1 hour (the default) for general fair-share operation. Decay rates of minutes or seconds are recommended only for debugging purposes.

4.6.1.9 Example Parameter Settings

The following parameter settings show sample values at a site presently running fair-share. The goals for choosing these settings were to provide the following:

- Good interactive response during prime time regardless of the share allocations or past usage
- Close tracking of share allocations and usage during nonprime time
- Incentive for nonprime time usage
- Usage decay rate that would be similar in effect to the one-week refreshing of "bank point" (a local concept for this site)

- Limit on the impact of long-running batch jobs on interactive performance

The following option settings are used during prime time:

Option setting	Description
`-D 2,60`	Priority decay rate
`-F 006`	`LIMSHARE` and `ADJGROUPS` flags
`-K 120`	Usage decay rate of 5 days
`-R 4`	*delta*
`-X 2.0`	*maxushare*
`-Y 0.7`	*mingshare*
`-Z .02`	*sharemin* (2%; default)
`-b 2`	Cost of block I/O
`-m 2`	Cost of memory click
`-t 200`	Cost per CPU tick
`-s 2`	Cost of system calls
`100`	Percent multiplier (100%)

The following option settings are used during nonprime time:

Option setting	Description
`-D 2,60`	Priority decay rate
`-F 006`	`LIMSHARE` and `ADJGROUPS` flags
`-K 120`	Usage decay rate of 5 days
`-R 8`	*delta*
`-X 8.0`	*maxushare*
`-Y 0.9`	*mingshare*
`-Z .01`	*sharemin* (1%)
`-b 2`	Cost of block I/O
`-m 2`	Cost of memory click
`-t 200`	Cost per CPU tick

`-s 2`	Cost of system calls
`60,40`	Percent multiplier (60% night, 40% weekend)

4.6.2 Fair-share and the Memory Scheduler

When using the fair-share scheduler, it is important to consider how fair-share interacts with the memory scheduler. If you are using fair-share, the memory scheduler parameters should be set to achieve the CPU scheduling goals without excessive swapping overhead. If your goals are to base scheduling completely on share allocations, it is possible that the system could become idle as large memory processes dominate the system. Using both the memory scheduler and the fair-share scheduler provides your site with a compromise between improving system throughput and meeting scheduling objectives.

Deciding what values to use in order to achieve the desired results is more difficult than if you were using only the memory factors or only the priority factors. It is important to investigate the following areas:

- Memory oversubscription
- Fair-share priorities of jobs in relation to memory size (in other words, does a user with low fair-share priority run big memory jobs and a user with high fair-share priorities run small memory jobs?)

Using this type of scheduling ensures that processes with high fair-share priorities are more likely to be in memory without causing excessive swapping and system overhead.

For more information on the memory scheduler, see the `nschedv`(8) man page in the *UNICOS Administrator Commands Reference Manual*.

4.6.2.1 Priority-based Scheduling and I/O Resources

When you use both fair-share and priority memory scheduling, it is important to consider how the system I/O resources are used. If a job with a low fair-share priority uses a large amount of disk or SDS resources, significant problems could result for the entire system.

If you can determine that I/O-intensive jobs are a problem for your system, consider changing the memory scheduling parameters so that either this type of job is favored or has equal access to memory. Changing fair-share parameters might also be necessary. Getting this job out of the system as soon as possible helps the total system throughput.

4.7 Using CPU Quotas

The CPU quota feature allows you to control the total CPU time used by each user login or account on the system in increments of tenths of seconds. CPU quotas function only when your site is running the fair-share scheduler.

The CPU quota feature is similar to the CPU limits feature, but resource consumption information is accumulated for the user rather than the job or process. When a user reaches the quota, a `SIGCPULIM` signal is sent to the user's processes. (The `SIGCPULIM` signal is ignored by default.) When the user reaches a specified threshold above the quota (the default is 3 seconds), the kernel sends a `SIGKILL` signal to all the user's processes and terminates them.

The UDB contains a quota field (`cpuquota`) and a time-used field (`cpuquotaused`) that set a user's CPU quota in seconds and the amount of CPU time used in seconds, respectively. You can use the `udbgen`(8) command to update these fields. For example, to set a quota of 10 seconds for user `xyz`, you would enter the following command:

```
udbgen -c 'update:xyz: cpuquota:10:'
```

To cancel the accumulated time for user `xyz`, you would enter the following command:

```
udbgen -c 'update:xyz: cpuquotaused:0:'
```

The amount of time that a user accumulates while running jobs (the *accumulated CPU time*) is calculated by the kernel. After a user's last session exits the system, the fair-share daemon updates the `cpuquotaused` field in the UDB (as well as other fields) with the accumulated CPU time. During this process, any changes that have been made to an active user's `cpuquotaused` field in the UDB are overwritten with the old value from the UDB (plus the latest accumulated CPU time). Therefore, you must take extra steps to ensure that changes made to an active user's `cpuquotaused` field, such as clearing the field, remain in effect after the user has exited the system.

To ensure that changes to an active user's `cpuquotaused` field are not lost, use the `shrsync`(8) command. This command synchronizes various fields in the UDB, including the `cpuquotaused` field, with the corresponding data in the kernel.

The following procedure shows the `shrsync` commands used to update the `cpuquotaused` fields in the UDB while in multiuser mode:

1. Use the `-u` option of the `shrsync` command to update the UDB with the information from the active system. Because the active system will be

updated with information from the UDB, the information in the UDB must first be brought up to date (in particular, the CPU-quota-used information of the running sessions).

```
/etc/shrsync -u
```

2. Update the `cpuquotaused` fields in the UDB, as in the following example for user `xyz`:

```
/etc/udbgen -c 'update:xyzfP: cpuquotaused:0:' ; ...
```

3. Use the `-q` option of the `shrsync` command to indicate that all active users will have their `cpuquotaused` information updated on the system from the UDB, as follows:

```
/etc/shrsync -q
```

See the `udbgen`(8) man page for more information on the CPU quota fields; see the `shrsync`(8) man page for more information on synchronizing UDB and kernel information.

4.8 Additional Reference Material

The paper, "A Fair Share Scheduler," by J. Kay and P. Lauder, was published in the January 1988 (volume 31, number 1) issue of *Communications of the ACM* (pages 44-55). Although the UNICOS implementation does not exactly parallel the description as published, it is substantially similar, and this paper is recommended to anyone interested in the theory and philosophy of this scheduling mechanism.

The paper "The Fair Share Scheduler," by G.J. Henry, was published in the October 1984 *Bell Labs Technical Journal*, LVIII-8b. This paper also contains design information about the fair-share scheduler.

File System Quotas [5]

The file system quota enforcement feature allows you to control the amount of file system space in blocks and the number of files used by each account, group, and user on an individual basis. Controls may be applied to some or all of the configured file systems, except for the root file system. Attempts to exceed these limits result in an error similar to the error that occurs if the file system is out of free space. Optional warning levels are also available for informing users when usage gets close to the quota.

Warning: This section contains warnings and information critical to the proper use of a Cray ML-Safe configuration of the UNICOS system.

5.1 Components of the File System Quota Feature

The following components make up the file system quota feature:

- Quota control files (see `quota`(5)).
- Kernel support and enforcement code.
- Quota administration tools (`quadmin`(8) and `qudu`(8)).
- Quota configuration additions to `/etc/fstab` (see `fstab`(5)).
- Quota reporting tool (`quota`(1)).
- Quota warning and limit signal (`SIGINFO`).
- Quota limit error numbers (`EQACT`, `EQGRP`, and `EQUSR`).
- User warning and limit message generation. This feature is provided automatically by the Korn, Bourne, and C login shells. The `quotamon`(1) command is no longer needed.

The quota control files contain all quota information and must be created and maintained correctly. The following sections concentrate on how to set up the quota files and perform the configuration tasks for various operation modes. The quota control files are discussed as though there were one file per controlled file system. Other modes of operation are described in Section 5.15.2, page 265, but you must fully understand the fundamentals of the feature before going on to more elaborate configurations.

Throughout this section, the term *ID* is a generic term meaning account, group, and user IDs. A distinction among the different IDs is needed only when you actually reference specific data structures.

> **Note:** If users on your system use the UNICOS Network File System (UNICOS NFS) to move files, the account numbers will be assigned by the NFS daemon, and the account quotas will not be accurate.

5.2 Enabling the Quota Feature

If you are not using the menu system to set the configuration, find the following line in the `/etc/config/config.mh` file:

```
#define CONFIG_FQUOTAS n
```

Edit this line to set *n* to 1 to enable the quota feature, or to set *n* to 0 to disable it. The kernel must be rebuilt for this change to become effective. For more information on the `config.mh` file, refer to Chapter 4 of the *UNICOS Configuration Administrator's Guide*.

If you are using the menu system, you can turn quotas on or off by selecting `on` or `off` in the `Configure System -> Major software configuration -> File quotas` menu of the UNICOS Installation and Configuration menu system. For more information on the UNICOS installation and configuration menu system (ICMS), refer to *UNICOS System Configuration Using ICMS*.

These methods apply to all systems that support UNICOS.

5.3 Changing the NQUOTA Value

The `NQUOTA` value represents the default number of in-core quota entries. If you are not running file system quotas, the default `NQUOTA` value is `0`. If you are running file system quotas, the default `NQUOTA` value is `1400`. Note that this value must equal `((NINODE/4)` + `NC_SIZE`). If desired, you can change this value to better fit your site's needs.

If you are not using the menu system, you can change the `NQUOTA` value by editing the following line in the `/usr/src/uts/cf.`*SerialNumber*`/config.h` file:

```
#define NQUOTA      1400
```

The kernel must be rebuilt for any changes to become effective. For more information on the `config.h` file, refer to Chapter 3 of the *UNICOS Configuration Administrator's Guide*.

If you are using the menu system, you can change the `NQUOTA` value in the `Configure System -> UNICOS Kernel Configuration -> Table Size Parameters -> In core quota entries (NQUOTA)` menu of the UNICOS Installation and Configuration menu system. For more information on the UNICOS installation and configuration menu system (ICMS), refer to *UNICOS System Configuration Using ICMS*.

These methods apply to all systems that support UNICOS.

5.4 Quotas and Data Migration

In the default setting, when data migration is turned on and a file is migrated, the space the file occupied is credited to the file owner's ID. When a file is brought back online, the number of blocks is added to the ID's file quota. If bringing a file back would violate an enforced quota limit, that file cannot be brought online.

With the optional aggregate quotas setting, the off-line and on-line date is tracked together, and users are limited in the total amount of space they can use. Because of this, unmigration of a file is always allowed. See Section 5.16, page 266, for more information.

5.5 Configuring Quotas

Information on file system quota configuration is contained in the `/etc/fstab` file (see `fstab`(5)). The instructions for creating quota control files assume that the `/etc/fstab` file has been set up correctly.

Warning: The `/etc/fstab` file is part of a Cray ML-Safe configuration of the UNICOS system. For more information on configuring the `/etc/fstab` file, see the UNICOS Configuration Administrator's Guide, publication SG-2303.

The options field in the `/etc/fstab` file includes a `quota` option, which can be in one of the following three formats. Each of these formats is discussed in one of the following sections.

`quota=`*quota_file_relative_name*

`quota=`*quota_file_full_name*

`quota=/dev/dsk/`*filesystem_name*

5.5.1 Format 1: Relative File Name

Use the first format (`quota=` *quota_file_relative_name*) if you want the quota control file to reside on the file system it controls. The file name is relative to the root directory of the file system and, if you use the default name as recommended (`$QFILE`, as defined in `quadmin`(8)), the option would be written as follows:

```
quota=$QFILE
```

By default, the special name `$QFILE` maps to the `.Quota60` file in the root directory of the file system.

5.5.2 Format 2: Absolute File Name

Use the second format (`quota=` *quota_file_full_name*) if you want the quota control files to reside in a directory other than the root directory of the file system it controls. For example, if the quota files were to reside in the `/etc/admin/quota70` directory, the options field of the `/etc/fstab` file would contain the following line:

```
quota=/etc/admin/quota70/$FILESYS
```

The special name `$FILESYS` is the last component of the file system name on this line in the `/etc/fstab` file. For example, the `/etc/fstab` file contains the following line:

```
/dev/dsk/slash_b    /b    C1FS    quota=/etc/admin/quota70/$FILESYS
```

This line would resolve to the following:

```
quota=/etc/admin/quota70/slash_b
```

A directory, `quota70`, holds all quota control files. The file system name is used to identify each individual quota control file within the directory.

5.5.3 Format 3: Quota Control Groups

Use the third format (`quota=/dev/dsk/`*filesystem_name*) to show that the specified file system is under the control of a quota file defined and used to control other file systems as well. You should use this format when multiple file systems will be controlled as a group. (Multiple `tmp` file systems can be handled this way so that the user's `tmp` quota is independent of which or how many `tmp` file systems might be in use.)

For example, assume that three lines from the `/etc/fstab` file were written as follows:

```
/dev/dsk/tmp_1     /tmp_1     C1FS     quota=$QFILE
/dev/dsk/tmp_2     /tmp_2     C1FS     quota=/dev/dsk/tmp_1
/dev/dsk/tmp_3     /tmp_3     C1FS     quota=/dev/dsk/tmp_1
```

These lines define the quota control file as `.Quota60`, residing in the root directory of `/tmp_1`. The `/tmp_2` and `/tmp_3` file systems are controlled by the same quota control file; therefore, the quota information for usage of any or all of the three file systems is common and reflects the combined usage of all three.

Note: If the right-hand side of a quota option matches one of the other file system names in the `/etc/fstab` file, it is a declaration in the third format, and the specified file system must contain a quota option naming a file. Only one level of indirection is supported.

5.6 Determining Defaults and Special Users

For each file system for which you intend to enforce quotas, you must choose between various quota enforcement options, as follows:

1. Select the class or classes of enforcement you want to use (a combination of user, group, and account quotas).
2. For each of those classes, decide whether to enforce inode quota limits, file quota limits, or both.
3. Pick default values for inode and file limits and their corresponding warning values that will work for most users on the file system.
4. Decide if any special users require limits different from the default values, and decide what those values should be.

5. If you want to use oversubscription, pick one of the evaluation algorithms and an evaluation period. You can perform this task later if you do not want to make a decision now.

All of these choices depend heavily on how the file system is used by users, how many users use the system, and so on; there are no easy rules for making those decisions. However, the `qudu`(8) command allows you to observe current usage on the file system, which helps you choose proper values. After you have picked the values, read Section 5.7, page 246, which shows you how to produce a quota source file for each file system using these values.

The following examples show how to collect both file and inode usage data for the hypothetical file system `/dev/dsk/netos`.

In example 1, the command line prints a list of all user IDs, group IDs, and account IDs that currently have files allocated on the `/dev/dsk/netos` file system. The list is sorted in ascending order, first by ID class and second by number of inodes in use.

Example 1:

```
/etc/qudu /dev/dsk/netos | cut -d’ ’ -f1-5 | sort +0 -1 +4n
```

In example 2, you want to enforce file quotas only for user IDs. The command line prints a list of user IDs on the `/dev/dsk/netos` file system, sorted into descending order based on the number of disk blocks currently in use.

Example 2:

```
/etc/qudu /dev/dsk/netos | grep uid | cut -d’ ’ -f1,2,6-8 | sort 4nr
```

5.7 Creating a Quota Control Source File

When a UNICOS kernel has been built with quota enforcement enabled, you must create quota control files for each file system on which you want to enforce quotas. The simplest way to enforce quotas is to place the file in the controlled file system, as described in this section. Other methods are discussed in Section 5.10, page 253. To create a quota file, the information needed in the file must be expressed in `quadmin` directive format (see the `quadmin`(8) man page). You can make a file in the correct format by using `qudu`(8), or you can create your own `quadmin` source file. See Section 5.7.1, page 247, and Section 5.7.2, page 250, for more detailed information on this process.

When you initially create the source file, you must use the `quadmin` command with the `-F` option. The name of the quota control source file is determined from the `/etc/fstab` line belonging to the file system specified on the `filesystem` directive in the `quadmin` input file.

It is not recommended that you create a very elaborate quota scheme the first time. Using appropriate defaults generates quota files almost automatically, and you can always adjust specific quotas later if the need arises.

Note: The file system that is specified and the file system that has the quota control file must be mounted.

5.7.1 The qudu(8) Method of Source Creation

The `qudu`(8) command looks at each inode in a file system and accumulates inode counts and total file size for every account ID, group ID, and user ID found. This information is written in a form suitable for input to `quadmin`(8), which creates or updates the quota control file. When you have run `qudu`, the output file contains usage information for every ID currently using space on the file system you selected. Run `qudu` on one file system and examine the output to get an idea of its appearance and the information it contains. (It takes a few minutes to run this command on a large file system.)

If you do not want usage information for all account, group, and user IDs, select the specific class or classes of ID you want with the `-A` (account), `-G` (group), or `-U` (user) options. If none of these options is present, the default is the same as if `-AGU` had been included. For example, if you want to control only group and user IDs, specify `-GU`.

If you use the resulting output file as input to `quadmin`, a quota control file is created with an entry for every ID in use. The usage information is current; if users are running on the file system, the information becomes obsolete quickly, but every entry has default quotas and warnings assigned to it. If the defaults are satisfactory, you are finished; otherwise, you must change whatever is necessary to get to the state you want.

5.7.1.1 Changing Defaults

If default values are suitable for all IDs using the file system, but the defaults defined in the `sys/quota.h` file are not correct for your site, edit the output from `qudu` and add the correct defaults to the file by using the `quadmin default` directive (see the `quadmin`(8) man page for a description of the directive format). This information should be placed after the `filesystem` and `open` directives and the first `acid`, `gid`, or `uid` directive line in the file. If you use this file as input to `quadmin`, the defaults reflect the values you specified and are applied to every entry in the file.

The following example shows how to use the output file from qudu to change default values:

```
#    Usage report by qudu: (SN-1203) on Thu Feb 15 09:36:05 CST 1990
version 6
filesystem dsk/usr_c /usr/c/$QFILE; open dsk/usr_c
remove all usage

#    The following lines were inserted to change the defaults

     default account file quota 8500 inode quota 700
     default group file quota 10000 inode quota unlimited
     default user file quota 2500 inode quota 300

#    The remainder of the file is exactly as written by qudu

uid 0 inode usage 1375 file usage 25721
gid 0 inode usage 1401 file usage 26165
acid 0 inode usage 7114 file usage 74981
gid 1 inode usage 2 file usage 71
uid 2 inode usage 1 file usage 71
gid 3 inode usage 1 file usage 32
acid 2 inode usage 168 file usage 0
uid 233 inode usage 361 file usage 2500
uid 247 inode usage 1 file usage 0
uid 258 inode usage 12 file usage 2
uid 263 inode usage 334 file usage 5201
uid 264 inode usage 29 file usage 937
uid 269 inode usage 505 file usage 6893
uid 273 inode usage 23 file usage 6548
uid 283 inode usage 1 file usage 0
uid 285 inode usage 29 file usage 0
```

Setting defaults needs to be done only once for each quota control file, because the defaults persist for the life of the file, unless you change them.

Note: You must apply infinite quotas to all account IDs, group IDs, and user IDs used by system daemons, root, and other special users. These types of special users should not receive write errors because a quota limit has been reached. Infinite quotas for special users is not part of the software design because the IDs used cannot be predicted. See Section 5.7.1.1, page 247.

5.7.1.2 Setting Specific Quotas

If some IDs in the quota control file need quota values different from the default, follow these steps:

1. Decide on suitable defaults and follow the procedure recommended in the previous section.
2. Using an editor, find the IDs in the file created by `qudu` that need special attention (they appear in ascending numeric order) and insert appropriate file quota, file warning, inode quota, and inode warning values for the relevant ID class (account ID, group ID, or user ID). If there are only a few IDs that need changing, this process does not take much time. If many IDs require different quotas, this process takes some time, but it is possible to automate the insertion of quota information with a stream editor or a program.

The following example shows how to change quota values for specific IDs:

```
#    Usage report by qudu: (SN-1203) on Thu Feb 15 09:36:05 CST 1990
version 6
filesystem dsk/usr_c /usr/c/$QFILE; open_dsk/usr c
remove all usage

#    The following lines were inserted to change the defaults

     default account file quota 8500 inode quota 700
     default group file quota 10000 inode quota unlimited
     default user file quota 2500 inode quota 300

#    Change the quotas for uid, gid, and acid 0 to unlimited

     acid 0 file quota unlimited inode quota unlimited
     gid 0 file quota unlimited inode quota unlimited
     uid 0 file quota unlimited inode quota unlimited

#    The remainder of the file is exactly as written by qudu

uid 0 inode usage 1375 file usage 25721
gid 0 inode usage 1401 file usage 26165
acid 0 inode usage 7114 file usage 74981
gid 1 inode usage 2 file usage 71
uid 2 inode usage 1 file usage 71

acid 2 inode usage 168 file usage 0
```

```
gid 3 inode usage 1 file usage 32
uid 233 inode usage 361 file usage 2500
uid 247 inode usage 1 file usage 0
uid 258 inode usage 12 file usage 2
uid 263 inode usage 334 file usage 5201
uid 264 inode usage 29 file usage 937
uid 269 inode usage 505 file usage 6893
uid 273 inode usage 23 file usage 6548
uid 283 inode usage 1 file usage 0
uid 285 inode usage 29 file usage 0
```

Setting specific quotas needs to be done only once for each quota control file, because the quota settings persist for the life of the file, unless you change them. If you have done much manual work, create an ASCII back-up copy of the quota control file by executing `quota` with the `-b` option.

5.7.2 Manual Source File Creation

You can create a source file for `quadmin`(8) manually, as shown in the example found on the `quadmin`(8) man page. If your installation uses ranges of IDs for specific categories of accounts, groups, or users, this method allows you to create a file fairly easily using the `enable` directive provided by `quadmin`. If each ID has a different quota, you must create directives one at a time. To make this process easier, you can create a skeleton directive file with all of the general information and then edit that file to insert specific quota information. You can always create a source file with a program, if the information needed to specify the quotas for each user is available or can be derived from an existing source.

5.8 Generating the Quota Control File

When the source file has been created, you must generate a quota control file. Use `quadmin` with the `-F` option for this step, because the quota file does not exist and therefore cannot be accessed through the `quotactl`(2) system call. The `-F` option causes `quadmin` to write the file directly.

Follow these steps to generate the file:

1. If the source file has been created initially by `qudu`, and `/etc/fstab` has the current quota configuration, the quota file name on the `filesystem` directive line in the source file should be correct. If it is not correct, choose one of the following methods to rename the quota file:

- Ensure that `/etc/fstab` is correct and remove the file name from the `filesystem` directive. This forces `quadmin` to use the file system name specified in `/etc/fstab`.
- Do not have this file system configured in `/etc/fstab`. This forces `quadmin` to use the file system name specified in the `filesystem` directive.
- Change the file system name on the `filesystem` directive line to match the one in `/etc/fstab`. In this case, `quadmin` uses the name from `/etc/fstab`, but, because the name matches the one on the `filesystem` directive, `quadmin` does not warn you about a name mismatch.

Assuming that `/etc/fstab` is set up, the directive line for creating a quota control file in `/usr/c`would read as follows:

```
filesystem /usr/c; open /usr/c
```

Always use an explicit `open` directive so that, if the source files are ever combined, there will be no confusion about what information belongs to each file system.

2. With the proper information in the source file, run the following command:

   ```
   quadmin -F -m source_file_name
   ```

   This command creates the quota control file specified either in `/etc/fstab` or with the optional second parameter of the `filesystem` directive. You must have permission to create files in the directory specified for the quota control file.

3. When you are finished, ensure that the quota file has `root` ownership and owner-only permissions, so other users cannot access or accidentally alter or remove it. `quadmin` creates the file with whatever ownership it has inherited and owner read / write access only.

It is recommended that you create each quota control file separately in order to deal with any mistakes or problems more easily. Also, having a quota source file for each file system makes maintenance easier.

Quotas applied to the root file system or to user ID `0` (`root`) are ignored. The kernel refuses to activate quotas on this file system. Be especially cautious that quota control files do not get removed by accident or inadvertently reloaded from a backup.

5.9 Activating Quota Enforcement

After a quota control file has been created, you can activate quota enforcement on that file system by entering the `quadmin`(8) command with the `-c` option (you must be super user to activate the quota feature; for use on a Cray ML-Safe configuration of the UNICOS system, see the `quadmin`(8) manual page for more information). To activate quota enforcement, you must specify one of the following activation choices as an argument to the `-c` option:

Activation choice	Description
`count`	Turns on quotas for file system; maintains counts.
`default`	Turns on quotas for the relevant file system in the mode it was previously in. The kernel records the activation mode and uses the most recent mode when `default` is used. Unless explicitly changed by use of a `default` directive, a newly created quota file has `count` as the default mode.
`inform`	Turns on quotas for the relevant file system, maintains counts, and issues warning and quota limit signals.
`enforce`	Turns on quotas for the relevant file system, maintains counts, issues warning and quota limit signals, and enforces quota limits.

These choices are described in more detail on the `quotactl`(2) man page.

The following examples use `enforce`, but the other activation choices can be used in the identical manner. The easiest way to activate quota enforcement is to use the quota source file you used to create the quota file. The information in the quota control file has not been changed; `quadmin` searches for the `filesystem` directive in the source file and gets the needed information from it. Run the following command:

```
quadmin -c enforce source_file_name
```

If you do not want to use the directive file, run the following command to activate quota enforcement on the `/usr/c` file system:

```
quadmin -c enforce -s /usr/c
```

If there are no error messages, quota enforcement is running on the file system. You may change the enforcement level at any time.

Quotas may be activated automatically when the file system is mounted if the `-q` option is added to the `mount`(8) command line. The enforcement level is `default`.

5.10 Setting Current Usage Information

The `qudu`(8) command generates usage information based on the content of the target file system at the time it is run. Because `qudu` uses the raw device interface, you can run usage extraction on an unmounted device. If the quota control file resides on the controlled file system as recommended, the usage information cannot be entered in the file until the device is mounted. `quadmin`(8) should be run immediately after the device is mounted in order for the quota control file usage information to be as accurate as possible.

Usage information in the quota control file must be set initially when quota control is installed on the file system and whenever `fsck`(8) changes information related to usage. (You will not know when `fsck` changes information, so run `qudu` routinely after `fsck`.) Inaccurate usage information in the quota control file does not affect system processes, but quota limit and warning thresholds are dependent upon the accuracy of the quota control file. To ensure that quota control is consistent and accurate, make it a policy to update usage information before users can alter the file system.

5.11 Usage Accumulation Rules

Assuming that both inode and file space are controlled, the following sections describe how the kernel accounts for space usage.

5.11.1 Inode Usage

Only inodes associated with regular files (`IFREG`), migrated files (`IFOFL`, `IFOFD`), symbolic links, and directories (`IFDIR`) are counted. The account, group, and user IDs found in the inode are charged one unit for each inode belonging to them. Inodes are not counted as file space.

5.11.2 File Usage

Any indirect blocks (blocks containing disk allocation information not accessible to the user) are counted as file space, as are access control limit (ACL) blocks, if the UNICOS multilevel security (MLS) feature is enabled.

Files and directories whose data resides entirely in the inode (possible only on Cray T90, and Cray C90 systems) have an allocated block count of 0. Only actual blocks allocated are counted, rather than the space logically used. Therefore, the length reported by the `ls`(1) or `du`(1) commands can be very different from the amount of space really used (either more or less), especially if sparse allocation occurs or files have data residing in the inode. The amount charged is always in units of allocation rather than logical length, which varies depending on the implementation and device configuration from one sector to many tracks per unit.

When files are migrated, the space occupied is credited to the IDs in the inode of the file. If the file is later brought back online, the space needed for the file is applied against the quotas. If bringing a file online violates an enforced quota limit, that file cannot be brought online.

5.12 Administering the Quota Enforcement Feature

The following sections contain information about basic administration of the quota feature, including starting up the system, activating quota enforcement control, adding users, deleting users, creating or extending files, and viewing network quota information.

5.12.1 System Startup

If you run `fsck`(8), also run `qudu`(8) to generate correct usage information. Immediately after the file system is mounted, run `quadmin`(8) to correct the quota control file.

The following example shows how the file system quota feature can be activated at system startup. The first box contains a script from the `/etc/rc.mid` file, which is called by the system start-up script file `etc/rc`. (The script can be placed in any file called by the system start-up script.) In this example, the script uses the shell script `/admin/etc/quotas/qurun` (shown in the second box) to create and update each file system's quota control file, start the file system quota enforcement feature, and set the enforcement level to `enforce`.

```
#   initialize quotas for the day
#
               if [ -f /admin/etc/quotas/qurun ]
               then
                  echo "Rebuilding disk quotas... (could take several minutes)..."
                  /admin/etc/quotas/qurun -i
                  echo "Disk Quotas rebuilt"
               fi
#
#   turn quotas on and run the exceptions files
#
/admin/etc/quotas/qurun -s enforce
/admin/etc/quotas/qurun -d
```

In this example, the `/admin/etc/quotas/qurun` shell script is used to automate routine administrative duties for each file system specified on the shell script's command line. If no file systems are specified, the list `$FILESYSTEMS` is used. The options that can be specified on the command line are as follows:

Option	Description
`-b`	Backup. For each file system, this option creates a back-up file in directory `$BACKDIR` containing all the `quadmin` directives necessary to reconstruct the file system's current quota file, excluding usages. The quotas feature can be running, and users can be in the file system during a backup.
`-i`	Install. For each file system, this option creates a new quota control file by using the defaults in the back-up file (if it exists) in directory `$BACKDIR`. The `qudu` command is then used to update the quota file with the current usages. This option is useful when setting up quotas for the first time, or when the quota file has been destroyed or is out-of-date. The quota feature cannot be running during an install, and users cannot be in the file system.
`-d`	Defaults. For each file system, this option updates the quota control file with any defaults from the back-up file in directory `$BACKDIR`. This option is useful when you have made changes to the back-up file and you want to activate those changes in the file system. The quotas feature can be running, and users can be in the file system when the quota control file is updated.
`-r`	Report. For each file system, this option issues a report showing the number of disk blocks and inodes owned by each user, group, and account, sorted by increasing usage amount. This option is useful when picking defaults or when looking for users who have exceeded their limits. The quotas feature can be running and users can be in the file system while a report is issued. The `-r` option requires that the `awk` script `qusort` be in your binary search path. (The `qusort` script is intended to provide a

mechanism for formatting the `qurun` report; it is not provided in the release or in this example.)

`-s` Start. For each file system, this option starts the quotas feature or, if the feature is already running, it changes the enforcement level. The available levels are `count`, `inform`, and `enforce`. If you do not specify a level with `-s`, the `enforce` level is used. To preserve accuracy, you must make sure no users are in the file system when turning on the quotas feature. Users can be in the file system when changing enforcement levels.

`-u` Update. For each file system, this option uses the `qudu` command to update the quota file with the current usages. This option is primarily used while the quota feature is running to update the quota file when it is believed that the quota file is out-of-date. The update option improves the accuracy of the usages, but it does not guarantee absolute accuracy.

The following box contains the shell script `/admin/etc/quotas/qurun`.

```
#
#Sample code for the shell script /admin/etc/quotas/qurun:
#

FILESYSTEMS="/sn1001/soft/os /sn1001/cnn /sn1001/mktg"  # default list of
filesystems to act on
BACK51=/admin/etc/quotas/backup    # directory for 5.1 backup files
BACK60=/admin/etc/quotas/backup60  # directory for 6.0 backup files
QUOTDIR=                           # directory for quota files

BIN=/usr/bin/   # directory where the quota commands reside
ETC=/etc/       # directory for qudu and quadmin

#
#       Determine the operating system level.  Use it to pick the correct
#       backup directory.
#

${ETC}quadmin -Q >/dev/null 2>&1
if [ $? -eq 0 ]; then           # if at 6.0 or greater
        QFILE=.Quota60
        BACKDIR=$BACK60
else
        QFILE=.Quota51
        BACKDIR=$BACK51
fi

flags=0
set -- ‘getopt ’bdhirsu’ $*‘
if [ $? -eq 0 ]; then
    while [ "$1" != "--" ]
    do
        case $1 in
        -b)
            command=Backup;;
        -d)
            command=Defaults;;
        -i)
            command=Install;;
        -r)
            command=Report;;
        -s)
            command=Start;;
        -u)
            command=Update;;
        esac
        shift
        flags=‘expr $flags + 1‘
    done
    shift
fi

if [ $flags -ne 1 ]; then
    cat >&2<EOF
Usage: ‘basename $0‘ [ -b | -i | -d | -r | -s [count|inform|enforce] |
-u] [filesys ...] ‘basename $0‘ is used to administer the File Quotas
feature.
EOF
    exit 1
fi

if [ "$command" = "Start" ]; then   # if Start, check for enforcement level
    level=enforce
    case $1 in
    count|inform|enforce)
        level=$1
        shift
    esac
fi

if [ $# -ne 0 ]; then       # if no filesystems specified, use default list
    FILESYSTEMS=$*
fi

for filesys in $FILESYSTEMS
do
    echo $filesys
    DEVDSK=‘/etc/mount | awk ’$3 == "’$filesys’" {print $1}’‘
    if [ "$DEVDSK" = "" ]; then
        echo "$filesys is not mounted!  $command not done."
        continue
    fi
#
#   Determine if quotas is running on the filesystem.  Sometimes it matters.
#
    QUOFF=F                                 # assume quotas not running
    ${BIN}quota -s $filesys 1>/dev/null 2>&1
    if [ $? -eq 0 ];then
        QUOFF=                              # quotas is running
    fi
#
#   Compute the backup file and quota file names for this filesystem.
#
    bakfile=$BACKDIR/‘basename $filesys‘
    if [ "$QUOTDIR" ]; then
        quofile=$QUOTDIR/‘basename $filesys‘
    else
        quofile=$filesys/$QFILE
    fi

#   Perform the requested subfunction.
#
    case $command in
    Backup)
        rm -f $bakfile
        if [ ! -r $quofile ]; then
            echo "Quota file $quofile not found.  Will create it now."
            $0 -u $filesys
        fi
        ( ${BIN}quota -bEs $filesys -q $quofile > $bakfile ) &
        ;;
    Defaults)
        if [ -r $bakfile ]; then
            ( ${ETC}quadmin -m$QUOFF $bakfile ) &
            echo ${ETC}quadmin -m$QUOFF $bakfile
        else
            echo "File $bakfile is unreadable; Defaults are unchanged.">&2
        fi
        ;;
    Install)
        if [ "$QUOFF" ]; then
            rm -f $quofile
            $0 -d $filesys
            ( ${ETC}qudu -q $quofile $DEVDSK | ${ETC}quadmin -mF ) &
       else
            echo "$filesys has quotas enabled!  Install not done."
        fi
        ;;
    Report)
        echo "                        filesystem $filesys"
        ${ETC}qudu -U $DEVDSK | /bin/grep uid | /bin/sort -r -n +7
        echo ""
        ;;
    Start)
        if [ ! -r $bakfile ]; then
            echo "Backup file $bakfile not found.  Will create it now."
            $0 -b $filesys
        fi
        ${ETC}quadmin -c $level $bakfile
       ;;
    Update)
        ( ${ETC}qudu -q $quofile $DEVDSK | ${ETC}quadmin -m$QUOFF ) &
       ;;
    esac
done
wait
```

If you decide to use the example script, you must first set the first three variables to match your configuration. The variables are as follows:

Variable	Description
FILESYSTEMS	Set to the list of file systems on which you intend to run quotas. The file system names should be separated by blanks.
BACKDIR	Set to the path name of a directory where back-up copies of each quota file will be kept. The back-up files contain all the default quota settings and all settings for users whose limits are different than the defaults.
QUOTDIR	Usually, leave this blank, which causes the quota file for each file system to be placed in the root directory of that file system. If you are short of disk space in the quota file systems, set QUOTDIR to the path name of a directory on a file system where there is more room and where all the quota files are kept. If you install quotas and then later decide to change QUOTDIR, make sure that you place the new quota file path names in the filesys directive of each of your back-up files.

5.12.1.1 Back-up File Example

The following box contains an example of a back-up file. The back-up file is located in directory `$BACKDIR` and is created by `/admin/etc/quotas/qurun` option `-b`. The back-up file is used by options `-i` (install) and `-d` (defaults).

```
# Created by quota on Thu Aug 31 11:54:48 1989
filesystem /sn1001/soft/os /sn1001/soft/os/.Quota60; open /sn1001/soft/os
default account flags off
default account file quota 40000 file warning 0.900000
default account inode quota 200 inode warning 0.900000
default group flags off
default group file quota 40000 file warning 0.900000
default group inode quota 200 inode warning 0.900000
default user flags fi
default user file quota 4000 file warning 0.900000
default user inode quota 1000 inode warning 0.900000
user root file quota infinite inode quota infinite
```

5.12.2 Quota Enforcement Control

When a file system is mounted by using `mount`(8), quota control is activated in its default mode. If a `default level` directive is not included when the quota control file is created, the default mode is `count`. The enforcement mode can be changed by using the `quadmin -c` option, and the specified mode then becomes the default mode. After quota control is activated, you can change its mode, but you cannot deactivate it; only unmounting the file system by using `umount` (see `mount`(8)) deactivates quota control.

5.12.3 Adding Users

If a new user of the file system becomes active, the kernel automatically creates quota control entries with default quota and warning values for any IDs not already defined in the quota control file. If the default values are improper, you must run `quadmin` when you create the new IDs to set up the correct quota information on all file systems available to those IDs. When you run `qudu` to set up initial usage information, `quadmin` adds records to the quota control file with default quota and warning values for all IDs found on the file system, whether or not they were defined in the existing quota control file.

5.12.4 Deleting Users

You can use `quadmin` to delete entries from the existing quota control file. When an entry is deleted, its usage value is set to `0` and the limit and warning values are set to their defaults; the entry is not removed.

To remove an entry from the quota control file, use the following steps:

1. Create the ASCII version of the quota control file by using the `quota` command.
2. Using an editor, delete all records corresponding to the deleted entry.
3. Create a new quota control file by using the `quadmin` command.

   Note: If any inodes with the deleted or removed ID exist in the file system, those entries are restored (with default limit and warning values) if the output of `qudu` is processed by `quadmin` without any editing.

5.12.5 Creating or Extending Files

One of the following rules is applied when files are being created or extended. Both inode and file space quotas are dealt with at the same time (assuming both are being enforced). The privilege of the process making the request is not an issue, so processes owned by root are constrained by the same rules as others.

- If the file is owned by root, quotas are not enforced, and the file can be extended without limit.
- If the file is not owned by root, the owner's quotas are enforced.

When ownership of a file changes, one of the previous rules is followed, depending on the target's ID.

5.12.6 Viewing Quota Control

The quadmin(8) command executed with the -v option displays generic quota information and the enforcement level of each file system with active quota enforcement. The following example shows a typical display created by quadmin -v:

```
Quota Generic Information on Fri Feb  9 10:17:51 CST 1990
Quota control configured for: Accounts, Groups, Users
Quota entries configured:        375
        Maximum used:             78, Current:       36
Quota entry accesses: get     487532,    put     487294
Quota file accesses: read      54219, update       1004
Quota hash chain:    read         34
Inode cache flushes:               0,  sleep          0

/c.......       INFORM     /d.......COUNT    /fsmtest..ENFORCE
```

This display created by quadmin shows that the kernel is configured to support account, group, and user quotas and has a total of 375 quota entries configured. The Maximum used and Current fields show the greatest number of quota entries used at one time and the current number in use. The maximum value is intended to help you decide how large the quota table in the kernel must be. If the maximum is far below the number of configured entries and stays that way over a period of time when you know typical workloads are being run, you may reduce the number configured, thereby reducing the size of the kernel.

The number of hash chain reads indicates how many quota records were found other than at the beginning of a hash chain. This number is not important in most situations, because it is mostly an indication of the hashing mechanism's efficiency relative to the particular mix of IDs being used.

Usually, tuning the configuration too close to the minimum is not advisable. Individual quota table entries are not very large; reducing the total number by a small percentage does not result in any significant benefit. If you see consistent behavior such as that indicated in the example display (where 375 entries are configured and the maximum usage is only 78), and it would be advantageous for your kernel size to be reduced slightly, you can safely lower the number of configured quota entries to 150.

The remainder of the statistics portion of the display shows information that is useful only in extreme situations. The `get` and `put` values show how many times quota table entries were read and changed. The `read` and `update` counts show the actual number of disk accesses needed to retrieve new quota entries and to keep the file current.

The ratio between `get` and `read` or between `update` and `put` indicates how beneficial cache is in reducing actual I/O activity caused by quota enforcement. If any I/O errors occur on the quota control files, an error count also appears in the display.

The last statistics line shows the number of times the quota system was required to flush inodes and/or sleep in order to acquire a quota table entry for a newly opened file. These numbers should normally be small; significant increases indicate an insufficient number of quota entries. If it persists, this condition can reduce system performance.

The last line on the display lists the file systems and their level of quota enforcement. Only file systems with some level of quota enforcement are shown, and the format is adjusted to fit the maximum number of names on a line consistent with an attractive and readable format.

5.13 Additions to Login Profile

If you want automatic reporting of quota warning and limit conditions when the user logs in or a batch job starts, place the `quota`(1) command in the login profile (`.login` or `/etc/cshrc` (see `cshrc`(5)) for `csh`(1) and `.profile` or `/etc/profile` (see `profile`(5)) for `ksh`(1)), as follows:

```
quota -r wl
```

This command line reports the names of any file systems where the user ID, default account ID, or default group ID have usage values above the warning level. If you want to check all users' authorized account IDs and group IDs, add the following command to the login profile:

```
quota -A -G -r wl
```

If you only want to report on the user's home file system, add the `-s` option to `quota`, specifying the path of the home file system.

Because `quota` provides special exit codes for specific conditions, you can write scripts to analyze the information and produce more informative messages, provide information to local log files, inform the administrator, or perform special actions if batch jobs are involved. Refer to the `quota`(1) man page for more information.

5.14 User-level Behavior

User warning and limit messages are automatically written to the `stderr` file by the Korn, Bourne, and C login shells. However, if you are not using any of these shells, and if you want to enable user warning and limit messages, you must start one copy of `quotamon`(1) for each session (interactive and batch). Without it, users are not notified of quota warning conditions unless they use `quota`(1) periodically. Before using `quotamon`, decide whether the cost of running this background process is acceptable in your environment; `quotamon` is small and uses no CPU time except when writing messages.

If you are running a Cray ML-Safe configuration of the UNICOS system, prior to changing your active security label you must kill `quotamon`, because it is a background process. After changing your label, you can restart `quotamon`.

Starting `quotamon` is best done by adding code to `/etc/profile` and `/etc/cshrc`, as in the following:

```
ps | grep quotamon > /dev/null
if ($status == 1) then
   /usr/bin/quotamon -s 60
endif
```

The `-s` option specifies the amount of time in seconds to delay repetitions of the same kind of quota message; the default is 45 seconds. You may specify as large a value as you wish (up to the signed integer maximum).

The `quotamon` process can be killed by the user. This capability allows users to remove the process if they object to its presence.

If a program traps a `SIGINFO` signal, you can use the `getinfo`(2) system call to determine what event occurred. If a `SIGINFO` occurs, the signal is sent to all processes attached to the same job table entry that have not cleared a previous signal occurrence by executing a `getinfo` system call. The signal is ignored by default.

5.15 Nonstandard Configuration Options

This section discusses alternative ways to configure quota control files; the recommended configuration provides for one quota control file per file system, with each such file residing on the file system it controls (all configuration options are provided through the `/etc/fstab` file).

The kernel can activate quota control where the quota control file is on another mounted file system and also allows more than one file system to be controlled by the same quota control file. However, you must understand the consequences of nonstandard configurations before using them.

5.15.1 Nonresident Quota Control Files

The major operational complications resulting from configuring a quota control file that does not reside on the file system it controls are as follows:

- The file system on which the quota control file resides must be mounted before quotas can be activated on the controlled file system. Conversely, you must unmount the controlled file system before unmounting the file system on which the quota control file resides.
- You should place the configuration information in `/etc/fstab` to ensure that quota usage is consistent. You can use the `-Q` option with the `mount`(8) command to perform this function, but this is not recommended.
- Performance may be affected by too much quota control file traffic on a single file system.
- Quota control would be lost on the controlled file system (but the system would continue to perform I/O correctly) if the control file became inaccessible because of some malfunction on its file system.

5.15.2 File System Groups under One Quota Control File

File system groups under control of one quota control file (also known as *quota groups*, *domain quotas*, or *quota domains*) are similar in operational complexity to the configuration described in the previous section. The key to successful use of this feature is placing the configuration information in the `/etc/fstab` file so that all administrative commands have a consistent view of the organization.

Usage information generated by `qudu`(8) consists of the sum of usage for all IDs present on all the file systems that are members of the quota group when any one file system is specified on the command line.

5.16 Aggregate Quotas

The aggregate quotas option enables sites to charge for off-line files in much the same way as they charge for online files. In the default quotas setting, an attempt to migrate a file could fail because bringing the file online might violate a quota limit. With the aggregate quotas setting, file unmigration is always allowed by the quota mechanism. The UNICOS operating system tracks off-line and online data together, and users are limited in the total amount of space they can use.

Using aggregate quotas has the following ramifications:

- Administrators can limit (approximately) the amount of space used in the off-line archive.
- Administrators have no quota control over what users have online; they can control only the total space used.
- Off-line file data blocks are always charged in units of 4096-byte blocks; therefore, the quota charge for an off-line file is always less than or equal to the charge for the same file online, depending on how the file was created or how the file system is configured.
- Users can accidentally unmigrate more data than the file system can hold.
- Users are always charged for data blocks and MLS blocks, whether or not the file is migrated.

Aggregate quotas affect the following commands:

`qudu`(8)	The `-a` option must be used to specify to use the aggregate quota counting method.
`quadmin`(8)	The default style directive must be changed from `online` to `aggregate`. This directive specifies whether quota counts are maintained only for files that are physically on disk (online), or whether files migrated off line by the Data Migration Facility (DMF) are included in the count (aggregate).

`quota`(1)	The quotas report column *File blocks* is changed to *Aggregate blocks*. This column shows the quota block counts and block usage for aggregate quotas.

5.17 Using the Oversubscription Option (Soft Quota)

The normal operation of the file quota mechanism is to issue warning messages to the user when the warning level is exceeded and to stop any new allocation if the quota is reached. In addition to this mode of operation, an oversubscription mechanism is also available on a file system basis. This mechanism lets users exceed quotas by a controlled amount for a limited period of time.

5.17.1 Behavior of the Oversubscription Mechanism

Each quota control file has header fields that select the oversubscription algorithm to be used and specify parameters to the algorithm. This means that all account IDs, group IDs, and user IDs controlled from the file are under a single discipline. Oversubscription is available for file space only; inode use is enforced with the standard mechanism. When you select this mode of operation by setting the algorithm selector in a header field, quota control changes its behavior as follows:

- The quota remains the hard upper boundary on allocation.
- A second field in the quota record, `f_runquota`, becomes the enforced quota value. When this field is nonzero, the kernel enforces at this value rather than at the `f_quota` allocation limit.
- The warning level becomes the oversubscription threshold.
- The `quota`(1) command changes its display to show information related to the soft quota, such as the time when new allocation is prohibited or permitted.
- The `quadmin`(8) command supports additional fields needed by the evaluation algorithms in the header and record structures.

5.17.2 Supported Algorithms

Two oversubscription algorithms, an exponential and a linear function, are supplied, and two additional algorithm selection values are reserved for site use.

The default algorithm (named `none`) realizes the original behavior of file quotas. Sites that have access to source can add local algorithms (named `site1` and `site2`) to the kernel and their inverses to the `quota`(1) command. The `quota`(1) command uses the inverse to provide predictive information to the user.

5.17.2.1 Exponential Algorithm

The exponential algorithm, as follows, is based on the COS RDM oversubscription model.

$$A = U_s + (A_s - U_s)e^{\frac{-(t-s)}{P}}$$

where:

A = Average now
A_s = Average at time s
P = Characteristic period (in seconds)
s = Time at A sub s
t = Time now
U_s = Usage at time s

In Table 32, the second column is the name used by the `quadmin`(8) command to set the field. Note that the implementation is such that if usage changes from one calculation period to the next, $U\ s$ is set to current usage, s is set to the current time, and $A\ s$ is set to A. This prevents usage changes from affecting the average until at least some history of that usage level can be accumulated.

Table 32. Field Usage of the Exponential Algorithm

Location	Field name	Variable	Description
Header	`algorithm`		Exponential (actual value)
Header	`ef1`	P	Characteristic period in seconds
Header	`ef2`		Not used
Record	`ef1`	A_s	Average at time s

Location	Field name	Variable	Description
Record	`ef2`	U_s	Usage at time *s*
Record	`ef3`		Not used
Record	`ef4`		Not used
Record	`ef5`	*s*	Prior evaluation time

5.17.2.2 Linear Algorithm

The linear algorithm is a linear form of oversubscription that eliminates the more time-consuming calculation of the exponential function.

$$A = U_s \frac{(t - s)}{P} + A_s \frac{P - (t - s)}{P}$$

where:

A = Average now
A_s = Average at time s
P = Characteristic period (in seconds)
s = Time at A sub s
t = Time now
U_s = Usage at time s

In Table 33, the second column is the name used by the `quadmin`(8) command to set the field. Note that the implementation is such that if usage changes from one calculation period to the next, U_s is set to current usage, s is set to the current time, and A_s is set to A. This prevents usage changes from affecting the average until at least some history of that usage level can be accumulated.

Table 33. Field Usage of the Linear Algorithm

Location	Field name	Variable	Description
Header	`algorithm`		Linear (actual value)
Header	`ef1`	*P*	Characteristic period in seconds
Header	`ef2`		Not used

Location	Field name	Variable	Description
Record	`ef1`	A_s	Average at time *s*
Record	`ef2`	U_s	Usage at time *s*
Record	`ef3`		Not used
Record	`ef4`		Not used
Record	`ef5`	s	Prior evaluation time

5.17.2.3 Algorithm Comparison

Both the exponential algorithm and the linear algorithm have similar long-term behavior, but the linear form generally reacts more quickly to drastic changes in usage. The inverse functions mentioned previously are used to predict when inflation or decay will cross the warning value (soft limit) and so answer the following questions:

- When will I be able to allocate more space if the average is above the soft limit but actual usage is below?
- When will I be prevented from allocating more space if the average is below the soft limit but actual usage is above?

5.18 Changing ID Class Control

If control over an ID class (account, group, or user) is added or deleted while quota enforcement is active on a file system, files whose inodes are in the cache before the change is made are not affected and retain their prior class enforcement state. This happens, for example, if a file system has been set up with `default account flags off` and then, while the system is mounted, a `default account flags fi quadmin` directive is issued. This behavior should be taken into account if you change the class enforcement policy on an active file system.

5.19 Quota Enforcement Across a Network

When dealing with the control of file space consumed by processes running as servers to another machine, quotas can be enforced only on the machine that physically owns the resource. This situation occurs because quota enforcement on a file system must be done in one and only one place, and the machine that owns the resource presumably has administrators most interested in its control.

Also, in heterogeneous networks, there is no guarantee that any node other than the one owning a resource really knows what it is, how it is allocated, and so forth.

A question you may have about this situation is how much information about the resource consumer is available to the enforcing node. To enforce quotas, the node must know the relevant IDs of the consumer, but in situations where IDs are mapped onto the node there is a potential for ambiguity. Also, UNICOS supports account IDs, but most network protocols, including NFS and OSI, do not have the capability of passing an account ID to the server.

File System Space Monitoring [6]

The file system space monitoring capability improves the usability and reliability of the system. Space monitoring observes the amount of free space on the mounted file systems and takes remedial action if warning or critical thresholds are reached.

This section describes the space monitoring feature, its installation, and use of the commands `fsdaemon`(8), the monitor daemon, and `fsmon`(8), the daemon interface.

6.1 Operation of the Space Monitor

There are three major components to the space monitor: the monitor daemon (`fsdaemon`), the daemon interface (`fsmon`), and the operator's interface (`msgdaemon`).

The daemon may be started by the `/etc/rc` script (see `brc`(8)) in the area of the script reserved for starting daemons, or it can be started manually by using the `fsdaemon` command. Once the daemon starts and configures its monitor tables, it begins monitoring file system free space on a timed cycle. It remains running until stopped by an operator command or a system shutdown.

If the operator wants to continuously examine the current status of the monitored file systems or to alter some aspect of the monitoring process, the operator enters a command through the `msgdaemon` operator interface command. The `fsmon` command must be defined in the list of valid operator commands in the configuration file `$HOME/.operrc`. The operator interface executes `fsmon` and the output is returned to the operator's display screen.

The daemon interface may also be called by a privileged user. In this case, it returns its output to the `stdout` file.

The purpose of the daemon is to take action when the free space on one or more monitored file systems reaches either a warning or a critical free space threshold. When this occurs, the daemon may perform the following actions based on its configuration instructions:

- Send a message to the operator through `msgdaemon`(8)
- Initiate a shell script or command

6.2 Interprocess Communication

Communication between fsdaemon(8) and fsmon(8) occurs by way of named pipes. When it initializes, fsdaemon establishes a request pipe with a known name; all requests to it are made through that pipe.

When fsmon executes, it opens the fsdaemon request pipe and includes the name of the reply pipe along with the request it writes into the request pipe. fsdaemon replies to each request using the reply pipe name specified with the request. This allows multiple instances of fsmon to run without conflict.

Communication between fsdaemon and msgdaemon(8) is used only for warnings and critical operator messages. It works in much the same way as the fsdaemon to fsmon mechanism previously described. In this case, however, fsdaemon is the originator rather than the recipient of requests. An include file (msg.h) provided with the operator message daemon (msgdaemon) ensures that the interface is structured properly. The form of operator messages indicates that an operator response is expected even though the actual response is discarded. This is merely to ensure that operator messages are displayed and stay displayed until the operator responds or they are canceled by fsdaemon.

When an fsmon command is given to the operator interface, the parameters are passed on the command line. Output from fsmon is sent back to the operator interface through the stdout file. This interface is not interactive but allows fsmon to be run either as a normal or an operator command.

Communication between fsdaemon and administrator-specified commands or scripts occurs when fsdaemon detects that a warning or critical threshold has been reached for a particular file system. fsdaemon appends the name of the file system, preceded by a space, to the specified command or script, executes a fork(2) system call, and runs the command or script with ksh(1). The command or script is initiated with the daemon's environment (the stdin file is closed and the stdout file and stderr file are open on /dev/null) and, once started, fsdaemon pays no attention to the command and does not wait for it to complete. No other command or script is executed for the affected file system until a reset request has been received, or until a critical threshold has been reached.

6.3 The Monitor Daemon, fsdaemon(8)

This section describes the monitor daemon capabilities, both functionally and from a user's point of view.

Except when initializing and terminating, `fsdaemon`(8) always evaluates the current state of file systems and watches for requests from the interface program. The monitoring process is controlled by these parameters:

- Names of the file systems to monitor
- Warning and critical threshold values for each file system (default values are used if specific values are not provided)
- A Monitoring enable switch for each file system
- Operator message enable switches for warning and critical thresholds
- Critical and warning command execution flags
- A command (if any) to run if a warning or critical threshold occurs
- Length of time between each monitor cycle
- Priority (nice value) of the daemon
- `plock`(2) state of the daemon
- Ability to add monitored file systems at any time

6.3.1 File System Monitoring

When the daemon starts, it does not know which file systems you want it to monitor or the thresholds you want to assign to each file system. The file systems are identified and their thresholds set with the `fsmon` command. The following paragraphs describe the monitoring process after the daemon has been started and configured. See Section 6.3.3, page 277, for more information about configuring the daemon by using the `fsmon` command.

Once each cycle, `fsdaemon` collects usage information for each file system enabled in the monitor list and takes the following steps:

1. Each entry is examined to determine whether a threshold has been reached (a threshold is reached when usage is greater than or equal to the threshold). If not, the daemon waits for the next cycle time and repeats the process. If a threshold is reached, the next stage of processing is started.

2. If a new critical threshold has occurred and a critical command is not already running, the daemon starts the critical threshold command (if one has been defined using the `fsdaemon` command), issues the critical operator log messages for this file system (unless operator messages have been disabled with `fsmon -n`), marks that this entry has been processed,

and records the time the condition was detected. Then it moves to the next enabled entry.

3. If a new warning threshold has occurred, and a critical or warning command is not already running, the daemon starts the warning threshold command (if one has been defined), issues the warning operator log messages for this file system, marks that this entry has been processed, and records the time the condition was detected. Then it moves to the next enabled entry.

Once a file system has reached the warning or critical threshold, the messages have been logged, and the command started, the daemon does not attempt to start another command of the same kind until a reset request is processed.

6.3.2 Critical and Warning Command Processing

In addition to starting the daemon, the `fsdaemon` command can be used to specify commands to be run when critical or warning thresholds are reached. Full or relative path names for the commands must be specified. Options to the commands and command redirection can also be specified; if this is desired, the command must be enclosed in quotation marks. You start the command by using the `exec`(2) system call to execute `ksh`(1) with the command as the `-c` option, and so you should initialize the environment variables and working directory as needed for execution.

When a critical or warning command is started, all files except the `stdin` file, the `stdout` file, and the `stderr` file have been closed. These special files are initiated as follows:

- The `stdin` file is closed.
- The `stdout` file and the `stderr` file are open. In normal mode, they are assigned to `/usr/spool/fsmonitor/Fd.fd12`. In test mode, they are assigned to `./Fd.fd12`.

This means that, if the command needs to access any of these files, they must be redirected properly.

In test mode (`-t` option on the `fsdaemon` command), the file `Fd.fd12` is created in the directory the daemon inherits when it is started. In normal mode, the file is in the directory `/usr/spool/fsmonitor` unless you specify a different directory using the `-p` option directory. In normal mode, the daemon's home directory also defaults to `/usr/spool/fsmonitor`, unless the `-p` option specifies another directory. The command always starts with the home directory set to `/`.

See the `fsdaemon`(8) man page for a detailed description of the `fsdaemon` command and the options you can use to specify critical or warning commands.

6.3.3 Request Processing

The `fsdaemon` program includes a number of requests designed to configure the daemon, control the monitoring process, and allow the operator to view the current state of the file systems. Requests are created from the options specified with the `fsmon` command. Access to the request processor is through the request pipe (using `fsmon`).

The following paragraphs provide a brief overview of `fsmon` command options. For a detailed description of each option, see the `fsmon` man page.

The `-a` option lets you add a file system to the table of monitored file systems. The command lines for this function are as follows:

```
fsmon -a [-c nnn] [-i [c][w]|-n [c][w]] [-w nnn] [-e] filesystems
fsmon -a [-c nnn] [-w nnn] [-d] filesystems
```

The `-m` option allows you to change entries in the table of monitored file systems. The command lines are as follows:

```
fsmon -m [-c nnn] [-i cw | -n cw] [-w nnn] [-e] [-f cw] filesystems
fsmon -m [-c nnn] [-w nnn] [-d] [-f cw] filesystems
```

You can use the optional options with the `-a` or `-m` option to do the following:

- Set critical and warning threshold percentages.
- Enable or disable operator messages.
- Enable or disable monitoring for the specified file system.
- Manually set the critical or warning condition on the specified file system. (This function is intended for testing use.)

The *filesystems* operand can be a name, an ordinal, or the special name `all`. An *ordinal* is defined as the number that appears in the `n` column of the file system status display for a particular entry in the table. (See Example 1, page 279.) Ordinals allow you to indicate a particular file system without typing the full name. (Ordinals are not fixed but depend on the specific configuration and can change whenever an entry is added to the table of monitored file systems. Ordinals have meaning only to `fsdaemon`, as shown on the status display, and have no connection with any kind of file system ordinals UNICOS may use internally.) If you use the file system name `all` with the `-m` option, it alters every entry in the table.

New entries are inserted in alphabetical order as determined by the result of a `strcmp` () comparison, which causes the ordinals of all entries beyond the new one to increase by one.

The `-q` option causes `fsdaemon` to close and rename the log file and terminate. This option is available only if `fsdaemon` had been started with the `-q` option. The command line is as follows:

```
fsmon -q
```

The `-r` option signals the daemon that the critical or warning command has completed on the specified file systems by causing the "command is running" state to be removed from the selected entries. In effect, this reenables the monitoring of the specified file system. The command lines are as follows (the first is for the completion of a critical command, the second for the completion of warning command):

```
fsmon -r c filesystems
fsmon -r w filesystems
```

A command of the same type (critical or warning) is not restarted until the occupancy of the file system falls below and then crosses above the threshold again. This mechanism prevents more than one instance of a warning or critical command from being active on the same file system, but allows a critical command to be started if the warning command has not yet completed its operation. To prevent the daemon from immediately restarting the command (in the event that the actual state of the file system has not been changed enough to remove the threshold condition), the occupancy of the file system must fall below the threshold and then rise through it again before the command will start.

If appropriate for your site, you may also specify the `-r` option with `cw` to indicate that both critical and warning commands have been completed. The special file system name `all` resets every file system in the table. Use this name carefully because it could allow more than one command to be active on the file system at the same time.

The `-R` option resets file system monitoring by clearing the internal flags that indicate whether warning or critical commands are executing, and most state information from the table of monitored file systems. The command line is as follows:

```
fsmon -R filesystems
```

This option is intended to be used when corrective action has been completed following a warning or critical event and you want to enable critical and warning monitoring. The special name `all` resets every file system in the table. Use this option carefully because it could cause multiple commands to be active in the same file system. `fsdaemon` evaluates the state of the file system after this request is processed. A warning or critical event could occur immediately.

The `-s` option lets you display file system status by sending a status block in addition to the normal reply to a request. `fsmon` formats the information for the operator display as shown in Example 1, page 279. The command line is as follows:

`fsmon [-s cdew]` [*filesystems*]

The `-s` option with possible arguments `c`, `d`, `e`, and `w` limits the class of table entries to display from the list. `c` is critical, `d` is disabled, `e` is enabled, and `w` is warning. Any combination may be set, but you must specify at least one type.

The `-t` option suppresses display headers and informative messages. The following example displays all critical file systems, but suppresses the header shown in Example 1, page 279:

```
fsmon -t c
```

Currently this is the only directive that has a header, but `-t` is allowed with all directives.

The `-p` option specifies an alternate path to the daemon request pipe and log file. Communication between the daemon and `fsmon` uses named pipes and consists of mutually defined structures containing mixed ASCII and binary data.

6.3.4 Status Display

Example 1, page 279 is an example of a file system status display. This section describes the contents of this display.

Example 1:

```
n    File System    Status       Use    Warn    Crit   Time encountered
1    /              E-------     71.6%  85.0%   95.0%
2    /a             E-X----      95.2%  85.0%   95.0%   10:22:53 06/23
3    /arch          E-------     5.7%   85.0%   95.0%
4    /b             E-X----      97.6%  85.0%   95.0%   10:22:53 06/23
5    /bnch          E---W-X-     92.6%  85.0%   95.0%   10:22:53 06/23
6    /c             E---W-X-     92.3%  85.0%   95.0%   10:22:53 06/23
```

```
7    /core          E-------     84.6%  85.0%   95.0%
8    /d             E-------     83.4%  85.0%   95.0%
9    /drop          E-------     2.3%   85.0%   95.0%
10   /e             E-X----      96.3%  85.0%   95.0%   10:22:53 06/23
11   /ea            E-------     50.3%  85.0%   95.0%
12   /ea/usr        E-------     31.7%  85.0%   95.0%
13   /ea/usr/src    E-------     65.1%  85.0%   95.0%
14   /g             E---W-X-     89.3%  85.0%   95.0%   10:22:53 06/23
15   /h             E-X----      97.1%  85.0%   95.0%   10:22:53 06/23
16   /j             E---W-X-     92.9%  85.0%   95.0%   10:22:53 06/23
17   /l             E---W-X-     90.2%  85.0%   95.0%   10:22:53 06/23
18   /n             E-------     84.1%  85.0%   95.0%
19   /p             E-X----      99.5%  85.0%   95.0%   10:22:53 06/23
```

The `n` column holds the file system ordinal that can be used as a synonym for the file system name. The `File System` column holds the name of the file system. In order to fit the entire display line on an 80-column window, very long names may cause the remainder of the status line to appear below the name rather than to the right of it. The `Status` column has eight status flags. A position with `-` means that the status does not apply to the file system. Position 1 always has either `D`, `E`, or `*` visible.

The following list describes the status flags:

Status	Meaning
`E-------`	Monitoring is enabled
`D-------`	Monitoring is disabled
`*-------`	Monitoring error
`-C------`	Critical threshold detected
`--c-----`	Manual critical forced
`---X----`	Critical command executing
`----W---`	Warning threshold detected
`-----w--`	Manual warning forced
`------X-`	Warning command executing
`-------?`	Internal error

The `Use` column is the current usage level of the file system expressed as percentage. A file system that is disabled or in the error state does not have a value displayed. The `Warn` column is the warning percentage threshold currently set. The `Crit` column is the critical percentage threshold currently set. The `Time encountered` column is the time, in *hh*:*mm*:*ss mm*/*dd* format, when the warning or critical threshold level was reached. Critical conditions take precedence over warnings. If there is a monitoring error, a message is displayed in this column.

6.3.5 Using the `fsdaemon`(8) and `fsmon`(8) Commands

The following example shows how you can use the `fsdaemon` and `fsmon` commands to start and configure the file system monitor. In this example, the following commands are executed as part of the startup file when the system is brought up:

```
/etc/fsdaemon -w /admin/scripts/warning/ -c /admin/scripts/critical
/etc/fsmon -a -w 93 -c 97 / /usr /usr/tmp /tmp
/etc/fsmon -a -w 88 -c 90 /core
```

The `fsdaemon` command line starts the daemon and configures it to execute a warning script named `/admin/scripts/warning` when a warning threshold is reached and to execute a critical script named `/admin/scripts/critical` when a critical threshold is reached. What the scripts do (or if they exist) depends on the needs of your site. Because the daemon does not attempt to start another script of the same kind until a reset request is processed, the warning and critical scripts must end with a `fsmon` command to perform the reset request. See Section 6.3.3, page 277, for a description of `fsmon` reset options.

The `fsmon` command configures the daemon. The first `fsmon` command line (using the `-a` option) adds the file systems named `/`, `/usr`, `/usr/tmp`, and `/tmp` to the table of monitored file systems, and sets the warning threshold (using the `-w` option) to 93% and the critical threshold (using the `-c` option) to 97% for each of those file systems.

The second `fsmon` command line adds the file system named `/core` with a warning threshold of 88% and a critical threshold of 90%. You can use as many `fsmon` command lines as you need to add the file systems you want to monitor. All the file systems with the same threshold values can be added with one command.

With the daemon set up using the previous example, a display of the current status might look like the following:

```
n   File System   Status      Use   Warn   Crit  Time encountered

1   /             E-------    88.2% 93.0%  97.0%
2   /core         E-------    81.4% 88.0%  90.0%
```

```
3   /tmp          E-------     5.7% 93.0%  97.0%
4   /usr          E-------    61.3% 93.0%  97.0%
5   /usr/tmp      E-------     1.6% 93.0%  97.0%
```

If you want to change the critical threshold on `/tmp` to 95% and the warning threshold to 80%, you could use the following command:

```
/etc/fsmon -m -c 95 -w 80 /tmp
```

Because the `n` value shown on the display for `/tmp` is `3`, you could also use the following command:

```
/etc/fsmon -m -c 95 -w 80 3
```

After either of these commands has been executed, a display of the current status would look like the following:

```
n   File System   Status      Use   Warn   Crit  Time encountered

1   /             E-------    88.2% 93.0%  97.0%
2   /core         E-------    81.4% 88.0%  90.0%
3   /tmp          E-------     5.7% 80.0%  95.0%
4   /usr          E-------    61.3% 93.0%  97.0%
5   /usr/tmp      E-------     1.6% 93.0%  97.0%
```

6.3.6 The Log File

The log file records events that occur during the monitor daemon operation, and are intended for operations personnel. The recorded events include automatic actions taken by the monitor, operator requests that cause a change in the operation of the daemon, and error events detected by the daemon. An example of log file messages follows:

```
06/22  16:05:34  (06)  Fsmonitor initiated.
06/22  16:05:34  (06)  Critical command: 'critical_command'

06/22  16:05:34  (06)  Warning command: 'warning_command'
06/22  16:05:34  (06)  Entered Main loop
06/22  16:06:12  (06)  Added: /a -c  95.0% -w  85.0% -d
06/22  16:06:12  (06)  Added: /b -c  95.0% -w  85.0% -d
06/22  16:06:12  (06)  Added: /c -c  95.0% -w  85.0% -d
06/22  16:06:12  (06)  Added: /d -c  95.0% -w  85.0% -d
06/22  16:06:12  (06)  Added: /e -c  95.0% -w  85.0% -d
06/22  16:06:12  (06)  Added: /f -c  95.0% -w  85.0% -d
06/22  16:07:05  (06)  Changed: /c -c  98.0% -w  91.0% -e
```

```
06/22  16:07:05  (06)  Pid 28083 running: 'warning_command /c'
06/22  16:07:05  (06)  WARNING THRESHOLD ON /c
06/22  16:07:06  (06)  Changed: /f -c  98.0% -w  91.0% -e
06/22  16:07:07  (06)  Changed: /g -c  98.0% -w  91.0% -e
06/22  16:08:13  (06)  Daemon terminated; exit(0) "Stopped by quit"
```

The sample log shows the configuration of six file systems (`/a` through `/f`) with default levels of critical (`-c`) and warning (`-w`) thresholds. All file systems were initially disabled, but later (at `16:07:05`) a request changed the threshold values and enabled threshold detection for file system `/c`. As soon as monitoring was enabled on `/c`, a warning was detected, causing process ID `28083` to run the warning command. The last line of the log shows a normal termination caused by the quit request. When `fsdaemon` terminates, the name of the log file is changed to include the process ID. If the daemon is restarted, a new log file is opened.

6.4 Informative and Error Messages

Messages seen by a user of `fsmon` may come directly from `fsmon` or by way of the reply pipe from `fsdaemon`. This section lists the most important messages from both sources. All messages indicate errors, unless specifically stated otherwise, and cause an exit value of 1; informative messages cause an exit value of 0.

Usage messages that are not explained here adequately describe the error without additional explanation.

Message Description

`Daemon did not respond in 30 seconds`

`fsmon` waits for a reply from the daemon for a limited period of time before indicating that it did not respond. This could be caused by system load or scheduling problems, but also may indicate that `fsdaemon` is present but not responding.

`File system monitor daemon may not be running`

When `fsmon` has accepted its options, they must be translated and passed on to `fsdaemon` for processing. This message indicates that communication was impossible, possibly because the daemon was not running. This message also occurs when the `-p` option is given the wrong directory tree, or when the user does not have permission to access the daemon’s request pipe.

`fsnames required for -a, -m or -r`

The file system name is mandatory with these options.

`File system` *xxx* `exists in monitor list`

The named entry cannot be added because it already exists.

`File system` *xxx* `not in monitor list`

The named file system is not currently in the monitor list.

`Monitor list empty`

This is an error only when the `-m` option has been used, but it can occur as an informative message as well.

`Monitor list full`

Another file system may not be added to the monitor list. Because the monitor list size is set to twice the maximum number of mounted file systems when the daemon is started, this should not be a problem unless some sort of configuration error occurs.

`No entries selected`

This informative message states that no monitor table entries matched the type criteria. Select `-s cdew` to see everything in the table.

`Ordinal` *nn* `not in monitor list`

The file system ordinal is 0 or greater than the current maximum value.

6.5 Installation and Operation Information

This section describes some useful information about the monitor’s operational characteristics.

6.5.1 Installation

The commands `fsmon`(8) and `fsdaemon`(8) are installed by default in the `/etc` directory because they are intended as operator or administrator commands. The log file and command pipe are found in the `/usr/spool/fsmonitor` directory. Except for the file `Fd.fd12`, other files used during operation are named using `tempnam` (see `tmpnam`(3)) and so reside in the preferred temporary directory. None of the temporary files are intended to exist for long periods of time. File `Fd.fd12` is created in the directory described in Section 6.3.2, page 276, each time a command runs. This file holds all non-redirected standard output and standard error text from a command until the next command is started by the daemon. This file is intended for debugging or monitoring and is not intended as a log or other recording mechanism.

6.5.2 Operation

You must start the monitor daemon with critical and warning commands specified on its command line if you want any action to occur when a threshold is reached. You cannot change these command names without terminating and restarting `fsdaemon`. You can circumvent this restriction by writing critical and warning scripts that start the desired warning or critical operations. You can then alter or switch these initiation scripts during operation to provide whatever degree of flexibility is necessary.

Getting the monitor configured correctly is the most important issue following installation. Once `fsdaemon` is running and all the file systems are mounted, the easiest way to establish monitoring of all mounted file systems is to execute the following command line:

```
fsmon -a -c nnn -w nnn all
```

You must decide what critical threshold values (`-c` *nnn*) and warning values (`-w` *nnn*) are appropriate. (The defaults are 95 for critical and 85 for warning.) When starting `fsdaemon`, the critical and warning commands are the most important options. You should select which command to execute if either of the threshold conditions occurs.

After the monitor is configured and running, you may ask for displays and change its configuration by using the `fsmon`(8) options described previously. Because the monitor keeps track of old log files (it renames them before termination so the old file does not interfere with the next initiation), you should develop a procedure to clean up the old files from time to time. Do not allow the log to become very large under normal circumstances.

6.5.3 Permissions

Who uses the fsdaemon commands is determined by who can write to the command pipe. The daemon creates the command input pipe with owner read/write and group write permissions. This limits access to the owner and group that started the daemon. This may be changed by altering the value of IN PIPE MODE in the dmparams.h file in the fsdaemon_source directory. The directory in which the special files reside is created by the installation process to be owned by root and to belong to the operator group. This may also be changed if necessary. If permissions are changed, think about the implications, because other users could disable threshold monitoring. This could affect data migration, because in many environments threshold detection is used to activate the data migration feature.

6.5.4 Testing

To do any testing without disrupting a running version of the monitor, the path option (-p) is available. This makes it possible to run the monitor with its files and pipes in a directory separate from the default, so that two or more versions can run at the same time. The -t option with fsdaemon is also convenient for testing, because it prevents the process from detaching itself and running without a controlling terminal. The following command line is an example of testing fsdaemon:

```
fsdaemon -t -q -p /tmp/fsd.test -w warncmd -c critical &
```

6.5.5 Related Files

The running log file is usually named as follows:

```
/usr/spool/fsmonitor/Fd.log
```

When fsdaemon is terminated, the running log is renamed with the process ID of the daemon appended (Fd.log becomes Fd.log.12345, if the process ID is 12345). If for some reason the daemon does not rename the log file (for instance, if the system crashed), a new invocation of the daemon appends its output to the existing file.

Temporary files are created for the reply pipe (managed by the requesting side, not fsdaemon) and the reply pipe from the message daemon. The file names are prefixed with Fs. None of the temporary files should be left around when the daemon is correctly terminated.

System Activity Monitoring [7]

This section describes the design and implementation of the following system activity monitoring packages:

- Standard UNIX system activity package (`sar`, `sag`, `timex`, `mppview`, `xmppview`, `sadc`, `sa1`, and `sa2` commands)
- Cray system activity monitoring package (`sam`, `xsam`, `csam`, and `samdaemon` commands)
- Cray system activity reporting package (`sdc` and `tsar` commands)
- Disk usage monitoring

Warning: Although the standard UNIX system activity package (`sar` and related commands) is part of a Cray ML-Safe configuration of the UNICOS system, the Cray system activity monitoring and reporting commands described in this section are **not** part of the Cray ML-Safe configuration, including `sam`, `tsar`, and their associated commands. This section **does not** contain any further warnings or information pertaining to the use of a Cray ML-Safe configuration.

7.1 Standard UNIX System Activity Package (`sar`)

The UNICOS operating system contains a number of counters that are incremented as various system actions occur. The standard UNIX system activity package reports system-wide measurements for the UNICOS operating system, including central processing unit (CPU) utilization, disk and tape input/output (I/O) activities, terminal device activity, buffer usage, system calls, system switching and swapping, file-access activity, and queue activity. The package provides commands that generate various kinds of reports, allow you to monitor Cray MPP systems, and automatically generate daily reports.

The functions of the activity package are as follows:

Command	Function
`sag`(1)	Produces data and command files suitable for use on a front-end machine to produce graphic displays of system activity

`sar`(1)	Allows users to generate system activity reports in real time and to save system activities in a file for later use
`timex`(1)	A modified `time`(1) command that times a command and produces a report on concurrent system activity
`mppview`(8)	Displays massively parallel processing (MPP) system activity
`xmppview`(8)	Displays Cray MPP system activity through a graphic user interface

You can produce system activity daily reports automatically by using the `sadc`, `sa1`, and `sa2` commands (see `sar`(8)). Use of these commands is discussed in Section 7.1.3, page 295.

The system activity information reported by this package is derived from a set of counters in the operating system kernel, as described in Section 7.1.1, page 288.

The following sections provide information about the standard UNIX system activity package:

- System activity counters
- System activity commands
- Daily report generation
- Source files and scripts
- Derivations

7.1.1 System Activity Counters

The UNICOS operating system manages a number of counters that record various activities and provide a basis for the system activity reporting system. The data structure for most of these counters is defined in the `sysinfo` structure in the `/usr/include/sys/sysinfo.h` file. The system table overflow counters are stored in the `syserr` structure. The device activity counters are extracted from the device status tables. The I/O activity of any disk devices attached to the I/O subsystem (IOS) is recorded by the device activity counters.

The system activity counters monitored by the system activity package are described in the following sections.

7.1.1.1 CPU Time Counters

When the `sadc` program (see `sar`(8)) collects the data, it increments four standard time counters. The counters give an overview of CPU performance and record activity on each the information is also provided on a per-CPU basis. The counters are updated from the counters maintained in the processor working storage (`pws`) structure. These counters record activity on each CPU, also providing information on a per-CPU basis. Each time the UNICOS operating system executes an exchange into the kernel, it updates the counters in the `pws` table. These counters are incremented according to the mode that the CPU is in at the time of the exchange: idle, user, or kernel. I/O wait time is always 0. The `pws` counters are incremented by the number of machine cycles since the last exchange.

7.1.1.2 `lread` and `lwrite` Counters

The `lread` and `lwrite` counters record the number of logical read and write requests issued by the system block devices.

7.1.1.3 `bread` and `bwrite` Counters

The `bread` and `bwrite` counters record the number of times data is transferred between the system buffers and the block devices. The actual I/O operations are triggered by the logical I/O operations that cannot be satisfied by the current contents of the buffers. The ratio of block I/O to logical I/O is a common measurement of the effectiveness of the system buffer cache.

7.1.1.4 `phread` and `phwrite` Counters

The `phread` and `phwrite` counters record read and write requests issued by the system to raw devices.

7.1.1.5 `swapin` and `swapout` Counters

The `swapin` and `swapout` counters are incremented for each system request that initiates a transfer to or from the swap device. The amount of data transferred between the swap devices and memory is measured in blocks and counted by `bswapin` and `bswapout`.

7.1.1.6 `xswapin` and `xswapout` Counters

The `xswapin` counter is incremented each time a shared text segment is swapped in. The `xswapout` counter is incremented each time a shared text segment is freed.

7.1.1.7 `switch` and `syscall` Counters

The `switch` and `syscall` counters are related to the management of multiprogramming. The `syscall` counter is incremented every time a system call is invoked. In addition, the numbers of `read`(2), `write`(2), `fork`(2), and `exec`(2) system calls are kept in the `sysread`, `syswrite`, `sysfork`, and `sysexec` counters respectively.

The `pswitch` variable counts the times that the kernel subroutine `swtch` () was called to switch to another user. This occurs in the following situations:

- A system call results in a road block
- An interrupt occurs and awakens a higher-priority process
- A clock interrupt occurs

7.1.1.8 `runque`, `runocc`, `swpocc`, and `swpque` Counters

The `runque`, `runocc`, `swpocc`, and `swpque` counters record queue activities; they are implemented in the `clock.c` routine. At each 1-second interval, the `clock` routine examines the process table to see whether any processes are in core memory and are in the ready state. If so, the routine increments the `runocc` counter and adds the number of such processes to the `runque` counter. While examining the process table, the `clock` routine also checks to see whether any processes in the swap queue are in ready state. If the swap queue is occupied, the `clock` routine increments the `swpocc` counter and adds the number of processes in the swap queue to the `swpque` counter.

7.1.1.9 `iget`, `namei`, and `dirblk` Counters

The `iget`, `namei`, and `dirblk` counters apply to file-access operations. The `iget` and `namei` counters, in particular, are the names of UNICOS routines. These counters record the number of times that the respective routines are called.

The `namei` routine performs file path searches. It searches the various directory files to get the associated i-number (inode) of a file corresponding to the path.

The `iget` routine locates the inode entry of a particular file (i-number). It first searches the table of inodes in core memory. If the inode entry is not in the table, the `iget` routine gets the inode from the file system in which the file resides and enters the information in the table. The `iget` routine returns a pointer to this entry. The `namei` routine calls `iget`, but other file-access routines also call routine `iget`. Therefore, the `iget` counter is always greater than the `namei` counter.

The `dirblk` counter records the number of directory block reads issued by the system. The number of directory blocks read divided by the number of `namei` calls provides an estimate of the average path length of files.

7.1.1.10 `readch` and `writech` Counters

The `readch` and `writech` counters record the total number of bytes transferred by the `read`(2) and `write`(2) system calls.

7.1.1.11 `rcvint` and `xmtint` Counters

The device `rcvint` and `xmtint` counters measure T-packet activity to and from terminals attached to the IOS. The `rcvint` counter measures the number of packets flowing into the Cray computer system. The `xmtint` counter measures the number of packets acknowledged as received by the IOS.

7.1.1.12 `rawch`, `canch`, and `outch` Counters

The `rawch`, `canch`, and `outch` counters record the number of characters in the unbuffered queue, canonical queue, and output queue, respectively. Characters generated by terminal devices operating in unbuffered mode are counted in both `rawch` and, as they are edited, in `canch`.

7.1.1.13 `clists` Counter

The `clists` counter records the usage and overflow of `clists` so that you can carefully set the number of `clists` for terminal devices.

7.1.1.14 I/O Activities

All disk and tape I/O is monitored for each device. This counter shows the number of read and write operations on each device. Each read or write operation represents a maximum of 4096 bytes transferred. Response time and active time recorded on other UNIX systems is not useful here, because the IOS deals with the device.

7.1.1.15 Logical Device Cache

This counter records all logical device cache activity. The ratio of cache-to-user and cache-to-disk is a common measurement of the effectiveness of logical device cache.

If a user without root privileges executes the `sar`(1) command with the frequency arguments *t* and *n* specified, logical device cache data will not be available. Logical device cache statistics are obtained from `/dev/dsk`, which can only be read by users with root privileges.

7.1.1.16 Inode, File, Text, and Process Tables

The information from the inode, file, text, and process tables comes in two forms. The first shows the current utilization of table entries and the maximum configured number of table entries. The second is extracted from the `syserr` structure. When a table overflow occurs, the counter for the corresponding table is incremented.

7.1.1.17 `sysrda`, `syswra`, and `syslstio` Counters

The `sysrda` and `syswra` counters monitor `reada`(2) and `writea`(2) asynchronous I/O system call usage respectively. The `syslstio` counter counts `listio` usage.

7.1.1.18 `pkin`, `pkout`, and `pkbad` Counters

The `pkin`, `pkout`, and `pkbad` counters monitor IOS packet traffic.

7.1.1.19 `shuffle`, `textlock`, `datlock`, and `punlock` Counters

The `shuffle` counter measures the number of requests to shuffle a process into low memory. The `textlock` and `datlock` counters measure the number of text and data areas locked in memory. The `punlock` counter measures text and data area unlocks, unlocking the process.

7.1.1.20 Exchange Counts

The kernel logs user-initiated exchanges of the following error types:

Type	Description
`err`	Error exchange

`pre`	Program-range error
`ore`	Operand-range error
`fpi`	Floating-point interrupt
`dli`	Dead lock interrupt

7.1.2 System Activity Commands

The system activity package provides commands for generating various system activity reports. The `sar`(8) and `sag`(1) commands allow users to specify a sampling interval and number of intervals for examining system activity, and then to display the observed level of activity in tabular and graphic form. The `timex`(1) command reports the amount of system activity that occurred during the precise period of execution of a timed command. The `mppview`(8) command displays the system activity of a Cray MPP system. The `xmppview`(8) command displays Cray MPP system activity through a graphic user interface.

7.1.2.1 `sar`(8) Command

The `sar`(8) command reports on various types of system activity. When you specify frequency arguments *t* and *n*, `sar` invokes the `sadc` data collection program to sample the system activity counters in the operating system every *t* seconds for *n* intervals and generates system activity reports in real time. Generally, it is desirable to include the `-o` option to save the sampled data in a file for later examination. The format of the data file is shown in the file `/usr/src/prod/admin/sa/c1/sa.h` (not available on UNICOS binary-only systems). In addition to the system counters, a time stamp is included. It lists the time at which the sample was taken. If you do not supply frequency arguments, `sar` generates a system activity report for a specified time interval from an existing data file created by `sadc` (see `sar`(8)) at an earlier time. The `sar`(8) man page describes the use of the command and the various types of reports.

The `sar`(1) command extracts operating system activity information for a specified time interval. The following sections describe, by option, all displays that the `sar`(1) command can produce.

Note: By default, the `sar`(1) command runs every 10 minutes, reading data from tables in the kernel and writing to the `sa` file. It is recommended that the 10-minute default not be increased. (The overhead for `sar` is quite low.) You can decrease this interval to cause `sar`(1) to run more often, which can help solve performance problems by providing more continuous data; however, if the `sar`(1) interval is too small, this could generate too much system activity data for daily operation, increase requirements for disk space, and potentially degrade system performance.

7.1.2.2 `sag`(8) Command

The `sag`(8) command displays system activity data graphically. It relies on the data file produced by a prior run of `sar`(8); `sag` then creates plot commands and data files for any column or combination of columns of data from the `sar` report. These files are run on a machine that supports plotting. The command syntax allows you to specify either cross plots or time plotting. Select data items by using the `sar` column header names. (See the `sar`(8) man page for information about its options and usage.) The system activity graphics program cannot run on Cray systems.

7.1.2.3 `timex`(1) Command

The `timex`(1) command is an extension of the `time`(1) command. When you do not specify any options, `timex` behaves exactly like `time`. In addition to giving time information, it also prints a system activity report derived from the system counters. The `timex`(1) man page explains its use.

Although you, as an administrator, typically will use `timex` to measure a single command, you also can time multiple commands, either by combining them in an executable file and timing it, or, more concisely, by typing the following command:

```
timex sh -c "cmd1;cmd2; ..."
```

This use of `timex` establishes the necessary relationships between parent and child processes so that the user and system times consumed by *cmd1*, *cmd2*, and `sh` can be extracted correctly.

7.1.2.4 `mppview`(8) Command

The `mppview`(8) command displays a map of active partitions running on a Cray MPP system (see the `mppview`(8) man page). It uses the `curses`(3) library to drive the terminal display so that many terminal types can be supported. You can specify the host system to be monitored.

Through `rpc`(3), `mppview` communicates with the system activity monitoring (`sam`) server, `samdaemon`(8), running on the host system to obtain the information that you request.

7.1.2.5 `xmppview`(8) Command

A graphical user interface also is available through the `xmppview`(8) command. It uses the X Window System X11 tool. The display graphically represents usage of processing elements according to criteria you select and gives system performance statistics in tables accessible through pull-down menus. `xmppview` also contains a tutorial. (See the `xmppview`(8) man page for set-up requirements.)

7.1.3 Daily Report Generation

It is recommended that you routinely monitor and record system activity for historical analysis. This section describes how to produce a standard daily report of system activity automatically.

7.1.3.1 Facilities

For full descriptions of the commands used in generating reports, see the `sar`(8) man page. Use the following commands to produce a system activity report:

Command	Function
`sadc`	Reads system counters from the `/dev/kmem` and `/dev/dsk` files and records them in a file. In addition to the file argument, two frequency arguments are usually specified to indicate the sampling interval and number of samples to be taken. If no frequency arguments are given, `sadc` writes a dummy record in the file to indicate a system restart.
`sa1`	Invokes the `sadc` command to write the system counters in the daily data file `/usr/adm/sa/sa` *dd*; *dd* represents the day of the month. `sa1` may be invoked with sampling interval and iterations as arguments.
`sa2`	Invokes the `sar` command to generate the daily report `/usr/adm/sa/sar` *dd* from the daily data file `/usr/adm/sa/sa` *dd*. `sa2` also removes the daily data files and report files after one

week. The starting and ending times and all the report options of `sar` are applicable to `sa2`.

7.1.3.2 Suggested Operational Setup

It is suggested that you have the `cron`(8) daemon control the normal data collection and report generation operations. For example, the following sample entries in the `crontab` file `/usr/spool/cron/crontabs/root` would cause the data collection program `sadc` (see `sar`(8)) to be invoked every 20 minutes between 8 A.M. and 5 P.M., and every hour otherwise:

```
0 8-17 * * 1-5 "/usr/lib/sa/sa1 1200 3 &"

0 18-7 * * 1-5 "/usr/lib/sa/sa1 &"
```

Data sampling is more frequent during prime time to allow more detailed analysis. The `sa1` program should be invoked hourly rather than daily to ensure that, if the system crashes, data collection will resume within 1 hour after the system is restarted.

Invoking `sadc` through the `root` crontab file ensures that all data is collected. Logical device cache statistics are read from `/dev/dsk`, which is readable only by `root`.

Because the system activity counters are reset to when the system is restarted, `sadc` writes a special record in the data file to reflect this situation. To accomplish this process, `sadc` is invoked with no frequency arguments within `/etc/rc` when going to multiuser state:

```
/usr/lib/sa/sadc /usr/adm/sa/sa‘date +%d
```

For more information on using `rc`, see General UNICOS System Administration, publication SG-2301.

The `cron` command also controls the invocation of `sar` to generate the daily report by the shell procedure `sa2`. You may choose the time period to be covered by the daily report and the groups of system activity to be reported.

For example, if the following is an entry in the `crontab` file `/usr/spool/cron/crontabs/root`, `cron` executes the `sar` command to generate daily reports from the daily data file at 20:00 on weekdays:

```
0 20 * * 1-5 su sys -c "/usr/lib/sa/sa2 -s 8:00 -e 18:00 -i3600 -A"
```

The report includes a full set of information from 08:00 to 18:00.

In case of a problem, such as disk space shortage, these data files and report files can be removed by the super user. The man page entry for `sar`(1) describes the daily report generation procedures.

7.1.4 Source Files and Scripts

(Not available on UNICOS binary-only systems.) The source files and shell programs of the system activity package are located in the `/usr/src/prod/admin/sa/c1` directory.

File or Program	Description
`sa.h`	A system activity header file that defines the structure of data file and device information for measured devices. It is included in `sadc.c`, `sar.c`, and `timex.c` records.
`sadc.c`	A data collection program that accesses the `/dev/kmem` and `/dev/dsk` files to read the system activity counters and that writes data either on standard output or on a binary data file. It is invoked when the `sar`(1) command generates a real-time report. It is also invoked indirectly by entries in the `/usr/spool/cron/crontabs/root` file to collect system activity data.
`sar.c`	A report generation program that invokes the `sadc` command to examine system activity data, generates reports in real time, and saves the data to a file for later usage. It also can generate system activity reports from an existing data file. It is invoked indirectly by the `cron`(8) daemon to generate daily reports.
`saghdr.h`	A header file for the `saga.c` and `sagb.c` programs. It contains data structures and variables used by `saga.c` and `sagb.c`.
`saga.c`, `sagb.c`	A graph generation program that first invokes `sar` to format the data of a data file in tabular form and writes a command file and data files so that the `sar` data can be displayed in graphic form on another machine.

`sa1.sh`	A shell script that invokes `sadc` to write data file records. It is activated by entries in the file `/usr/spool/cron/crontabs/root`.
`sa2.sh`	A shell script that invokes `sar` to generate the report. It also removes the daily data files and daily report files after a week. It is activated on weekdays by an entry in the `/usr/spool/cron/crontabs/root` file.
`timex.c`	A program that times a command and generates a system activity report.

7.1.5 Derivations

This section lists the derivations of reported items. Each item discussed is the data difference sampled at two distinct times, *t1* and *t2* (in seconds).

CPU utilization time is as follows:

%-of-cpu `=` `(`*cpu-util* `/` *cpu-total*`)` `* 100`

The value of *cpu-util* is either the idle, user, or kernel (system) CPU utilization time. The value of *cpu-total* is the sum of the idle, user, and kernel (system) CPU utilization time. For a system with multiple CPUs, the information is also presented as the sum of the utilization time of each CPU, divided by the sums of the idle, user, and kernel utilization time, divided by the number of CPUs.

The cache hit rate is as follows:

%-of-cache-I/O `=` `((`*logical-I/O* `-` *block-I/O*`)` `/` *logical-I/O*`)` `* 100`

I/O can be either a cache read or cache write operation.

Queue activity is as follows:

avg-x-queue-length `=` *x-queue* `/` *x-queue-occupied-time*`;`
%-of-x-queue-occupied-time `=` *x-queue-occupied-time* `/` `(`*t2* `-` *t1*`);`

The value of *x-queue* is the length of either a run or swap queue.

The remainder of system activity is as follows:

avg-rate-of-x `=` *x* `/` `(`*t2* `-` *t1*`)`

The rate is expressed as occurrences per second. The value of *x* can be swap or drop in / out, terminal device activities, `read/write` characters, block read/write, logical read/write, process switch, system calls, read/write, fork/exec, iget, nami, directory blocks read, disk activities, IOS packets in and out, secondary cache read and written, clists active, shuffles, text/data and process unlocks, and asynchronous I/O activities.

7.2 Cray System Activity Monitoring (sam) Package

The Cray system activity monitor, sam, collects and displays system activity data from selected Cray computer systems. It consists of a data acquisition daemon, samdaemon, and two display clients, xsam and csam. Figure 6 illustrates the hardware configuration options for sam.

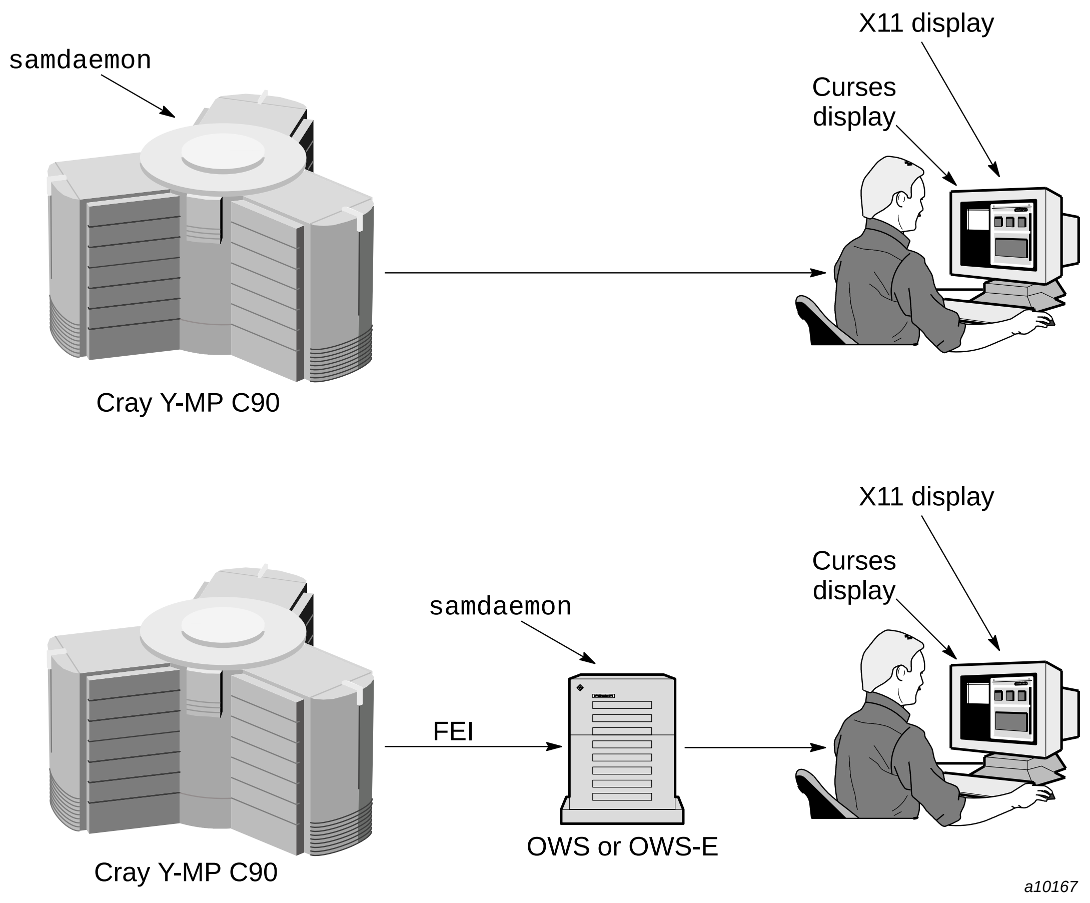


Figure 6. Hardware Configuration Options for Sam

The samdaemon process can run on a Cray mainframe or on a connected operator workstation (OWS). When samdaemon is executed on a mainframe, data collection is done through the /dev/kmem special file using read(2) system calls. When samdaemon is executed on an operator workstation it uses the I/O path of the front-end interface to access data fields in memory, thus reducing the load on the UNICOS operating system.

The sam(8) command is a generic interface that invokes either the xsam utility (for use with terminals that run the X Window System) or the csam utility (for use with terminals compatible with curses(3)). Both utilities provide a user interface with menus for selecting and displaying system activity data. Alternately, you can use the `xsam` or the `csam` command to enter the menus.

See the `sam`(8) man page for more information on which client `sam` invokes and on their respective displays.

The following sections discuss the daemon and commands:

Utility	Description
`samdaemon`	System activity data acquisition daemon
`csam`	Displays system activity data using the `curses`(3) library routines
`xsam`	Displays system activity data using the X Window System

7.2.1 samdaemon(8) Process

The system activity monitor daemon, `samdaemon`(8), reads kernel data structures in memory as requested by the display clients, `xsam`(8) or `csam`(8). `samdaemon` can run on a Cray mainframe or on a connected operator workstation. The daemon and the clients communicate through the Remote Procedure Call (RPC) utility. When a client is not requesting data, `samdaemon` ceases to read from memory.

When started, `samdaemon` generates a private configuration database by `forking` a configuration utility, `sama`. On the Cray mainframe, `sama` uses the `nlist`(3) library routine to obtain the necessary data addresses. When `samdaemon` is running on the operator workstation, it generates this database by having `sama` make a request to the `samdaemon` process running on the Cray mainframe. Once `samdaemon` is initialized on the operator workstation, all subsequent data is obtained directly from mainframe memory through the front-end interface (FEI) connection without involvement of any processes running on the mainframe.

The `samdaemon` process provides access control to UNICOS data with the use of a validation file. If `samdaemon` finds a validation file (`/usr/lib/samdaemon.val`, by default), it will return data only for validated users.

See the `samdaemon`(8) man page in the *UNICOS Administrator Commands Reference Manual*, for more details on `samdaemon`, its options, and setup.

Note: The `samdaemon` command is not intended to be built and installed on an operator workstation by a site; it is part of the binary release for the operator workstation. `samdaemon` will not build on an operator workstation.

7.2.2 `csam`(8) Utility

The `csam`(8) utility displays system activity on a terminal that is compatible with the `curses`(3) library routines. `csam` can execute either on a Cray system or on a connected operator workstation. `csam` receives data by communicating with the `samdaemon` process on the local host, or on a remote host if one has been specified.

Because `csam` relies on the `curses` library routines, it can support a variety of terminals. `csam` is able to check the size of the screen and the `LINES` environment variable and adjust most displays to fill the screen.

You can invoke `csam` by using the `csam` command or the generic `sam` command. You can select various system performance statistics for display. See the `csam`(8) man page in the *UNICOS Administrator Commands Reference Manual*, for more detailed information on the `csam` command and its options.

7.2.2.1 `csam`(8) Commands

Once `csam` is running on your terminal, you can use the following commands to change displays, move within a display, increase or decrease refresh rates, and exit the system:

Command	Description
`c` or `C`	Selects host system configuration display.
`d` or `D`	Selects disk display.
`e` or `E`	Quits program.
`f` or `F`	Selects ldcache display.
`g` or `G`	Leaves single-step mode.
`h` or `H`	Selects help screen.
`k` or `K`	Selects kernel display.
`l` or `L`	Selects logical device display.
`m` or `M`	Selects memory display.
`n` or `N`	Advances to next page (disk, ldcache, logical device, process, and tape displays). To reset these

	scrolled displays, use the `d`, `f`, `l`, `p`, or `t` command.
`p` or `P`	Selects process display.
`q` or `Q`	Quits program.
`r` or `R`	Resets bar graphs and ldcache display.
`s` or `S`	Selects user/kernel/wait/idle usage on kernel display.
`t` or `T`	Selects tape display. (Implementation deferred)
`u` or `U`	Selects user and system CPU usage on kernel display.
`w` or `W`	Selects swap map display.
`x` or `X`	Selects top process display.
`y` or `Y`	Selects system call display.
`z` or `Z`	Selects record/replay control panel. The `csam` record/replay function is described in detail in this section.
`+`	Increases refresh interval by 1 second.
`-`	Decreases refresh interval by 1 second. The refresh interval cannot be lowered below the refresh rates set by the server.
`.`	(Type the dot character.) Enters single-step mode. This freezes the display refresh until you press the space bar to advance it.
space bar	Advances the display in single-step mode.
numeric input	Selects a process ID for the snap display. This is only accepted in the process display. The ID must be terminated by a carriage return. The erase and kill characters are accepted during this numeric input.

7.2.2.2 Record/replay Function

The record/replay function works in a manner similar to that of an audio tape recorder. From the record/replay control screen, you can set up `csam` to record data for replay at a later time or to replay data that was recorded at an earlier time. Enter the `csam` record / replay control screen by typing the `z` or `Z` command while the cursor is in any `csam` terminal screen.

By default, the display refresh rate of the replay function is set at 1 second, which is the minimum rate. You can adjust this rate by using the `+` and `-` commands. However, altering the playback refresh rate does not alter the refresh rates for displays obtained directly from `samdaemon`.

The following default values are displayed near the top of the record / replay control screen when it is invoked:

```
Record/Replay   File name:  client.recplay
                Mode     :  record
                State    :  not loaded
```

These lines are referred to as the status lines.

Status line	Indicates
`File name`	The name of the file to which data will be recorded or from which data will be replayed.
`Mode`	Either `record` or `replay`.

`State`	The current state of the file.
	
	The `State` status line contains two fields. The first field displays one of three values:
	
	• `not loaded`
	• `stopped`
	• `running`
	
	The second field may be blank or may display either of two values:
	
	• `rewound`
	• `end of tape`
	
	The following states are what you typically see:
	
	`not loaded`
	`stopped` `rewound`
	`stopped` `end of tape`

`running`	`end of tape`
`running`	`rewound`

Commonly used command keys are displayed below the status lines on the screen. While the cursor is in the record / replay control screen, you can use the following keys to set up either a record session or a replay session:

Key	Description
`s`	Stops recording or replaying. The `State` status line displays `stopped`.
`b`	Begins (starts) recording or replaying. The `State` status line displays `running`.
`r`	Rewinds the file. You must return to the beginning of the file in order to replay the data recorded there. A file is at `end of tape` position when you stop recording or when all the data has been replayed. A file is rewound automatically when it is loaded.
`f`	Fast forward to the end of file. This allows you to append additional data to the end of the file.
`l`	Load the file. This is required before recording or replaying can begin. The `State` status line displays `stopped rewound`.
`e`	Eject the file. The file is no longer loaded. The `State` status line displays `not loaded`.
`d`	Done setting up record / replay. This command returns you to the `csam Help` screen. You can use this command to exit the record / replay control screen in order to switch to one of the data display screens and begin recording or replaying data.
`m`	Switch mode. This command toggles between record mode and replay mode. The `Mode` status line displays which mode is currently active. The file is unloaded automatically when the mode is changed.
`n`	Change file. This allows you to change the file name for recording or replaying. Terminate the name with RETURN. The `File name` status line displays the current file name.
`q`	Quit. This command allows you to exit `csam` and return to the shell.
`t`	Truncate. This command truncates the file at the position where the command is entered. For example, if you are replaying data

and want to truncate the file, you would return to the Record / Replay Control screen by entering the `z` or `Z` command, and then pressing the `t` key. This position becomes the end of the file. The only way to overwrite the file is to use the `t` (truncate) command subsequent to pressing the `r` (rewind) key in the record / replay screen; pressing the `b` (begin) key in record mode automatically moves to the end of the file (the `end of tape` position).

Recording data. To record data, follow these steps, starting with the cursor in the record / replay control screen:

1. Press `m` to select record mode if `record` is not currently displayed on the `Mode` status line.
2. If desired, press `n` to change the file name, then press the `RETURN` key.
3. Press `l` to load the file.
4. Press `b` to start recording.
5. Press `d` to exit the record / replay control screen in order to select data for recording.
6. In the `csam` terminal screen, select the data you want to record and enter the appropriate command to display that screen. For example, to record kernel data, enter the `k` command. The data displayed on the kernel screen will be recorded to the file.
7. To stop recording data, enter the `z` or `Z` command, which returns you to the record / replay control screen, and then press the `s` key.

Replaying data. To replay data, follow these steps, starting with the cursor in the record / replay control screen:

1. If the second field of the `State` status line does not display `rewound`, assure that replay starts at the beginning of the file by pressing `r` to rewind the file.
2. To replay a file other than the one displayed on the `File name` status line, press `n` and enter a file name and press `RETURN`.
3. Press `m` to select replay mode if `replay` is not currently displayed on the `Mode` status line.
4. Press `l` to load the file.

5. Press `b` to start the replay.
6. Press `d` to exit the record / replay control screen in order to select the data to replay. For example, in the `csam` terminal screen, enter `k` to replay kernel data. This will cause `csam` to switch to that screen and immediately begin displaying the recorded data. The message, `End of file reached on playback file`, will be displayed after all data have been displayed.
7. To stop replaying data, type `z` or `Z`, which returns you to the record / replay control screen, and then type `s`.

7.2.3 `xsam` Utility

The `xsam`(8) utility allows you to use graphic displays to monitor system activity on a UNICOS system. `xsam` can execute on either a Cray system or on a connected operator workstation, and its display can be viewed on any terminal running the X Window System that is connected to the network. The `xsam` utility receives data by communicating with the `samdaemon` process on the local host, or on a remote host if one has been specified.

You can invoke the `xsam` utility by using the command `xsam` or the generic command `sam` with a valid `DISPLAY` shell environment variable. `xsam` accepts the following options:

Option	Description
`X11 options`	Regular X11 options to control X11 defaults.
`-f file`	Initial startup script.

7.2.3.1 X11 Window Settings

You can set your regular X11 default settings (in the `.Xdefaults` file or using the `xrdb`(1X) command) by using the application name `xsam`. For example, the following command will set your default font to `10x20`:

```
xsam*Font:      10x20
```

In addition, you can specify individual X11 defaults by including the widget name in the `.Xdefaults` statement. The following widget names are defined by `xsam`:

```
"menu bar"              : the menu bar for all windows
"menu entry"            : all pop-up menus
"console"               : the main xsam console
```

```
"setup display"         : the setup window
"host display"          : the host window
"help display"          : the help window
"config display"        : the target configuration display
"device display"        : disk and logical I/O display
"graph console"         : graph console for kernel and I/O graphs
"graph display"         : graph display for kernel and I/O graphs
"map display"           : main and swap memory display
"snapshot display"      : process snapshot display
```

The following example sets the background for `menu bar` to `lightblue` and also specifies the font for all menus. The font for the `graph console` is set to `fg-22` and to `fixed` for the `graph display` (different from the `menu bar` in the `graph display`).

```
xsam*menu bar.background:       lightblue
xsam*menu entry.font: -*-helvetica-bold-o-*-*-20-*-*-*-*-*-*-*
xsam*graph console*Font:   fg-22
xsam*graph display.font:   fixed
```

7.2.3.2 `xsam` Windows

The `xsam` utility creates six types of windows, described as follows:

Window	Description
Console window	Controls all aspects of `xsam`. It can be used to request other `xsam` displays. It also contains a message section that displays all error and informational messages.
Configuration window	Shows aspects of the hardware and software configuration of the target system.
Graph window	Monitors various kernel counters including CPU utilization, I/O statistics, and system call usages.
Map window	Displays a representation of memory. Within the map are several columns of boxes, each with its own name and each corresponding to a process in memory. The size of each box is proportional to the size of the process in main memory. As the process moves or grows, the box moves and grows.

	Command options allow you to display the process ID, view a section of memory, and view a snapshot of any process.
Snapshot window	Displays information about a specific process in text form. The data is extracted from the kernel's process structure for the target process. A snapshot can be requested from a memory map or with the snapshot command. The snapshot window contains several sets of statistics including user information (such as the size and address of the process and user and process flags), scheduling parameters, and an area devoted to user identification.
Device I/O window	Displays device configuration and I/O performance on a system-wide level, as well as on an individual device level. Possible device types are disks and logical devices.

7.2.3.3 Console Window and Available Commands

After the `xsam` utility is started, it opens a console window. This window is the primary interface for all information about `xsam` and interaction with the `xsam` utility, and is described in detail in this section. The console window, shown in Figure 7, consists of four major parts:

- Menu bar, at the top of the window
- Information area
- Command input area, in the middle of the console window
- Message output area, at the bottom of the console window

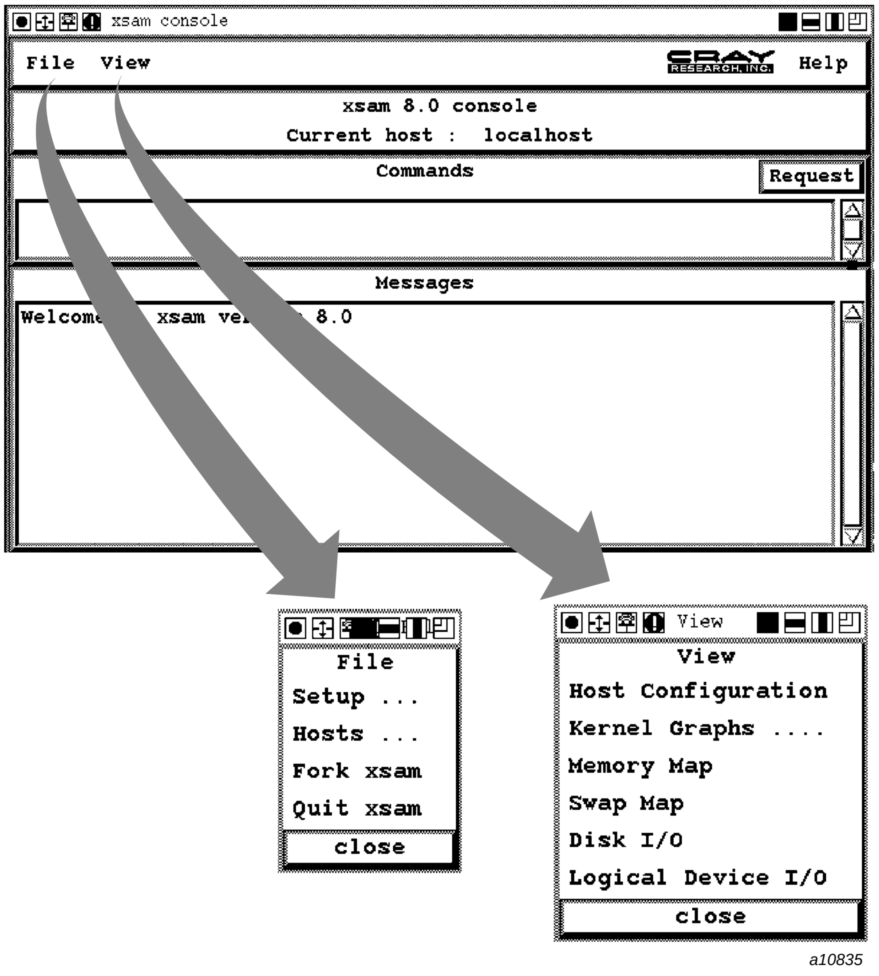


Figure 7. The `xsam` Console Window

To show a menu, click the left mouse button when the cursor is placed on the menu name you want to select. Most of the menu items will start a new display.

The console menu bar provides access to three menus: `File`, `View`, and `Help`. A Cray logo also is shown: when selected, it displays copyright information about `xsam`.

- The `File` menu has the following entries:

  ```
  Setup ...
  Hosts ...
  Fork xsam
  Quit xsam
  ```

- The `View` menu has the following entries:

  ```
  Host Configuration
  Kernel Graphs ...
  Memory Map
  Swap Map
  Disk I/O
  Logical Device I/O
  ```

- The `Help` menu has the following entry:

  ```
  General Information
  ```

To activate the associated window for these entries and others, select the corresponding menu entry by clicking the left mouse button.

> **Note:** Alternately, as mentioned throughout the discussion of `xsam` displays, you can enter the commands directly in the command input area of the console window to bypass the menu structure or to specify options to your command.

The information area in the console window shows some global identification, such as `xsam 9.0 console` and the name of the current host.

The `Commands` input area provides an alternative way to activate a function of `xsam`, by entering a command in the input area. With some exceptions, both the menu structure and direct command input allow access to the same functions. Whereas using `xsam` through the menu interface is self explanatory, the command interface is summarized here.

After you enter a command in the `Commands` input area, start the command by using the left mouse button to click on the `Request` button.

> **Note:** Under normal conditions, requests also can be started by pressing the `RETURN` key while the cursor is positioned in the input area. However, the widget used to implement the input area allows some sophisticated editing (see the "Text Widget" section in the *Cray Doo-Dad Set Reference Manual*) that can disable this feature (such as using the built-in `vi` style editing). In those situations, a command must be started by using the `Request` button.

The following list shows the commands you can use in the commands input area:

`quit`
`help`
`host` [*host name*]
`setup` *setup options*
`script` *filename*
`config` [*X11 options*]
`graph` [*X11 options] name [name* ...]
`map` [*X11 options] name [map options*]
`snapshot` [*X11 options] pid*
`device` [*X11 options] name*

The menu interface allows you to issue all of these commands except the `script` command. (In addition, the `fork` and `record/playback` commands are available from the console menu.) The `script` command initiates execution of `xsam` commands from a script file. No additional support, such as flow control within the script or parameter substitution, is available. Although scripts can be nested, all scripts operate within the same `xsam` environment. (For instance, if a lower-level script changes the current host, this effects the execution of the higher-level scripts as well).

Several commands also allow you to specify X11 options. However, not all normal X11 options are available on the `xsam` command input level.

The following X11 options are available:

```
bg        color
fg        color
geometry  [[Width]x[Height]][+Xposition][+Yposition]
fn        font
```

You can also specify X11 options by using the X11 set-up tools. For more information about X11 setup, see the online help facility available in `xsam`.

> **Note:** Some window managers vary regarding when to grant geometry requests. Check the manual for your window manager for information about setting up your window manager correctly.

The `Messages` area echos the commands issued either through the menu selections or by entering the commands directly in the `commands` area. It also displays informational and error messages.

In both the `Commands` and the `Messages` areas, you can use the scroll bar at the far right of each area to view the history of entries. The left mouse button moves the text down, while the right button moves the text up; the position of the pointer along the scroll bar controls how far the text moves with each click.

7.2.3.4 `xsam` Commands

From the console window, the `xsam` utility allows all of the commands available from the command input area, except the `script` command. In addition, the console menu makes available the `fork` and `record/playback` commands.

Command	Description
`Quit`	To terminate the entire `xsam` session, you can either select the `Quit xsam` menu entry in the `File` menu or enter `quit` in the `Commands` area. The menu entry may ask for confirmation.
`Help`	Select the `General Information` entry in the `Help` menu or enter the `help` command to bring up this display. You can click on an item in the `Help Options` area to locate it quickly in the help display. Use the `DISMISS` button to exit the help display.
`Host`	`xsam` allows you to monitor several hosts from within a single session. This is controlled by using either the `host display` menu or the `host` command. You can view multiple displays monitoring multiple hosts in a single `xsam` session. To activate the `Host` display, select the `Hosts ...` entry in the `File` menu. To enter a new host name from the `Host` display, move the cursor to the area labeled `Set Host`, enter the host name and press the `RETURN` key. The new host becomes the current host and also is highlighted in the `Available Hosts` area. You also can change the current host by clicking on its name in the `Available Hosts` area. A host also can be entered and made current by typing *host hostname* in the `Commands` input area of the console window. The `host` command without any arguments displays the current host in the `Messages` area of the console window.

`Fork`

Use the `Fork xsam` entry in the `File` menu to disconnect your `xsam` session from your tty session. This allows you to continue running `xsam` and to execute other programs or UNICOS commands within the same tty session. `Fork` can be executed only once during a session. Its function is disabled if logging is active and connected to your tty. `Fork` also can be started by the command:

```
setup fork
```

`Setup`

The `Setup` display provides control over resources that are local to your `xsam` session. To activate the `Setup` display, select the `Setup ...` entry in the `File` menu. Alternately, you can execute set-up commands in the console window to customize your X11 display.

The `setup` display allows you to select three options:

```
Echo Commands to Console Window
Exit from Script upon Error
Confirm Program Quit Request
```

They can be activated either from within the `setup` display or by entering the following commands in the console window:

```
setup [no]echo
setup [no]abort
setup [no]confirm
```

The left mouse button acts as a toggle on the display to turn the options on (represented by a dark box) and off (represented by a light box).

The `Setup` display area labeled `Record and Playback Control` controls the data recording and playback features. To specify the file name you want to use, move the mouse to the area labeled `Record File Name` or `Playback File Name`, enter the file name and press the `RETURN` key to activate the file. You must click on the

`START` button to begin recording or playback. The record and playback feature is not accessible through the command interface. See the `Record/Playback` description in this list for further information.

The `Setup` display area labeled `Logfile Control` is used to control command and debug output logging. To activate logging, move the mouse into the area labeled `Log File Name`, enter a file name and press the `RETURN` key. Then select, by clicking, one or both of the types of logging messages available:

`Console`

or

`Debug`

An empty string entered as a file name will close the log file without opening a new one. The file names `stdout` and `stderr` correspond to the related standard UNIX files.

These logging functions also can be controlled by entering any of the following commands in the Console window:

```
setup logfile file name
setup nologfile
setup [no]debug
setup [no]conslog
```

`Record/Playback`

You must access `record` and `playback` sessions through the `Setup` display of the `File` menu; they are not available from the `Commands` input area of the console window. When `recording` is turned on, all incoming data is written to a file that can be used for `playback`. You can start recording at any time during a normal session.

When `playback` is turned on, data is taken from that file instead of contacting a running `samdaemon` process. Use of the `playback` feature is restricted in the following ways:

- `playback` sessions are single host sessions.
- `playback` can be activated only before connection to `samdaemon` has been established. After `playback` has been activated, no further real connections can be made within this `xsam` session.
- `playback` cannot show any data that was not recorded, but can show more information than was monitored while recording the data. What is or is not available depends on the packaging of the information that is exchanged between `samdaemon` and the `xsam` client. For example, if recording was done while a `graph user-0` was active, all information about CPU utilization is included in the data file. Thus, during playback, you may have `user-Sum`, `kernel-Sum`, and `idle-Sum` active.

  To activate either `record` or `playback`, you must first enter the name of the data file, press the RETURN key to select the file, and then click the START button. After `record` or `playback` is activated, use the associated buttons as you would on an audio tape deck, with the exception of the TRUNC button. This button truncates the file at the position when the button is clicked. For example, if you are playing back a file and stop the playback by clicking the STOP button, and then you click the `TRUNC` button, the file will be truncated at the point currently displayed.

After activating the playback process, you must select which data to display from the file (through the `View` menu or through commands).

`Config`

The `Config` display is available through the `Host configuration` option of the `View` menu. It displays information about the hardware and software configuration of your target system. You also can activate it with the following command:

```
config
```

`Graph`

Select the `Kernel Graphs ...` entry in the `View` menu to bring up the `graph console`, which will show you a list of available graphs for that specific host. You can select graphs by clicking on their names. Selected graphs are indicated by a dark box; clicking on them again will deselect them, as indicated by a light box. Click on the `Request Selected Graphs` button to bring up the `Graph` display. Graphs are shown in alphabetical order.

If a graph name is known to the user, it also can be activated by using the following command:

`graph` *graphname1 graphname2* ...

In addition, this allows you to specify some X11 options. Graphs are shown in the order specified in the command.

The `Graph` display shows actual performance data. It also shows the time of day of the host machine.

Within a `Graph` display, you can use the mouse to request additional displays such as time information. For information about using the mouse within a `Graph` display, see the online help facility available in `xsam`.

Two interesting types of graphs are the `-all` and `-sum` graphs for CPU utilization data. The `-all` graph shows the current utilization of all configured CPUs of the target system within a single graph (no history). This is useful for a quick overview of general system performance from hosts with many CPUs. The `-sum` graph shows the CPU utilization accumulated over all configured CPUs of the target host. These two options can be used for CPU utilization graphs `user`, `kernel`, and `idle`.

You can also display a selected graph for a specific CPU with the following `graph` command:

`graph` *graphtype-CPU*

`graphtype` may be `user`, `kernel`, or `idle`; the `-` is required (no spaces), and `CPU` is a valid CPU number.

`Map` — Select either the `Memory Map` or the `Swap Map` entries in the `View` menu to bring up the `map display`.

This display shows a box (or a line, for small entries) for every process that currently resides on the specified target memory. The size of a box is related to its actual process size, and the position of a box is related to its position within the target memory.

You can zoom into an active `Map` display by moving the cursor to the desired start position in the `Map` display, clicking the left mouse button (this will show you the actual start position), moving to the desired end position, and clicking the right mouse button. A new `Map` display will be activated.

A `Map` display also can be activated with the following command:

`map [`*map name*`][-start `*addr*`][-end `*addr*`][-id][-size]`

map name	Either memory (the default), or swap.
`-start` *addr*	Specifies a specific starting address of the map. Leading zeroes indicate an octal value.
`-end` *addr*	Specifies a specific ending address of the map. Leading zeroes indicate an octal value.
`-id`	Requests that process IDs be displayed. (Space permitting)
`-size`	Requests that process sizes be displayed. (Space permitting)

`Snapshot`

The `Snapshot` display shows detailed information about running processes. You can activate the display directly from the `Map` display by positioning the cursor inside the box that corresponds with the desired process and pressing either the middle mouse button or the `p` key. The snapshot display can also be activated with the following command:

`snapshot` < *pid* >

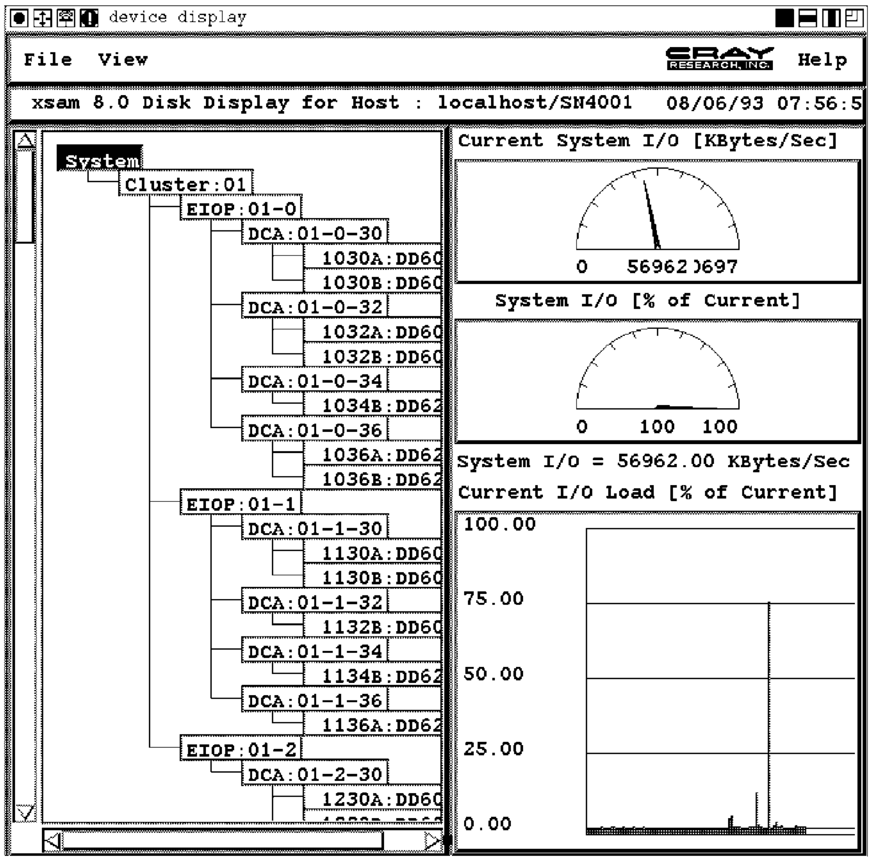
pid is a valid process ID number.

`Device`

Select either the `Disk I/O` or the `Logical Device I/O` entry in the `View` menu to bring up the `Device` display.

A `Device` display can also be activated by the `device` command. This command allows you to specify some X11 options. In addition you must specify a valid device name.

Figure 8 shows the `xsam` device display window.



```
xsam 8.0 Disk Display for Host : localhost/SN4001   08/06/93 07:56:5
System I/O = 56962.00 KBytes/Sec
```

a10825

Figure 8. The `xsam` Device Display Window

The `Device` display shows four fields of information.

- On the left side of the window is a tree of the current configuration. This tree contains all real nodes, such as physical disks or logical devices, and several configuration nodes.

- The disk display shows nodes for all `IO-Clusters`, for `EIOPs`, for disk channels (DCAs), and the global `System` node.
- The logical device display shows the global `System` node and two additional nodes for `Cached` and `Normal` (non-cached) devices.

  Possible actions you can take by using the cursor inside the configuration tree are described in the "Tree Widget" section of the *Cray Doo-Dad Set Reference Manual*. The mouse action `select-node`, triggered by pressing the left button on the mouse, selects a node. This selection affects the function of the lower speedometer, which shows the I/O rate of the sub-tree starting at this node.

• The right side of the device display shows two speedometers and a bar diagram.

- The upper speedometer (labeled `Current System I/O [KBytes/Sec]`) shows the current and maximum I/O rates in kilobytes per second. The maximum value is the highest I/O rate seen in the lifetime of this display.
- The lower speedometer (labeled `<NAME> I/O [% of Current`) shows the I/O percentage rate of the current selected device subtree with respect to the current I/O rate (as shown in the upper speedometer). The current selected device subtree starts at the node highlighted in the configuration tree. An additional line shows the absolute value of this I/O rate.
- The bar diagram shows the I/O percentage rate of all real devices with respect to the current I/O rate (as shown in the upper speedometer). Configuration nodes (such as `System`) are not part of this diagram.

  You can perform three actions with the cursor positioned in the bar diagram. The left and right mouse buttons change the lower and upper ends, respectively, of the nodes displayed. This allows you to zoom into the diagram for sites with a larger disk or logical device farm. The middle mouse button selects a specific disk or logical device node. This, too, affects the function of the lower speedometer. If the node is not hidden in the configuration tree, the configuration tree is changed in such a way that it is more likely that the selected node is visible; if the node is hidden, its name is printed on the main console.

The `View` menu of the `Device` display contains two entries:

```
I/O Graphs ...
Reset
```

Select the `Reset` entry to reset the internal counters (such as maximum I/O rate), clear the speedometers and the bar diagram, make the entire configuration tree visible, and select the `System` node.

Select the `I/O Graphs` entry to bring up a `graph console` that allows you to select detailed statistics about individual nodes. For disk nodes and non-cached logical devices, two graphs are available:

```
<NAME>:nrds
<NAME>:nwrts
```

For cached devices, four graphs are available:

```
<NAME>:crds
<NAME>:cwrts
<NAME>:drds
<NAME>:dwrts
```

The extensions have the following meaning:

Extension	Meaning
`nrds`	Number of blocks read
`nwrts`	Number of blocks written
`crds`	Number of blocks read from cache to user
`cwrts`	Number of blocks written from user to cache
`drds`	Number of blocks read from disk to cache
`dwrts`	Number of blocks written from cache to disk

7.3 Cray System Activity Reporting (`tsar`(8)) Package

You can use the `sdc`(8) and `tsar`(8) commands together to gather and to report system activity data.

Information about the system activity reporting package is divided into the following areas:

- `sdc` command
- `tsar` command
- Data collection
- Data file format

- `tsar` source scripts
- `tsar` language description
- Operational setup
- Examples
- Limitations

The functionality of `sdc` and `tsar` is similar to that of the `sadc` and `sar` commands. The `sdc` and `tsar` combination offers the following advantages:

- You can specify how much data is to be collected, either all of the `sdc` data or a subset. If you are interested only in CPU utilization, then you can direct `sdc` to gather only CPU data.
- Data can be summarized in a manner that is appropriate for your site. `tsar` report formats are not hard coded. Reports are formatted according to the directives in `tsar` source scripts, which you write.
- `sdc` data files are portable from one Cray platform to another. The files contain headers that describe the data records.

The current release has limitations on its implementation. See Section 7.3.10, page 352, for more information.

7.3.1 sdc(8) Command

The `sdc`(8) command gathers system activity data from the kernel by generating a C program and an executable, `sdcx`. The `sdc` process creates a new session and calls the `fork`(2) system call to create a child process, which executes `sdcx`. The child process performs the actual data collection while the parent `sdc` process exits.

The `sdc` command can collect the following types of data:

- CPU utilization
- System calls
- Process management
- Memory management
- System table management
- System I/O

- General system data
- Disk activity
- Tape activity
- TCP/IP activity
- Terminal activity
- NFS activity
- Network interface activity
- IPC activity

Tables in this section list the available data items for each of these types.

For more detailed information on the command see the `sdc`(8) man page.

7.3.2 `tsar`(8) Command

The `tsar`(8) command formats the data collected by the `sdc` command according to user-specified directives. The directives are placed in source scripts, which `tsar` processes.

The `tsar` utility is a translator that processes a subset of the `awk` language, described in *The AWK Programming Language*, by Alfred Aho, Brian Kernighan, and Peter Weinberger.

For more detailed information on the command see the `tsar`(8) man page.

7.3.3 Data Collection

By default, the `sdc` command collects data for over 100 system activity counters. You can sample a subset of these counters by invoking `sdc` with the `-R` option. The system activity data types and the data items (or counters) available for each type are described in tables in this section; the four column heads are defined as follows:

Column Head	Meaning
Name	The name that `tsar` and `sdc` use for the data item. This name should appear in the request file when you use the `sdc -R` option.

Cum	As shown in the tables, each data item is either cumulative (Y) or not (N).	
	Y	The value for this item is cumulative. Thus, to determine the item's value for an interval, the value from the previous record must be subtracted from the current record. This type of counter generally is initialized at boot time.
	N	The value for this item is not cumulative.
Unit	The unit, if any, in which the data item is reported.	

7.3.3.1 CPU Data

The data shown in Table 34 is available for each CPU. Each item, except `ncpus`, is an array indexed by the CPU number. Each array has `ncpus` entries.

Table 34. CPU Data

Name	Cum	Unit	Description
`ncpus`	N		Number of CPUs configured
`cpuuser`	Y	clocks	CPU time in user mode
`cpuUNIX`	Y	clocks	CPU time in kernel mode
`cpuidle`	Y	clocks	CPU idle time
`cpupre`	Y		Number of program range errors
`cpuore`	Y		Number of operand range errors
`cpuerr`	Y		Number of error exchanges
`cpufpi`	Y		Number of floating point errors
`cpudli`	Y		Number of deadlock errors
`cpurpe`	Y		Number of register parity errors

7.3.3.2 System Calls

The data shown in Table 35 is available for each system call. Each item, except `nsysc`, is an array indexed by the system call name prefixed by `sc_` (such as `sc_fork`, `sc_exec`, and `sc_read`) or by 0 through (`nsysc` -1). Each array has `nsysc` entries.

Table 35. System Call Data

Name	Cum	Unit	Description
`nsysc`	N		Number of system calls
`sc_name`	N	ASCII	Name of system calls
`scall`	Y		Number of requests
`scalltime`	Y	clocks	Total time used by system call
`scallmax`	N	clocks	Maximum time to complete
`scallmin`	N	clocks	Minimum time to complete

7.3.3.3 Process Management

The data shown in Table 36 is available to monitor process management.

Table 36. Process Management Data

Name	Cum	Description
`pswitch`	Y	Number of process switches
`punlock`	Y	Number of times a process is unlocked
`runocc`	Y	Number of times `runque` was updated
`runque`	Y	Length of the run queue
`srunwait`	Y	Number of times processes were loaded and runnable but not running
`srunwaitproc`	Y	Number of processes loaded and runnable but not running

7.3.3.4 Memory Management

The data shown in Table 37 is available to monitor memory management.

Table 37. Memory Management Data

Name	Cum	Unit	Description
`bswapin`	Y		Number of blocks swapped in
`bswapout`	Y		Number of blocks swapped out
`datlock`	Y		Number of data locks
`memlock`	N	clicks	Amount of memory locked
`shuffle`	N		Number of shuffles in memory
`swap`	N	swap allocation	Total amount of swap space units
`swapavail`	N	swap allocation	Amount of available swap space units
`swapin`	Y		Number of swap ins
`swapout`	Y		Number of swap outs
`swpocc`	Y		Number of times `swpque` was updated
`swpque`	Y		Length of the swap queue
`txtlock`	Y		Number of text locks
`umem`	N	words	Amount of memory available to user processes
`umemuse`	N	clicks	Amount of memory in use by user processes
`xswapin`	Y		Number of times a shared-text process was swapped in
`xswapout`	Y		Number of times a shared-text process was swapped out

7.3.3.5 System Table Management

The data shown in Table 38 is available to monitor system table management activity.

Table 38. System Table Management Data

Name	Cum	Description
`file`	N	Number of active entries in the file table
`file_sz`	N	Size of the file table
`file_ov`	N	Number of overflows in the file table
`nc1inode`	N	Number of active entries in the `nc1inode` table
`nc1inode_sz`	N	Size of the `nc1inode` table
`nc1inode_ov`	N	Number of overflows in the `nc1inode` table
`proc`	N	Number of active entries in the `proc` table
`proc_sz`	N	Size of the `proc` table
`proc_ov`	N	Number of overflows in the `proc` table
`text`	N	Number of active entries in the `text` table
`text_sz`	N	Size of the `text` table
`text_ov`	N	Number of overflows in the `text` table

7.3.3.6 System I/O, General

The data shown in Table 39 is available to monitor general system I/O.

Table 39. General System I/O

Name	Cum	Unit	Description - Number of
`arblks`	Y	blocks	Blocks read by `aread` ()
`aread`	Y		`aread` () calls
`awblks`	Y	blocks	Blocks written by `awrite` ()
`awrite`	Y		`awrite` () calls
`bdrblks`	Y	blocks	Buffer blocks read from devices
`bdread`	Y		`bread` () calls actually read from devices
`bdwblks`	Y	blocks	Buffer blocks written to devices

Name	Cum	Unit	Description - Number of
`bdwrite`	Y		`bwrite` () calls - buffer writes to devices
`burblks`	Y	blocks	`lread` blocks read - buffer blocks read to user
`buread`	Y	blocks	`aread` () and `bread` () requests - buffer reads to user
`buwblks`	Y		`lread` blocks written - buffer blocks written from user
`buwrite`	Y		`awrite` () and `bwrite` () requests - buffer writes from user
`pktin`	Y		I/O input packets processed
`pktout`	Y		I/O output packets sent
`pktbad`	Y		I/O illegal packets received
`prblks`	Y	blocks	Raw (`physio` ()) blocks read
`pread`	Y		Raw (`physio` ()) reads
`pwblk`	Y	blocks	Raw (`physio` ()) blocks written
`pwrite`	Y		Raw (`physio` ()) writes
`rchar`	Y	bytes	Bytes read by `read`(2), `reada`(2), `listio`(2)
`wchar`	Y	bytes	Bytes written by `write`(2), `writea`(2), `listio`(2)

7.3.3.7 System I/O, Caching

The data shown in Table 40 is available to monitor caching. Each item, except `nldds`, is an array indexed by the logical device name prefixed by `ld_` (such as `ld_root` and `ld_usr`) or by through (`nldds` -1). Each array has `nldds` entries.

Table 40. System I/O - Caching

Name	Cum	Unit	Description
`nldds`	N		Number of logical devices
`ld_active`	N		Number of current ldcache requests
`ld_agelo`	N	clocks	Trickle sync lower threshold

Name	Cum	Unit	Description
`ld_ageup`	N	clocks	Trickle sync upper threshold
`ld_bread`	Y	blocks	Number of blocks read from device
`ld_bwrite`	Y	blocks	Number of blocks written to device
`ld_dirthi`	N	cache units	Dirty unit high-water mark
`ld_dirtlo`	N	cache units	Dirty unit low-water mark
`ld_dirtpd`	N	cache units	Number of dirty units being synced
`ld_dirty`	N	cache units	Number of dirty units
`ld_dread`	Y	blocks	Number of reads from disk to cache
`ld_dwrite`	Y	blocks	Number of writes from cache to disk
`ld_name`	N	ASCII	Logical device name
`ld_nch`	N		Number of cache units
`ld_rhit`	Y		Number of read hits on ldcache
`ld_rmiss`	Y		Number of read misses on ldcache
`ld_rreq`	Y		Number of ldcache read requests
`ld_sch`	N	blocks	Size of each cache unit
`ld_uread`	Y	blocks	Number of reads from cache to user
`ld_uwrite`	Y	blocks	Number of writes from user to cache
`ld_whit`	Y		Number of write hits on ldcache
`ld_wmiss`	Y		Number of write misses on ldcache
`ld_wreq`	Y		Number of ldcache write requests

7.3.3.8 General System Data

The data shown in Table 41 is available to monitor other system activity.

Table 41. General System Data

Name	Cum	Unit	Description
`CLK_TCK`	N	clocks	Number of clock ticks per second
`HARDWARE`	N	ASCII	Machine type
`MACHINE`	N	ASCII	Machine identification
`MEMORY`	N	ASCII	Memory configuration
`NODENAME`	N	ASCII	Network node name
`RELEASE`	N	ASCII	UNICOS release
`SOFTWARE`	N	ASCII	Software release
`SYSNAME`	N	ASCII	System name
`VERSION`	N	ASCII	UNICOS version
`dirblk`	Y	blocks	Number of directory blocks read by `c1namei` ()
`iget`	Y		Number of `iget` () function calls
`ios_e`	N		Set if the system has an IOS model E
`namei`	Y		Number of `namei` () function calls
`nwpc`	N	words	Number of words per click
`semlockc`	Y		Number of locked events
`semlockt`	Y	clocks	Time spent waiting on a semaphore
`swapau`	N	clicks	Swap allocation units in clicks
`boottime`	Y	seconds	Time at which `sdc -B` was executed
`headertime`	Y	seconds	Time at which a header record was written
`shuttime`	Y	seconds	Time at which `sdc -S` was executed
`time`	Y	seconds	Time at which the record was written

In this table, `boottime`, `headertime`, `shuttime`, and `time` are in seconds since 00:00:00 GMT, January 1, 1970.

7.3.3.9 Disk Data

A set of data items (system activity counters) is available for the IOS model E (IOS-E).

The amount of I/O transferred is reported in sectors, the basic I/O unit for a given disk device type. On the IOS-E, a sector is not always a block. For example, DD-XX devices have a sector size of 2048 words (4 blocks), while DD-ZZ devices have 512 words per sector.

The data shown in Table 42 is available for each IOS-E physical disk. Each item, except `npdds`, is an array indexed by the physical disk number prefixed by `pd_` (such as `pd_1334_03`) or by 0 through (`npdds` -1). Each array has `npdds` entries.

Table 42. IOS-E Physical Disk Data

Name	Cum	Unit	Description
`npdds`	N		Number of physical disk devices
`dr_iotime`	Y	clocks	Actual I/O read time
`dr_rerrs`	Y		Number of recovered read errors
`dr_uerrs`	Y		Number of unrecovered read errors
`dr_rectime`	Y	clocks	Read recovery time
`dw_nqreqs`	Y		Number of queued write requests
`dw_qtime`	Y	clocks	Time waiting in the write queue
`dw_nioreqs`	Y		Number of I/O write requests to the IOS-E
`dw_nblks`	Y	sectors	Number of write sectors transferred
`dw_iotime`	Y	clocks	Actual I/O write time
`dw_rerrs`	Y		Number of recovered write errors
`dw_uerrs`	Y		Number of unrecovered write errors
`dw_rectime`	Y	clocks	Write recovery time

7.3.3.10 Tape Data

The data shown in Table 43 is available for each tape drive. Each item, except `ntpds`, is an array indexed by the physical tape number prefixed by `tp_` (such as `tp_170`) or by 0 through (`ntpds` -1). Each array has `ntpds` entries.

Table 43. Tape Drive Data

Name	Cum	Unit	Description
`ntpds`	N		Number of tape drives
`tp_name`	N	ASCII	Name of tape drive
`tp_mounts`	Y		Number of volumes mounted
`tp_nread`	Y	bytes	Number of bytes read
`tp_nwrite`	Y	bytes	Number of bytes written

7.3.3.11 TCP/IP Data

The data shown in Table 44 is available for analyzing TCP/IP performance.

Table 44. TCP/IP Performance Data

Name	Cum	Unit	Description
`tcp_hwint`	Y	clocks	Time spent on TCP/IP hardware interrupts
`tcp_swint`	Y	clocks	Time spent on TCP/IP software interrupts
`tcp_scall`	Y	clocks	Time spent on TCP/IP system calls

7.3.3.12 Terminal Data

The data shown in Table 45 is available for analyzing terminal activity.

Table 45. Terminal Activity Data

Name	Cum	Description
`canch`	Y	Number of canonical characters input through tty interface
`clist`	N	Number of `clist` entries in use
`clist_ov`	Y	Number of overflows in the `clist` buffer
`outch`	Y	Number of characters output through tty interface
`rawch`	Y	Number of raw characters input through tty interface
`rcvint`	Y	Number of T (terminal) packet input interrupts
`xmtint`	Y	Number of T (terminal) packet output interrupts

7.3.3.13 NFS Data

The data shown in Table 46 is available for analyzing network file system (NFS) server and client activity. Each item, except `nnfsc`, is an array indexed by the NFS call index (0 through `nnfsc` -1).

Table 46. NFS Data

Name	Cum	Unit	Description
`nfsc_name`	N	ASCII	Name of NFS call
`nfscl_calls`	Y		Number of client calls
`nfssv_calls`	Y		Number of server calls
`nnfsc`	N		Number of NFS call types

7.3.3.14 Network Interface Data

The data shown in Table 47 is available for analyzing activity for each network interface that has been configured. Each item, except `nnet`, is an array indexed by network (0 through `nnet` -1).

Table 47. Network Interface Data

Name	Cum	Unit	Description
`nw_colls`	Y		Number of collisions
`nw_ierrs`	Y		Number of input errors
`nw_ipkts`	Y		Number of input packets
`nw_name`	N	ASCII	Internet name (address) of the network
`nw_oerrs`	Y		Number of output errors
`nw_opkts`	Y		Number of output packets
`nnw`	N		Number of configured network interfaces

7.3.3.15 IPC Data

The data shown in Table 48 is available for analyzing IPC activity.

Table 48. IPC Activity Data

Name	Cum	Description
`msgsend`	Y	Number of IPC messages sent
`msgrecv`	Y	Number of IPC messages received
`semops`	Y	Number of semaphore operations
`shmat`	Y	Number of shared memory attaches
`shmdt`	Y	Number of shared memory detaches

7.3.3.16 Restricted Data Collection

As mentioned in Section 7.3.3, page 324, a subset of the data in the system activity counters can be sampled by using the `sdc -R` option. You must specify the name of a request file, which contains a list of the data to be collected.

Each request file must include at least the following data items: `boottime`, `headertime`, `shuttime`, and `time`. These items are needed to determine when the data was sampled and whether there was a system boot or shutdown.

For example, only data for memory and swap usage will be gathered when `sdc` is executed with a request file that contains these items:

```
boottime
headertime
shuttime
time
HARDWARE
MEMORY
umem
nwpc
swap
swapau
swapavail
umemuse
memlock
```

The `tsar` script `/usr/src/prod/admin/uma/sar.t/M` formats the data into a report.

7.3.4 Data File Format

The system activity data file, created by the `sdc`(8) command, consists of header and data records. A data record is created for each time interval that is sampled. The data records are preceded by two ASCII header records, which define the data records.

7.3.4.1 Header Records

There must always be header records at the beginning of the data file. Two ASCII header records define the data being collected: a definitions record and a meta-data record.

The purpose of the header records is to label the data in the data records that follow. If the header records do not correspond with the data records, `tsar` will be unable to process the data. Header records must be placed at the following locations.

- The beginning of the data file.
- When the system has been reconfigured and the reconfiguration affects the data that `sdc` is sampling.
- When a subset of the data listed in Section 7.3.3, page 324, is sampled.

When `sdc` is invoked with either the `-B` or `-S` option, header records also contain the system boot time and shutdown time.

7.3.4.2 Definitions Record

The first header record written to the data file is the data definitions record. This record contains system call names, device names, kernel table sizes, and ASCII strings that describe the system configuration. When processing the data file, `tsar` 8* updates the data definitions each time it encounters a definitions record.

7.3.4.3 Meta-data Record

The second header record written to the data file is the meta-data record. This record contains the name and size of each item or array that is in the actual data record. When processing the data file, `tsar`(8) updates the meta-data definitions each time it encounters a meta-data record.

7.3.4.4 Data Records

Data records are written each time `sdc` samples the system activity counters. Each set of data records must be preceded by a definitions and a meta-data header record.

There are two types of data records. The first type contains non-ldcache data. If ldcache statistics are requested, the second type of data record is generated. Each ldcache data record is immediately preceded by a definitions record and a meta-data record, which together define and describe the ldcache data record.

Table 49 describes which `sdc`, `sdcx`, and `tsar` options produce header records or data records. When the table shows that one header record is written, it means that both a definition and a meta-data record are written.

Table 49. Records Written

Command	Option	Record written
`sdc`	`-c`	One header record.
`sdc`	`-B`	One header record containing the boot time.

Command	Option	Record written
`sdc`	`-R`	One header record. If `-c`, `-B`, and `-S` are not also specified, data records are written.
`sdc`	`-S`	One header record containing the shutdown time.
`sdc`	none of the above	One header record and data records.
`sdcx`	any	Data records.
`tsar`	`-h`	One header record and data records.

⚠ **Caution:** The `tsar`(8) command assumes that the header records define the data records that follow them. If a system reconfiguration occurs, and no header record is written to define the reconfiguration, then the `tsar` output may be erroneous or `tsar` may abort.

7.3.5 `tsar`(8) Modes

The `tsar`(8) command operates in one of three modes: compilation-only mode, online mode, or playback mode.

7.3.5.1 Compilation-only Mode

The `tsar -c` option places `tsar` in compilation-only mode. This mode is used to debug `tsar` source scripts. `tsar` compiles the scripts but does not execute them. The system activity data is not formatted into an ASCII report.

7.3.5.2 Online Mode

The `tsar -h` option places `tsar` in online mode. In this mode, `tsar` collects data from the host system through the `sdc`(8) command. The data can be written to a file and/or formatted into an ASCII report as it is being collected.

7.3.5.3 Playback Mode

The `tsar -p` option places `tsar` in playback mode; this also can be done by not specifying the `-h` option. In this mode, `tsar` processes data from an existing system activity data file. This is the default `tsar` mode.

7.3.6 `tsar`(8) Source Scripts

The `tsar`(8) command is a translator that formats system activity data into an ASCII report according to the directives found in a source script.

The `tsar` scripts consist of six sections, including the body, any of which may be empty or missing. Scripts can contain any of the following sections, in any order:

`BEGIN` *{ statements }*
`END` *{ statements }*
`RESTART` *{ statements }*
`RECONFIG` *{ statements }*
`function` *name (arglist) { statements }*
statements

The `tsar` command can process multiple source scripts, and one script can contain multiple `BEGIN`, `END`, `RESTART`, `RECONFIG`, `function`, or body sections. In these cases, `tsar` executes the statements for all like sections in the order that they appear in the scripts or script.

For example, all statements in the various `BEGIN` sections will be combined into one `BEGIN` section. The statements will be in the same order as they appear in the scripts or script.

7.3.6.1 `BEGIN` Section

The statements associated with `BEGIN` comprise the preamble. The preamble is executed once after the first definition and meta-data records are read. The preamble can be used to print report headings and to initialize variables used in the body.

7.3.6.2 `END` Section

The statements associated with `END` comprise the postamble. The postamble is executed once after all the records in the data file have been read. You can instruct `tsar` in this section to process and print summary data.

7.3.6.3 `RESTART` Section

The statements associated with `RESTART` comprise the restart section. These statements are executed each time a system boot or shutdown is found in the header record. The amount of time the system was up or down can be calculated in this section.

7.3.6.4 `RECONFIG` Section

The statements associated with `RECONFIG` comprise the reconfiguration section. These statements are executed each time a system reconfiguration or a change in which data items are sampled is detected in the header records.

> **Note:** When a restart or reconfiguration is detected, `tsar` assumes that all of the system activity counters have been reset. For example, the built-in function `sdiff` (`x`) calculates the difference between the current sample of `x` and the first sample of `x` in this boot or reconfiguration period. It does not calculate the difference between the current sample and the first sample found in the data file.

7.3.6.5 `function` Section

The statements in the `function` section of `tsar` define functions as specified by the user. Functions always begin with the word `function` followed by the function name and the argument list. The *arglist* consists of names separated by commas. These argument names are the formal parameters of the function and the variables that are local to the function. Function calls may be nested and recursive. The return statement can be used to return a value.

7.3.6.6 Body

Statements that are not in any of the preceding sections form the body of the `tsar` source script. Typically, these statements calculate the usage for each interval. This section is executed once for each data record encountered.

7.3.6.7 Example Source Scripts

Examples of `tsar` source scripts can be found in the `/usr/src/prod/admin/uma/sar.t` directory. These scripts produce output that is similar to `sar`(8) output. There is one script for each `sar` option. The name of the `tsar` script is the same as in the `sar` option.

7.3.7 `tsar`(8) Language Description

The `tsar` language is the action language of `nawk` without the string processing operations. Users familiar with `nawk` will have little difficulty writing and understanding `tsar` scripts. The pattern part of `nawk` is unnecessary in `tsar`, because the data format is defined in the data file. Users merely select the data items to process by name.

`tsar` implements a subset of the `awk` language described in *The AWK Programming Language*, by Alfred Aho, Brian Kernighan, and Peter Weinberger.

7.3.7.1 Statements

A `tsar` script can include any of the following statements:

`if (` *expression* `)` *statement* `[ else` *statement* `]`
`while (` *expression* `)` *statement*
`do` *statement* `while (` *expression* `)`
`for (` *expression*`;` *expression*`;` *expression* `)` *statement*
`break`
`continue`
`{ [` *statements* `] }`
expression
`print` *expression-list* `[ >`*expression* `]`
`printf` *format*`[,` *expression-list* `] [ >`*expression* `]`
`next`
`exit [` *expression* `]`
`return [` *expression* `]`

The following explains further the contents of statements:

- Statement Terminators. Statements are terminated by semicolons, right braces, or newline characters.
- Statement Continuation. Statements can be continued on successive lines by using \ as the last character of the line. Statements can also be continued after the following symbols:

```
,       (comma)
{       (left brace)
&&      (logical AND)
||      (logical OR)

do
else)       (right parenthesis in an "if" or "for" statement)
```

- Comments. Nonexecutable comments begin with # and end with a newline character. They can appear anywhere in the source script.
- Expressions. Expressions include constants, variables, and operators. Parentheses can be used to control the grouping of the operations in an expression.

- Logical Expressions. Logical expressions have a value of 1 (true) and 0 (false). As in the C language, any nonzero value is taken to be true.
- Numbers. Numbers can be integers or floating points. The format is the same as that recognized by `strtod` and `strtol`: digits, decimal point, digits, `e` or `E`, signed exponent. At least one digit or a decimal point must be present; the other components are optional. Octal integers begin with 0. Hexadecimal integers begin with 0x.
- Variable Names. Variable names consist of a letter followed by a string of letters, numbers, or the character _. Variables are used to name the data items found in the data records of the system activity file.

  Some variables in the system activity data file are arrays. The elements of these arrays can be referenced by indexing. For example, the variable, `cpuuser`, is an array that contains the CPU time in user mode (see Section 7.3.3, page 324). The CPU time for CPU 0 is referenced by `cpuuser` [0].

  Users can also define additional variables within the `tsar` source script; however, user-defined arrays are not supported.

7.3.7.2 Operators

Prefix, infix, and suffix operators are available for use in `tsar` scripts.

Prefix Operators

The `tsar` command applies a prefix operator immediately preceding a term and any suffix operators. It then applies any prefix operators to the left of that operator, grouping them from right to left.

Operator	Action
`++X`	Preincrement
`--X`	Predecrement
`+X`	Plus
`-X`	Minus
`!X`	Logical `NOT`

Infix Operators

The `tsar` command applies `infix` operators, in descending order of precedence, as follows:

Operator	Action
`X^Y`	Exponentiation
`X*Y`	Multiplication
`X/Y`	Division
`X%Y`	Remainder
`X+Y`	Addition
`X-Y`	Subtraction
`X<Y`	Less than
`X<=Y`	Less than or equal
`X>Y`	Greater than
`X>=Y`	Greater than or equal
`X==Y`	Equals
`X!=Y`	Not equals
`X&&Y`	Logical `AND`
`X||Y`	Logical `OR`
`Z?X:Y`	Conditional
`X=Y`	Assignment
`X*=Y`	Multiply assign
`X/=Y`	Divide assign
`X%=Y`	Remainder assign
`X+=Y`	Add assign
`X-=Y`	Subtract assign
`X,Y`	Comma

Suffix Operators

The `tsar` command applies a suffix operator immediately following a term before it applies any other operator. It then applies any suffix operators to the right of that operator, grouping them from left to right. The following list shows the suffix operators:

Operator	Action
`X++`	Postincrement
`X--`	Postdecrement

`X[Y]`	Subscript
`X(Y)`	Function call

7.3.7.3 Built-in Functions

The `tsar` command has the following built-in functions, with the function parameters (given in parentheses) defined at the end of the list:

Function name	Description
`abs`(*exp*)	Return the absolute value of *exp*.
`diff`(*term*)	Return the difference between the current and previous sample values of *term*. If *term* is an array name, then `diff` () returns the sum of the differences of the array elements.
`close`(*str*)	Close the file stream specified by *str*.
`frac`(*exp*)	Return the fractional part of *exp*.
`imax`(*arr*)	Return the index of the maximum element of array *arr*.
`imin`(*arr*)	Return the index of the minimum element of array *arr*.
`int`(*exp*)	Return the integer part of *exp*.
`isdefined`(*sym*)	Return 1 if *sym* is defined. Otherwise return 0.
`rate`(*exp*)	Return the rate of *exp* in counts per second.
`sdiff`(*term*)	Return the difference in the value of *term* between the present and first sample in this boot or reconfiguration period. If *term* is an array name, `sdiff` () returns the sum of the differences of the array elements.
`sum`(*arr*)	Return the sum of the elements in array *arr*.
`system`(*str*)	Pass *str* to the shell for execution.
`strftime`(*fmt*)	Format the time into a string according to *fmt*.
`timen`(*val*)	Return the time stamp in seconds, minutes, or hours. The time stamp is normally represented as `hh:mm:ss`. `timen` (0) returns the time stamp in

seconds, which is `t = hh(3600) + mm(60) + ss.` `timen` (1) returns minutes, which is calculated as `t/60`. `timen` (2) returns hours, which is `t/3600`.

The function parameters are as follows:

arr — *arr* is an array name. For example:

`imax` (`ld_active`)

exp — *exp* is a variable name or a function invocation. For example:

`abs` (`runque`)

variable name

`rate` (`diff` (`scall` [`sc_reada`]))

function invocation

fmt — *fmt* is `NULL` or a valid `strftime` format, which is enclosed in double quotes. For example:

`strftime` ()

`ULL` format

`strftime` (" `%X` ")

`strftime` format

str — *str* is either a character string enclosed in double quotes or the name of a variable whose value is a character string. For example:

`close` (" `disk.data` ")

character string

command = `"date; uname -a"`

system(*command*)

variable which contains a character string

sym — *sym* is a variable name. Names of array elements are not valid symbols. *sym* can be defined by the `tsar -D` option. For example:

`if` (`isdefined` (`ios_e`))

variable

`if` (`isdefined` (`ld_active` [`ld_root`]))

invalid

`tsar -DCPU`

`if` (`isdefined` (`CPU`))

symbol defined by the `tsar -D` option

term *term* is an expression (*exp*) or an array name. For example:

`diff` (`scall` [`sc_reada`])

expression

`diff` (`tp_mounts`)

array name

val *val* is 0, 1, or 2. For example:

`timen` (0)	Returns the time in seconds
`timen` (1)	Returns the time in minutes
`timen` (2)	Returns the time in hours

7.3.7.4 Built-in Variables

The `tsar` command has the built-in variables shown in Table 50.

Table 50. `tsar` Command Built-in Variables

Variable	Description	Default value
`END_TIME`	Ending time of the sampling period	None
`NR`	Number of sample intervals	None
`OFMT`	Output format for printing numbers	`%.6g`
`OFS`	Output field separator	" "

Variable	Description	Default value
ORS	Output record separator	\n
RSIZE	Size of the data records in bytes	None
START_TIME	Start time of the sampling period	None

7.3.8 Operational Setup

Information about boot and shutdown times can be collected by executing the `sdc`(8) command during the system boot and shutdown process. A `crontab`(1) or `at`(1) job can be used to gather system activity data while the system is running.

Regardless of how data is collected, the `sdc`, `sdcx`, and `tsar -h` commands must be executed by a user who has read access to the `/unicos` and `/dev/kmem` files.

7.3.8.1 Difference between `sdc`(8) and `sdcx`

As explained in Section 7.3.1, page 323, `sdc`(8) generates and compiles `sdcx` (see `sdc`(8)). This process can take a significant amount of wall-clock time on a busy system. Instead of continually generating and compiling `sdcx`, you can use the `sdc -c` option to write a header record and save the `sdcx` binary. The `sdcx` binary can then be used to collect data samples, thus bypassing the need to regenerate and recompile `sdcx`.

The `sdcx` binary only writes data records and not header records, as mentioned in Section 7.3.4.1, page 336. Thus, when a system reconfiguration affects the `sdcx` data samples, a new header record must be written and a new `sdcx` must be generated. Failure to do this can cause `tsar` to create erroneous reports or to be unable to read the data files.

If no system reconfigurations occur during the sampling interval, `sdc` can be executed once to write a header record and to save the `sdcx` binary. `sdcx` can then be used to collect the data samples.

If reconfigurations are likely during the sampling interval, it is always best to execute `sdc`. Each time `sdc` is executed, a header record is written. Data samples also can be collected.

It is important to understand when `sdc` writes the header and data records. Section 7.3.4, page 336, and Section 7.3.8.4, page 348, describe which command options write the various records. Both sections show examples.

7.3.8.2 Boot Time Data

The `sdc` command with the `-B` option must be executed during the system boot process in order for `sdc` to record the boot time. One way of doing this is to invoke `sdc` from the `/etc/rc.pst`. Code similar to the following can be used in `/etc/rc.pst`:

```
if [ -x /usr/bin/sdc ]
then
        /usr/bin/sdc -B /usr/adm/tsar/dcf.‘date +%m%d‘
fi
```

This will write a header record, which contains the boot time, to the file `/usr/adm/tsar/dcf.`*MMDD*, where *MMDD* is the current month and day. No data will be sampled.

If no system reconfigurations are going to occur while the system is running, the `sdc -c` option can also be used in `/etc/rc.pst`. The `-c` option will save the `sdcx` binary in the specified directory. This binary can be executed from a `crontab` file.

7.3.8.3 Shutdown Data

The system shutdown time is recorded by the `sdc -S` option. This command can be invoked from the `/etc/shutdown.pst` file. Code similar to the following can be put in the file:

```
if [ -x /usr/bin/sdc ]
then
        /usr/bin/sdc -S /usr/adm/tsar/dcf.‘date +%m%d‘
fi
```

This will write a header record, which contains the shutdown time, to the file `/usr/adm/tsar/dcf.`*MMDD*, where *MMDD* is the current month and day. No data will be sampled.

The `sdc -B` and `-S` options may take significant wall-clock time to execute on busy systems, as noted in Section 7.3.8.1, page 347, and under the Section 7.3.10, page 352.

7.3.8.4 `crontab`(1) Entries

System activity data can be collected periodically by `sdc`, `sdcx`, or `tsar -h`, through the `crontab`(1) command. The following examples show how this can be done.

Example 1: This example writes a header record (definitions and meta-data records) followed by three data records each hour. Data is sampled at a rate of one sample every 20 minutes. A header and data record is written on the hour. Additional data records are written at 20 and 40 minutes after the hour.

If a system reconfiguration occurs at 09:25, then `tsar` may interpret the 09:40 data record written incorrectly. `tsar` assumes that the 09:40 record has the format described in the 09:00 header record.

```
0 8-17 * * 1-5 /usr/bin/sdc -i 20m -n 3 /usr/adm/tsar/dcf.‘date +%m%d
```

Example 2: This example writes a header and data record every 20 minutes. Because each data record is preceded by its own header record, system reconfigurations will be detected by `tsar`.

```
0,20,40 8-17 * * 1-5 /usr/bin/sdc -n 1 /usr/adm/tsar/dcf.‘date +%m%d
```

Example 3: This example writes a data record every 20 minutes. No header records are written. It is assumed that both a header record and the `sdcx` binary were generated at boot time. (See Section 7.3.8.2, page 348.) System reconfigurations will not be detected by `tsar`, as the data file contains only one header record.

```
0 8-17 * * 1-5 /usr/adm/tsar/sdcx -i 20m -n 3 >> /usr/adm/tsar/dcf.‘date+%m%d
```

7.3.9 Examples

System boot, shutdown, and system activity data can be collected by invoking `sdc` as described in Section 7.3.8.2, page 348, Section 7.3.8.3, page 348, and Section 7.3.8.4, page 348. Data gathering and reporting, however, do not have to be scheduled by `cron`(8).

7.3.9.1 `sdc`(8) Data Collection

Like `sadc` (see `sar`(8)), `sdc` can be executed at any time to collect data. For example, to sample data in the background at 10 minute intervals for 2 hours, you can execute the following:

```
nohup sdc -i 10m -n 12 dcf.‘date +%m%d‘&
```

The equivalent `sadc` command is as follows:

```
nohup sadc 600 12 sa.‘date +%m%d‘&
```

`sdc` data is written to the file `dcf.`*MMDD*, and `sadc` output is directed to `sa.`*MMDD*, where *MMDD* is the current month and day.

The startup time for `sdc` is longer than that for `sadc`, because `sdc` must generate and compile the `sdcx.c` code.

7.3.9.2 `tsar`(8) Data Collection

Like `sar`(8), `tsar` can be used to simultaneously sample and format data. For example, to sample data at 5-minute intervals for 1 hour and report swap activity on a machine named `gust`, you can execute the following:

```
tsar -h gust -i 5m -n 12 -r dcf.‘date +%m%d‘ M
```

In this example, `M` is a copy of a script that is found in the `/usr/src/prod/admin/uma/sar.t` directory.

The equivalent `sar` command is as follows:

```
sar -o sa.‘date +%m%d‘ -M 300 12
```

`sdc` data is written to the file `dcf.`*MMDD*, and `sadc` data is directed to `sa.`*MMDD*, where *MMDD* is the current month and day. The ASCII reports are written to `stdout`.

7.3.9.3 `tsar`(8) Report Formatting

The `tsar` utility allows users to generate ASCII reports in a variety of formats. Users specify the report format in scripts described in Section 7.3.6, page 339, and Section 7.3.7, page 340. This differs from `sar`, in that `sar` report formats are hard coded in the source code and cannot be readily changed by the user.

The `/usr/src/prod/admin/uma/sar.t` directory contains `tsar` scripts which correspond to the various `sar` reports. In addition to mimicking `sar` reports, users can write `tsar` scripts that create reports in a site-specific format or graph the data.

The following `tsar` script graphs the read and write activity of a specified device against the sample time. If the script is named `disk_plot`, and the sample data is in the file `dcf.1007`, then the plot for device `1130_00` is generated by the following command:

```
tsar -Ddisk=\"1130_00\" -p dcf.1007 disk_plot
```

```
#
#       tsar script to plot the total disk i/o of a specified device via
#       xgraph.  The i/o is plotted as the number of sectors read and
#       written vs. the time expressed as hours.
#
```

```
#       The tsar "-D" option is used to specify the device.
#       For example, to graph the i/o for a disk device named 1130_00,
#       you would execute something similar to:
#               tsar -Ddisk=\"1130_00\" -p dcf.1007 disk_plot

BEGIN {
F_ALL_DISKS = "disk.data"  # file with i/o stats for all disk devices
F_XDATA = "xgraph.data"    # file with info needed to plot the graph
F_CMD = "disk.cmd"         # file with grep command to extract specified i/o stats
system("rm -f disk.data xgraph.data disk.cmd")

if (ios_e) {
        nitem = npdds
} else {
        nitem = ndsds
}
first_time = 1
#
#       Setup the titles and limits for the graph.
#
printf("TitleText: %s Total Disk IO on %s  %s\n",
   disk, NODENAME, strftime("%D")) > F_XDATA
print("XUnitText: Time of Day")  > F_XDATA
print("YUnitText: Sectors") > F_XDATA
print("YLowLimit: 0") > F_XDATA
}

#
#       For each sample interval, write the i/o statistics for
#       busy devices to file F_ALL_DISKS
#
if (first_time == 1) {
        printf("XLowLimit: %2.4f\n", timen(2)) > F_XDATA
        first_time = 0
}

for (n = 0; n < nitem; n++) {
        if (diff(dr_nblks[n]) + diff(dw_nblks[n]) != 0) {
                printf("%2.4f %9.0f  ", timen(2),
                   diff(dr_nblks[n]) + diff(dw_nblks[n])) >> F_ALL_DISKS
                if (ios_e) {
                        printf("%s\n", pd_name[n]) >> F_ALL_DISKS
                } else {
                        printf("%s\n", dd_name[n]) >> F_ALL_DISKS
                }
        }
}

END {
#
#       Extract the i/o statistics for the specified device and
#       write them to file F_XDATA.
#
printf("XHighLimit: %2.4f\n", timen(2)) > F_XDATA
print("\"\"") > F_XDATA
print("  ") > F_XDATA
printf("grep ’%s’ %s >> %s\n", disk, F_ALL_DISKS, F_XDATA) > F_CMD

close(F_ALL_DISKS)
close(F_XDATA)
close(F_CMD)
system("chmod 755 disk.cmd")
system("./disk.cmd")

#
#       Plot the i/o via xgraph.
#
system("/usr/bin/X11/xgraph -title ’Total Disk IO’ -tk xgraph.data")
system("rm -f disk.data xgraph.data disk.cmd")
}
```

7.3.10 Limitations

The following are descriptions of limitations to use of the `sdc` and `tsar` commands.

Large system activity data file

The `sdc` output file (system activity data file) may be much larger than the equivalent `sar`(8) data file. The size of the `sdc` output file can be controlled by sampling a subset of the system activity counters through the `sdc -R` option. Ldcache and disk data can require an enormous amount of disk space, so sample them only when necessary.

Undetected ldcache or disk reconfigurations

If either ldcache or disks have been reconfigured, a header record must be written to the system activity file. If a header record is not written, `tsar` may report erroneous ldcache or disk information, or `tsar` may abort.

Long `sdc`(8) execution time

The `sdc` command always generates and compiles `sdcx`. Thus, when you specify the `-B` or `-S` options to `sdc`, it may take a while for them to complete on a busy system. This can be particularly true when `sdc -S` is executed from the system shutdown script.

`imin`() examines unused array entries

Arrays such as `dr_nblks`, `ld_active`, and `tp_mounts` often contain unused entries. `imin`() examines these entries and may return an index to them, because unused array entries contain 0.

No user-defined arrays

`tsar` does not support user-defined arrays. Therefore, it may be difficult to save or sum array elements across restarts or configurations.

No string comparison functions

Built-in `tsar` string comparison functions are not currently available.

Limited access to the first data record

Data from the first record in each boot or reconfiguration period cannot be accessed except through some of the built-in functions.

No `sdcx`(8) record locking (see `sdc`(8))

The `sdcx` binary does not lock records when writing output. Thus, if multiple `sdcx` or `sdc` processes write to the same output file, the output could become corrupt.

No support for foreign file systems

The `sdc` command locks the records in the output file before writing the output. Because record locking is not supported on foreign file systems (such as network file systems), `sdc` cannot write to a file that is on such a file system.

No support for `tsar`(8) running on the SunOS operating system

The `tsar` command may not run on the SunOS Release 4.1 operating system. The system must have an ANSI C compiler (`acc`) to run `tsar`. This feature is not supported in the current release.

Disk I/O units may differ

The `sdc` command records disk I/O in units of sectors, but the size of a sector differs according to the disk device type. (See Section 7.3.3.9, page 332.) The device type is not available through `sdc`, so `tsar` is unable to adjust the data to a common unit.

7.4 Disk Usage Monitoring

One of the most important responsibilities of the system administrator is to monitor system disk usage and to ensure that users have sufficient free space on their file systems to accomplish their work.

The following commands are useful in monitoring disk use:

Command	Description
`diskusg`(8)	Summarizes the disk usage on a file system by file ownership and identifies users who are using most of the space on a file system. The `/usr/lib/acct/diskusg` command is the preferred command for summarizing disk use. It is faster and more accurate than `du`(1) because it bypasses the file system software in the kernel.
`du`(1)	Provides a summary of the disk use on a file system, by directory structure. The `-s` option provides the total number of disk blocks used under each directory (or file) named. For example, if user

accounts are stored on the /u file system, the following lists the total number of blocks used by each account on the file system:

```
cd /u
du -s *
```

The output of du(1) may be piped into sort(1), as follows, providing a sorted list of directories that use the most disk space (allowing you to identify users who are using the most disk space):

```
du -s * | sort +nr
```

This command is useful for locating the points at which users are using most of the file system's disk space and, in conjunction with diskusg, identifying users who may own files located in directory subtrees other than their home directories. Results from the du(1) program can be affected by changes to the file system during execution of du.

find(1) Identifies large files in an emergency when disk space is scarce; however, find does not identify temporary files whose directory links are removed. The following command line, when executed from the topmost directory of a file system, lists all files on file systems that are larger than 1 Mbyte, sorting them in order of size:

```
find . -size +1000000c -print |
  xargs ls -li | sort +5nr
```

With such a list, you can look for a few large files that are likely candidates for deletion because they have not been modified or used for a specified length of time (this value is site-dependent, but is usually a month), because the owner no longer needs the file, or because the file is unreasonably large.

Similarly, the find command can look for all files that have not been modified or used within a certain span of time (for example, the last 3 months).

To free disk space, you may find it appropriate to perform one of the following functions:

- Archive such files to tape for permanent storage.
- Archive such files on tape for eventual deletion, perhaps using a set of four tapes rotated on a weekly basis. This would

allow a user 1 month to ask for retrieval of an archived file before its tape is reused.

- Delete such files.
- You can use the `crontab`(1) command to execute `find` regularly during off-peak hours in order to generate a daily list of large, old files that can be archived to tape by operators.

Unified Resource Manager (URM) [8]

The Unified Resource Manager (URM) is a job scheduler that balances the demands of both batch and interactive sessions. URM provides a high-level method of controlling the allocation of system resources to run jobs that originated either in batch mode or in an interactive session.

Note: In this section, the term *job* is used to identify the scope of the object being managed by URM. For purposes of this discussion, an interactive session is considered a job.

Controlled system resources include CPU time, memory, tape devices, number of jobs, and Secondary Data Segment (SDS) allocation. The URM feature is common among all Cray architectures, but the managed resources are tailored to the hardware configuration on which URM is installed.

Prior to the introduction of URM, the UNICOS operating system included separate facilities for scheduling batch jobs and interactive sessions. Batch jobs were scheduled by the Network Queuing System (NQS). Interactive sessions were scheduled by transient session initiators, such as `login` and `rsh`. However, URM takes input from all session initiators, including NQS and the transient session initiators.

URM combines the management of machine resources under a single umbrella, making it possible to provide consistent treatment of resource demands arriving from these various sources. In addition to monitoring resource demands, URM also monitors the following:

- History of system usage. By monitoring the UNICOS fair-share scheduler, URM monitors past use of important system resources by each user.
- Current system load. URM monitors current use of all important system resources.
- Future work backlog. URM predicts future demands on all important system resources, as indicated by the job backlog and the amount of work in process and waiting to be initiated.
- Target loads. URM manages memory oversubscription, number of active processes, SDS oversubscription, tape usage, and the number of active batch and interactive jobs.

Using this information, URM evaluates requests to initiate jobs. Requests arrive from such UNICOS service providers as NQS (for batch) and `login` (for interactive). Following the selection process, URM sends to each service provider a list of jobs that URM recommends for initiation.

URM does not actually initiate jobs. The service providers retain full responsibility for and control of jobs within their scope. Jobs are not required to be processed through the selection server portion of URM. This is allowed in order to retain command execution capability if the system is behaving improperly. Under normal operation, service providers use the URM selection server and follow the job initiation recommendations of URM.

The Unified Resource Manager is described in detail in the following sections:

- URM features
- Summary of URM commands
- Installing URM
- Configuring URM
- URM administrator tasks
- Troubleshooting URM
- URM architecture
- URM resources
- URM checkpointing
- Tuning URM

8.1 URM Features

The Unified Resource Manager provides the following functionality:

- Provides uniform services for all types of jobs (batch and interactive, for example) and eliminates non-essential differences among them.
- Uses `sdsmgr` to schedule use of SDS space for both batch and interactive sessions.
- Maintains job resource predictions (as offered by the service providers) as well as consumption information to assist the algorithms selecting jobs to recommend for initiation.
- Provides job initiation recommendations to NQS and other service providers.

- Supports SDS oversubscription through job preemption. (The preemption mechanism is suspend / resume.)
- Supports system scheduling on Cray T3D systems by monitoring the load information and job backlog, and by recommending the best job candidates for initiation.
- Controls the number of active jobs. This includes control over the number of jobs of each type as well as the total number of all jobs.
- Uses the information provided by the fair-share scheduler to evaluate a user's potential priority among those competing to have a job initiated.
- Provides a way to report an assessment of machine loading, a list of jobs currently recommended for initiation, and other information upon request.

8.2 Summary of URM Commands

The following UNICOS commands support URM:

Command	Description
`rmgr`(1)	Provides an interface to the URM daemon
`urmd`(8)	Starts the URM daemon
`urmsnap`(8)	Captures the current URM configuration information
`usetjob`(8)	Changes the minimum rank of batch jobs
`ustat`(1)	Displays URM job information

For details of the `rmgr`(1) and `ustat`(1) commands, see the *UNICOS User Commands Reference Manual*. For details of the `urmd`(8), `urmsnap`(8), and `usetjob`(8) commands, see the *UNICOS Administrator Commands Reference Manual*.

8.3 Installing URM

To install URM, use the UNICOS Installation / Configuration Menu System (the *menu system*). For details about the menu system and how to use it, see the following publications:

- *UNICOS System Configuration Using ICMS*
- *UNICOS Installation Menu System Reference Card*

Perform these basic steps to install URM:

1. Verify the automatic startup of the URM daemon
2. Configure the initial URM values
3. Verify the security parameters
4. Enable the service providers

The following sections discuss these steps.

Warning: The following information on configuring URM is not written for a site running a Cray ML-Safe configuration of the UNICOS system. For information on configuring URM for a Cray ML-Safe configuration, see the description of the UNICOS MLS feature in General UNICOS System Administration, publication SG-2301.

8.3.1 Verifying Automatic Startup of URM Daemon

To ensure that the URM daemon (`urmd`) is invoked automatically, verify that the `/etc/config/daemons` file includes a line for `urmd`. If the `/etc/config/daemons` file does not contain a line for `urmd`, you should create one by using the menu system. Traverse the menus by using the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    System daemons configuration ->
      System daemons table ->
```

The final menu selection displays a list of all active system daemons and their attributes, as shown in the following example:

```
     Group  Name       Start Opts   Kill                  Program    >
     -----  ---------  -----------  --------------------  ----------
     SYS1   cron       5=NO:*=YES   *                     /etc/cron
     SYS1   fsdaemon   5=NO:*=YES   *                     /etc/fsmon
     TCP    gated      NO           /etc/gated.pid        /etc/gated
E->  SYS2   urmd       YES          /usr/lib/urm/urmend   /etc/urmd
```

If this table does not include such a line for `urmd`, create a new entry having the following attributes:

```
S->  Group                   SYS2
     Name                    urmd
     Startup at boot time?  YES
     Kill action            /usr/lib/urm/urmend
     Executable pathname    /etc/urmd
```

The meaning of these attributes is as follows:

Attribute	Description
`SYS2`	Places `urmd` in the group of daemons started last.
`urmd`	Names the URM daemon.
`YES`	The URM daemon executes automatically at system startup. If this attribute were `NO`, the daemon would not execute automatically at system startup; however, the daemon could be started manually by using the command `sdaemon urmd`.
`/usr/lib/urm/urmend`	Specifies the command to shut down `urmd` and request interactive checkpointing (if configured).
`/etc/urmd`	Identifies the full path name of the executable for the URM daemon.

To activate this configuration change, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    System daemons configuration ->
      Activate the daemon configuration ...
```

Activation causes the menu system to add a `urmd` line to the `/etc/config/daemons` file. You can then stop `urmd` manually by using the `sdaemon -k urmd` command or restart `urmd` manually by using the `sdaemon urmd` command.

8.3.2 Configuring Initial URM Values

As-shipped default parameters for URM configuration are designed to minimize the impact of URM installation on a running system. This allows you to learn about URM and control its effects as you activate the configuration changes that allow URM to monitor various system limits.

The default configuration should have no effect on a running system until you explicitly change certain of these URM configuration values. The following section discusses these changes.

8.3.2.1 Individual Session Initiator Configuration Changes

The as-shipped default URM configuration overrides all URM limits and recommends initiation for all jobs started by all individual session initiators except batch. (In the case of batch jobs, this default never applies, since the NQS configuration parameters override these values.) To change these defaults and permit URM to make recommendations about whether or not to initiate jobs that started from individual session initiators, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Individual session initiator defaults ->
```

Initially, every entry in the default table of values is 0:

```
     Name   CPU Time  Memory Usage
     -----  --------  ------------
E->  batch  0         0
     cron   0         0
     ftp    0         0
     login  0         0
     null   0         0
     rexec  0         0
     rsh    0         0
     site1  0         0
     site2  0         0
     site3  0         0
```

For a given session initiator (`login`, for example), if both the `CPU Time` and the `Memory Usage` entries are 0, then you override URM limits and allow all jobs started by that session initiator. For example, if the `CPU Time` and `Memory Usage` entries for `login` are 0, URM favorably recommends all `login` requests, regardless of system load. Only when either entry for `login` is a nonzero value are `login` requests subject to URM limits.

Therefore, to enable communication between each session initiator and URM, change each 0 for that session initiator to a nonzero value. For example, for `login` and `rsh`, you might use a minimum of the estimated size of a shell process (in clicks).

Note: Configuration parameters in the NQS override these values for `batch`.

The `rsh` settings also affect `rcp` (remote copy).

For any changes to these configuration parameters to take effect, you must activate the changes by using the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Activate urm configuration ...
```

For details of the activation process, see Section 8.4.12, page 375.

8.3.3 Verifying Security Parameters

As part of any URM installation, you should verify that the URM configuration parameters establish only the authorized administrators and authorized hosts that you want. To learn about these parameters and how to change them, see Section 8.4.1, page 364.

8.3.4 Enabling Service Providers

With one exception, all service providers (such as `ftp`, `login`, and `rsh`) automatically use URM. The one exception is NQS. You must explicitly enable URM services in NQS.

To change the NQS configuration parameters to enable URM services, use the menu system. To enable communication between NQS and URM, the NQS configuration must include a statement to set URM on.

To ensure that this communication continues after a restart, the NQS configuration also must include a statement to set URM restart on. For details of how to enable URM services and how to enable URM restart in NQS, see the UNICOS NQS and NQE Administrator’s Guide, publication SG-2305.

8.4 Configuring URM

After URM has been installed and all initial values have been configured, you can change the URM configuration values as needed. The following sections describe the URM configuration parameters, how to use the menu system to implement changes in URM configuration, and how to activate changes.

Warning: The following information on configuring URM is not written for a site running a Cray ML-Safe configuration of the UNICOS system. For information on configuring URM for a Cray ML-safe configuration, see the description of the UNICOS MLS feature in General UNICOS System Administration, publication SG-2301.

To make changes to the URM configuration, use the menu system. Traverse the menus using the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
```

This menu selection displays the following menu:

```
M->  Authorized administrators ->
     Authorized hosts ->
     Machine load evaluation rates ->
     Machine target values ->
     Individual session initiator targets ->
     Individual session initiator defaults ->
     URM control settings ->
     Weighting factors for the selector ->
     Auto-configuration settings ->

     Reset DEFAULT urm configuration ...

     Import urm configuration ...
     Activate urm configuration ...
```

The following sections discuss each line of this URM configuration menu. In addition, online help is accessible on each menu in the menu system.

8.4.1 Authorized Administrators

To control access to URM, use the menu system. Traverse the menus using the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Authorized administrators ->
```

The as-shipped defaults for these parameters are as follows:

```
     Login name  Type
     ----------  ----------
E->  root        privileged
     *           public
     *           anonymous
```

These parameters have the following meanings:

Parameter	Description
`privileged`	Can read any internal information. Can stop the URM daemon (using `rmgr -c 'stopdaemon'`).
`public`	Can read only global information and information owned by this user.
`anonymous`	Can read only global information.

If you want to add an authorized administrator, create a line for the login of the new authorized administrator (`admin2`, for example):

```
     Login name  Type
     ----------  ----------
     root        privileged
     *           public
     *           anonymous
E->  admin2      privileged
```

If, for security reasons, you want to limit who can use `rmgr` to access URM, delete the two asterisked lines:

```
     Login name  Type
     ----------  ----------
     root        privileged
E->  admin2      privileged
```

This limits access privileges to `root` and `admin2`.

> **Note:** For proper URM functioning, you must retain the `root privileged` entry.
>
> The login name `*` is not recognized in the privileged node. Therefore, the entry `* privileged` is not allowed. This restriction also applies to the local URM configuration file; an entry of the form `/admin/privileged/*` is purposely illegal.

8.4.2 Authorized Hosts

To control access to URM from remote hosts, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Authorized hosts ->
```

The as-shipped default for this parameter is as follows:

```
     Host name
     ---------
E->  *
```

This parameter has the following meaning:

`*`	(all) Allows any client from a remote host to connect to `urmd`, assuming the administrator configuration also allows the connection.

If, for security reasons, you want to disallow connections from all remote hosts, remove the asterisk (`*`). You can then add a line for each host from which you want to allow connections (`system1` and `system2`, for example):

```
     Host name
     ---------
     system1
E->  system2
```

8.4.3 Machine Load Evaluation Rates

The as-shipped URM configuration includes a set of smoothing factors that are designed to reduce the impact of sudden changes in resource usage. These factors reduce the rate at which changes are factored into the URM decision-making process. This, in turn, protects URM users by making URM results more predictable.

To see or change the smoothing factors, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Machine load evaluation rates ->
```

The as-shipped default rates are as follows:

```
S->  MEMORY rate                          0.8
     SDS rate                             0.8
     TAPE rate                            1.0
     MPP BARRIERS rate                    1.0
     MPP PROCESSING ELEMENTS (pe)         1.0
```

These rates should not be changed, except by an analyst experienced in tuning systems using URM.

8.4.4 Machine Target Values

To properly perform its scheduling functions, URM must be aware of target usage limits for important system resources. These resources include memory, SDS, job count, tapes, and MPP (massively parallel processing systems).

During installation, a URM automatic configuration command changes many of the URM default targets. Automatic configuration determines the actual configuration of your system and replaces the as-shipped default values with values derived from the actual values for your system.

The formula used to change these values can be modified by using the menu system. Values that can be changed globally include memory and SDS oversubscription, and limits for job count, tapes, and MPP. In addition to these values, values for CPU time and memory usage can be set for each individual session initiator (see "Installing URM," Section 8.3, page 359).

The following sections describe the URM configuration values that can be set globally.

To verify or change the value (either multiplier or limit) for each important system resource, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Machine target values ->
```

The as-shipped URM configuration has the following values:

```
S->  Memory oversubscription multiplier  2.0
     SDS oversubscription multiplier     1.5
     Target session maximum              MAX
     Target tape limit                   MAX
     Target MPP barriers limit           MAX
     Target MPP PE limit                 MAX
```

The auto-configure script uses each of these values as follows:

Value	Description
`2.0`	Multiplies the value of your system's actual physical user memory (`user_memory`) by this value (`2.0`) to calculate a memory oversubscription value (`memory`). URM strives to hold memory usage below this memory oversubscription value. To see the value being used by URM, view the object `/machine/target/memory`. You cannot alter this object directly; you can only change the multiplier.
`1.5`	Multiplies the value of your system's actual SDS user space (`user_sds`) by this value (`1.5`) to calculate an SDS oversubscription value (`sds`). URM strives to hold SDS usage below this SDS oversubscription value. To see the value being used by URM, view the object `/machine/target/sds`. You cannot alter this object directly; you can only change the multiplier.
`MAX`	The following paragraphs describe, for each parameter, the effects of a `MAX` value. For `Target session maximum`, the `MAX` value sets the maximum number of jobs (both interactive and batch combined) that URM allows to be running at any time (`jobcount`). The upper limit for this range is defined by limits in the kernel configuration. That is, if this field contains a number that is higher than the maximum established in the kernel configuration parameters, this higher number is ignored, and the kernel configuration value is used. However, if this field contains a lower number, the lower number takes effect for URM. For `Target tape limit`, the `MAX` value compiles a list of all tape systems actually available on the system and puts this information in the field `tape`. For MPP limits, the `MAX` value allows URM to determine whether or not the system configuration includes an MPP system and, if so, allows URM to factor MPP limits into its scheduling algorithm.

Note: To allow proper URM autoconfiguration to occur, do not change the keyword MAX for this resource.

Using the menu system, you can reset these factors to any value you decide is reasonable.

For changes to these configuration parameters to take effect, you must activate the changes by using the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Activate urm configuration ...
```

For details of the activation process, see Section 8.4.12, page 375.

8.4.5 Individual Session Initiator Targets

To verify or change the target values for each individual session initiator, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Individual session initiator targets ->
```

This menu selection displays a table of target values, as in the following example:

```
     Name   bb   cputime  jobcount  memory  pe   petime   sds  tape
     -----  ---  -------  --------  ------  ---  -------  ---  ----
E->  batch  MAX  9999999  MAX       MAX     MAX  9999999  MAX  MAX
```

8.4.6 Individual Session Initiator Defaults

To change the default values for each individual session initiator, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Individual session initiator defaults ->
```

This menu selection displays the table of default values. For example:

```
     Name   CPU Time  Memory Usage
     -----  --------  ------------
E->  batch  0         0
     cron   0         0
     ftp    0         0
     login  40        250
     null   0         0
     rexec  0         0
     rsh    0         0
     site1  0         0
     site2  0         0
     site3  0         0
```

Entries for `CPU Time` are given in seconds. Entries for `Memory Usage` are given in clicks. You can raise or lower these limits by changing the entries in this table.

Lines for `site1`, `site2`, and `site3` are for local session initiators defined by the site.

For a given session initiator (except `batch`), if both entries are 0, that session initiator is not subject to URM limits.

Note: Configuration parameters in the NQS override these values for `batch`.

The `rsh` settings also affect `rcp` (remote copy).

For changes to these configuration parameters to take effect, you must activate the changes by using the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Activate urm configuration ...
```

For details of the activation process, see "Activating URM configuration changes," Section 8.4.12, page 375.

8.4.7 URM Control Settings

The URM configuration includes a variety of control factors that affect URM performance. To see a list of these control factors, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      URM control settings ->
```

This menu selection displays a table of URM control factors and current settings. For example:

```
     Seconds between load info access            10
     Seconds to wait for job initiation          1800
     Seconds between scheduling cycles           10
     Main loop sleep period                      10
     Old Caller timeout in seconds               600
     Share evaluation period in seconds          900
     Name of the SDS suspend command             UPATH/sdsspnd
S->  SDS residency time in secs (0=SDS mgmt off)  900
     Full path name of local URM config file     /etc/config/urm
     Full path name of URM chkpnt directory      /PPATH/chkpnt
     Checkpoint policy                           Shutdown
     Share policy                                Standard
     Minimum # of seconds between checkpoints    1800
     Checkpoint interval measurement base        Clock
     Maximum # of seconds to keep chkpnt file    432000
     Name of the interactive chkpnt command      /UPATH/intchkpt
     Restart flag policy                         Force
     Name of the interactive restart command     /UPATH/irstart
     Rank boost to previously running batch jobs  6.0
```

The following sections discuss three of these factors that may be particularly useful to administrators: `SDS residency time in secs`, `Full path name of local URM config file`, and `Rank boost to previously running batch jobs`.

8.4.7.1 SDS Residency Time

This menu selection determines the minimum amount of time that a job remains in SDS before being swapped out (900 seconds, in this example). This applies to both batch and interactive. Prior to UNICOS 8.0, `qfdaemon` performed this function, but only for batch jobs.

If SDS residency time is set to 0, then URM does not perform SDS management. If SDS residency time is set to a nonzero number, then URM performs SDS management on interactive jobs. For URM to perform SDS management on both interactive and batch jobs, SDS residency time must be set to a nonzero number and the NQS `set job scheduling` *option* parameter must be set to the correct option. For details of enabling URM services in NQS, see the UNICOS NQS and NQE Administrator's Guide, publication SG-2305.

8.4.7.2 Local URM Configuration File

This menu selection is used to identify a site-defined configuration file. When activated, changing this value to *filename* adds an `Include` *filename* statement to the end of the `/etc/config/urm/configuration` file.

You can change the URM configuration by editing this site-defined configuration file. Or you could change the URM configuration by writing a `cron` job that accesses this file.

8.4.7.3 Rank Boost to Previously Running Batch Jobs

This menu selection specifies an additional value given to checkpointed batch jobs to improve their rank. This value is applied to batch jobs that were previously running and were checkpointed due to a shutdown or hold. The rank boost is added after the job's rank has been calculated using the weighting factors (described in the following section).

8.4.8 Weighting Factors for the Batch Selector

The method by which URM selects the next batch job to recommend for initiation depends upon the interaction between the NQS and URM. This interaction is influenced by a set of weighting factors established in the URM configuration. To see or change these weighting factors, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Weighting factors for the selector ->
```

The as-shipped default settings for these factors are as follows:

```
S->  Age in queue weight                    0.5
     MPP Barrier bits weight                0.5
     MPP Processor elements weight          0.5
     MPP Requested PE time limit weight     0.5
     Requested CPU time weight              0.5
     Requested Memory weight                0.5
     Requested SDS weight                   0.5
     Requested Tape weight                  0.5
     Service provider priority weight       0.5
     Share priority weight                  0.5
     Share entitlement weight               0.5
     Usage weight                           0.5
```

Possible values are in the range of 0.0 to 1.0. For each batch job, URM assigns a ranking based upon place in each queue. For example, using the `Age in queue weight` factor, the oldest job is assigned a ranking of 0.5, while the youngest job is assigned a ranking of 0.0. By comparing the resulting weighting factor for each batch job, URM determines the highest priority job.

These default settings should be changed only by a system analyst experienced at tuning systems running URM.

Note: Share priority is the result of the following calculation:

$Usage/entitlement^2$

Although the share priority, usage, and share entitlement weighting factors all have a nonzero value by default, you must disable one or two of these values. Configure only one of the following choices:

- Share priority weight
- Share entitlement weight
- Usage weight
- Share entitlement weight and usage weight

8.4.9 Auto-configuration Settings

To enable auto-configuration of various subsystems, URM includes a tool in the menu system. To see or change the auto-configuration settings, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Auto-configuration settings ->
```

The as-shipped default setting for each subsystem is as follows:

```
     Resource  Enabled  Tries  Wait time
     --------  -------  -----  ---------
E->  memory    YES      10     20
     mpp       YES      10     20
     serial    YES      10     20
     session   YES      10     20
     sds       YES      10     20
     tape      YES      10     20
```

The URM has the ability to determine the available system resources. This menu allows you to modify the retry parameters of the auto-configuration mechanism.

The `Resource` field names the resource to be auto-configured. URM can automatically determine the system's `memory`, `mpp`, `serial`, `session`, `sds`, and `tape` resources. The `serial` resource configures the machine's serial number and the information available through `uname`(3). The `session` resource corresponds to the system's defined maximum number of sessions (NSESS). The `Enabled` (`Perform auto-configuration`) field is used to enable or disable the auto-configuration of the specified resource. Cray recommends that these fields remain set to YES, to allow auto-configuration.

Note that disabling auto-configuration prevents URM from using the MAX settings in the `URM Machine target values` and `URM Individual session initiator targets` menus.

The `Tries` (`Number of tries`) field indicates the number of times URM should attempt to auto-configure the specified resource. It is necessary to have a retry value, because some resources may not be available when URM is first initialized. For example, if URM is brought up before the tape daemon is running, URM's first attempt at auto-configuring tape resources fails.

The `Wait time` field represents the time in seconds to wait before retrying the auto-configuration of the specified resource.

8.4.10 Resetting to Default URM Configuration

The menu system allows you to reset all URM configuration parameters to the as-shipped default state. This could be useful if you find that configuration changes you have made have created unexpected problems and you want to start over with a known-working URM configuration. To remove all local changes, and return to the as-shipped default configuration parameters, use the menu system. Traverse the menus using the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Reset DEFAULT urm configuration ...
```

8.4.11 Importing URM Configuration

For systems on which a URM configuration has not yet been imported, the menu system provides a tool that reads in certain defaults. To read in all values contained in the `/etc/config/urm/configuration` file, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Import urm configuration ...
```

Warning: This menu item should be selected only on systems that do **not** have a running URM, as it overwrites any previously existing values for these URM parameters.

The values read in are for the following parameters: `init`, `machine`, `res`, `structure`, `urminfo`, and `val`.

8.4.12 Activating URM Configuration Changes

After implementing any URM configuration changes, you must activate the changes for them to take effect. To activate the changes from within the menu system, use the following menu selections:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Activate urm configuration ...
```

The next time `urmd` is started, the menu system updates the contents of the `/etc/config/urm/configuration` file.

Caution: The `/etc/config/urm/configuration` file should never be edited directly; any changes should be made through the menu system.

Changes activated from within the menu system do not affect the currently executing `urmd` process; you can make the changes known to the currently executing `urmd` process by typing the following:

```
rmgr /etc/config/urm/configuration
```

8.4.13 Using URM with NQS

By default, URM does not control NQS jobs. To enable full URM control over the initiation of NQS jobs, enter the following `qmgr` subcommand:

```
set job scheduling urm unlimited
```

With full URM control, NQS queue limits are no longer honored; instead, URM's resource targets and job ranking priorities determine which jobs are recommended to NQS for initiation. For information about NQS parameters, see the `qmgr`(8) man page. For information on changing NQS configuration parameters, see the UNICOS NQS and NQE Administrator's Guide, publication SG-2305.

8.5 URM Administrator Tasks

To administer URM, you should understand the following procedures:

- Using URM log files
- Viewing URM results
- Viewing machine load
- Viewing all users
- Viewing jobs of a given user
- Changing a job's minimum rank
- Changing URM configuration based on time of day (`cron`)
- Customizing URM (user exits)

These procedures are described in the following sections.

8.5.1 Using URM Log Files

The `urmd` process maintains a log file to record its actions. Each time the daemon initializes, it creates a log file if one does not exist in the URM log directory. The log file is placed in the `/usr/adm/urm/` directory and is named `Urm`. *yymmdd*. The log file is appended to throughout the day (*yymmdd*), recording all directives, errors, and other important actions when the daemon is running. The next day, when a log message occurs, the existing log file is closed, the date is incremented, and a new log file is created.

An example of log entries is as follows:

```
11:16:20    153 U D Set Basenode /
11:16:20    153 U D Set Kernelaccess
11:16:20    153 U D Set Initialized
```

The format of the log messages is as follows:

hh:mm:ss 000000 x y message

The message is prefixed by the time the message was logged (*hh:mm:ss*), the UID of the current user when the message occurred (*000000*), the access privilege (*x*), and the severity code (*y*). UID `999999` means that the message comes from `urmd` and is not associated with a user.

The access privilege labels (*x*) are as follows:

`L`	Public socket access
`N`	No user defined (internal message)
`U`	Unrestricted privilege socket access

The severity labels (*y*) are as follows:

`D`	Logging a directive
`F`	Fatal
`I`	Informative
`L`	Log message
`N`	No error

`R`	Error type out of range
`W`	Warning
`X`	Exit message

8.5.1.1 Monitoring URM Log Files

Because `urmd` creates a new log file each day (`/usr/adm/urm/Urm`. *yymmdd*), the number of log files grows rapidly. To conserve disk space, you should monitor this growth and implement a strategy to minimize the effects. For example, you could archive all but the most recent log files. You could use `cron` to accomplish this on a scheduled basis, such as running an archiving script at the end of each week.

8.5.1.2 Turning Off URM Logging

The log file can be turned off, using the local configuration file. To turn off logging the activities of URM commands, enter the following:

```
# rmgr
rmgr-> set nolog
```

8.5.1.3 Moving the URM Log Files

By default, the URM log files are in the `/usr/adm/urm` directory. You can change the default directory with either of the following methods.

When invoking the the URM daemon, `/etc/urmd`, use the `-l` option to specify an alternate log directory. The log file names cannot be changed, only the path. See the `urmd`(8) man page for more information about the `-l` option.

When URM is running, an authorized URM administrator can change the location of the log using the following `rmgr`(1) subcommand:

`set log` *directory*

If *directory* specifies a valid directory, the current log is terminated with a message and closed, and a new log is opened in *directory*. Error messages are issued if the new log file cannot be opened, and logging remains with the old log file.

The `set log` *directory* directive can also be specified in the local configuration file. However, when this method is used, the URM daemon writes the initialization log in the default directory (`/usr/adm/urm`). To change the location of the initialization log, use the `-l` option when starting `/etc/urmd`. See the `rmgr`(1) man page for more information about the `set log` directive.

8.5.2 Viewing URM Results

At any time while URM is running, you can generate a list of the jobs that URM currently recommends for initiation. To generate this list, first invoke the `rmgr`(1) command:

```
# rmgr
```

Then choose the `view jselect` (view job select) option:

```
rmgr-> view jselect
```

This yields a report of jobs currently recommended for initiation. You can use this report to help determine configuration changes needed to tune URM for your site.

8.5.3 Viewing Machine Load

To assess machine load, first invoke the `rmgr`(1) command:

```
# rmgr
```

Then choose the `view /machine/load` (view machine load) option:

```
rmgr-> view /machine/load
```

This yields a report of the resource usage of all jobs currently running, showing the load on all subsystems monitored by URM. This report can be useful in troubleshooting URM, to determine whether some parameter has been set too low and has been exceeded.

8.5.4 Viewing All Users

To generate a list of all URM users, first invoke the `rmgr`(1) command:

```
# rmgr
```

Then choose the `view users` (view all users) option:

```
rmgr-> view users
```

This yields a report of all users currently connected to the URM daemon and their respective privileges (privileged, public, or anonymous).

8.5.5 Viewing Jobs of a Given User

To monitor the activities of a single user, first invoke the `rmgr`(1) command:

```
# rmgr
```

Then choose the `view jobs` (view one user) option:

```
rmgr-> view jobs login | UID
```

This yields a report of all jobs currently recommended for initiation that are owned by the specified user (*login* or *UID*). This might be useful in predicting system load or in troubleshooting URM.

8.5.6 Changing a Job's Minimum Rank

By default, NQS sets the minimum rank of jobs in the job backlog. The `ustat`(1) command allows the URM administrator to change the minimum rank of a specific job or jobs to affect the URM rank of those jobs.

Use `ustat`(1) to display the minimum rank of batch jobs, as in the following example:

```
ustat -a -m
```

The `view` directive of `rmgr`(1) also displays minimum rank values.

Use the `usetjob`(8) command to change the minimum rank of batch jobs. The following example changes the minimum rank for NQS job ID 12345:

```
usetjob -r 7 12345
```

If multiple machines submit jobs, the machine name is necessary to prevent possible misinterpretation of the request ID. If the job in this example originated from the machine `fred`, you would use the NQS request ID `12345.fred`. The host ID (for example, 999) could be used instead of the machine name, giving a job identification field of `12345.999`.

8.5.7 Changing URM Configuration Based on Time of Day (`cron`)

You can use the `cron` command to change the URM configuration at certain times of day. This would be useful, for example, if you wanted one configuration for daytime loads that include more interactive sessions and a different configuration for night-time loads consisting primarily of batch jobs. To implement such a configuration change:

1. Use the menu system to create a configuration file for daytime use (for example, `config.day`) and activate the change.

   ```
   cp /etc/config/urm/configuration /etc/config/urm/config.day
   ```

2. Use the menu system to create a configuration file for night-time use (for example, `config.night`) and activate the change.

   ```
   cp /etc/config/urm/configuration /etc/config/urm/config.night
   ```

3. Specify a local configuration file to be included when `urmd` is started (for example, `/etc/config/urm/local`).

4. Edit the `config.day` and `config.night` files to remove from each file the following line:

   ```
   Include "/etc/config/urm/local"
   ```

   This allows the auto-configuration to work properly in the event of a restart.

5. Create a `crontab` file that includes the following:

   ```
   cp /etc/config/urm/config.day /etc/config/urm/local
   rmgr /etc/config/urm/local
   ```

   The correct configuration file for the time of day must be copied to the `local` file, so that, in the event of a restart, the correct configuration file is used automatically.

8.5.8 Customizing URM (User Exits)

The URM contains code that allows you to customize it for the specific needs of your site by creating a site-written job selector. In URM, this site-written selector code is called last.

As shipped, the selector code is empty; it defaults to exiting without doing anything. However, you can write your selector based on any ranking algorithm you choose, and insert your code into URM. Then you could turn off all URM selectors and allow your site-written selector to make all job-initiation recommendations based on your ranking algorithm.

Warning: Use of any user exit is not permitted on a Cray ML-Safe configuration of the UNICOS system. Use of any user exit may result in loss of the evaluated rating.

8.5.8.1 URM User Exits

URM includes two user exits. The URM daemon (`/etc/urmd`) contains two routines that can be replaced with a set of user-defined versions by re-linking `urmd` with them. These user exits are defined in `site_rank.c`.

User exit	Description
`site_adjust_merit`	Allows the user to modify the ranking factor. A higher number increases the chances that a job will be recommended for execution. A lower number makes it more difficult for the job to be recommended.
`site_target_check`	Allows the user to set and reset limit flags for a job entry in URM. Jobs that have flags set will not be recommended for execution.

The order in which these ranking functions are invoked affects how the user exits can be used.

To understand URM user exits, you must understand certain functions and data structures in URM that are involved with the URM user exits. These are described in the following two sections.

8.5.8.2 URM Job-ranking Functions

URM includes three job-ranking functions (defined in `rank.c`) that call the user exits. The job-ranking functions are as follows:

Function	Description
`rank_jobs`()	Forms the jobs list. To qualify, the job must pass `raw_select` () and `maximum` filters. Next, the relative priorities and resource requirements are used for the final ranking in the job list. The other two functions are called from this one.
`adjust_merit`()	Adjusts the merit parameters of a job. The user's share is evaluated and used to adjust the relative merit of a job. The relative ranking value is stored in `urm_rank`. Called with 0 to preset the statics. This function calls the user exit `site_adjust_merit`().

`target_check()`	Determines whether or not the job exceeds the target usage limits. If the job will not exceed the target, returns `UR_NONE`. If the job will exceed the target, returns one of the other `UR_`*xxx* reason codes. This function calls the user exit `site_target_check()`.

The pseudo-code for `rank_jobs()` is as follows.

```
rank_jobs {                             /* The real ranking function.   */

   initialize_the_first_list            /* Initialize firstlist of jobs.*/
   initialize_the_job_list              /* Initialize job list.         */
   initialize_the_rec_list              /* Initialize recommend list.   */
   initialize_the_pre_list              /* Initialize prempt list.      */
   initialize_the_rest_list             /* Initialize restore list.     */
   initialize_the_chkpnt_list           /* Initialize checkpoint list.  */

   get_loadinfo(objects)                /* Get system load information. */

   adjust_merit(initialize variables)   /* Initialize ranking weights.  */
      site_adjust_merit(initialize)     /* Allow user to initial other  */
                                        /*   user specific variables.   */

   target_check(0, TC_INIT)             /* Initialize usage limits.     */
      site_target_check(initialize)     /* Allow user to initial other  */
                                        /*   user specific limits.      */

   for (job_objects) {                  /* Add job objects to lists.    */
      update_prelist(job_object)        /* Add to prempt list.          */
      update_restlist(job_object)       /* Add to restore list.         */
      update_chkpntlist(job_object)     /* Add to checkpoint list.      */
      if (already_ranked_job)
         target_check(job, TC_PRELOAD)  /* Update potential usage and   */
            site_target_check(initialize) /*  initialize again.         */
      if (raw_select(job_object))       /* if job passes raw limits then*/
         update_firstlist(job_object)   /*    add to the first list.    */
   }

   for (first_list) {
      apply_jobmax_constraints          /* Begin to weed out jobs.      */
         create_job_list                /* Create actual job list.      */
   }
   for (job_list) {                    /* Loop through the job list.   */
      adjust_merit(job)                /* Calculate this jobs ranking  */
                                       /*   factor.                    */
         site_adjust_merit(job)        /* And user can make changes to */
                                       /*   the ranking factor.        */
   }

   sort_job_list                       /* Sort job list.               */

   for (job_list) {                    /* Loop through the job list.   */
      target_check(job)                /* Will the job exceed limits?  */
         site_target_check(job)        /* Allow user to do more checks */
                                       /*   and even reset limit flags.*/
      if (no_reason_code)
         add_job_to_rec_list           /* Add to recommend list.       */
   }
}
        /* The rec_list now contains the recommended list of jobs that  */
        /* the session initiator should try and start.                  */
```

8.5.8.3 URM Data Structures

The structures and typedefs that URM uses to contain the job list information are defined in the following header files:

```
#include <urm.h>
#include <errnum.h>
#include "urmdefs.h"
#include "object.h"
#include "job.h"
#include "rank.h"
```

Two URM data structures in particular are useful to the user exits. These structures are defined in the following two header files, which are found in the directory `/usr/src/prod/admin/urm/urmdaemon`:

`Typedef Object;`	Defined in `object.h`
`Typedef Job;`	Defined in `job.h`

The `Job` data structure contains a field that allows URM to manage site-specific job information. When the `JOBHIST_UXIT` flag in the `history` field is set, URM interprets the `uxit` union in the `Job` data structure as a pointer to allocated memory and frees the specified space. However, if the user exit does not use the `uxit` field or uses it as a number, the `JOBHIST_UXIT` flag must not be set.

It is vital to URM that a user exit correctly modifies data in these structures. Whoever is working with user exits should understand and be familiar with the structures present in the header files, in order to use them effectively.

There are also three predefined site flags and reason codes. When URM does a target check, it returns a flag and a reason code for what caused the job to not be recommended. The flags are defined as macros in `rank.h` and the reason codes are defined by the typedef `URM_code` in `job.h`. The predefined site flags and reason codes are as follows:

Flag	Reason code
`_SITE1_TAR`	`UR_SITE1_TAR`
`_SITE2_TAR`	`UR_SITE2_TAR`
`_SITE3_TAR`	`UR_SITE3_TAR`

Caution: The user exits can give a site almost total control over the ranking algorithm used by URM. In theory, a site can totally modify the recommended job list (and other internal URM structures).

When writing user exits, be careful when using any of the following.

These can have an adverse affect on URM and could result in an unreliable or unusable URM daemon. The user exit should, if possible, avoid any of these conditions or uses.

- System calls or functions that would block (such as trying to connect to a socket).
- `fork` and `exec` calls.
- Creating pipes or opening files.
- Use of signals.
- Code that significantly slows down the ranking.

8.5.8.4 URM User Exit Example 1

In this example, a user is submitting a weather model batch job that should be given a high priority to run, and should not have to wait in an NQS queue for very long. The URM user exits can be used to ensure that user’s job is set to run before most other jobs, if not first.

One solution is to use the `site_adjust_merit`() function. When `adjust_merit`() calls `site_adjust_merit`() with `obj` set to 0, this indicates that the user exit should initialize any variables. When an actual `obj` is passed to `site_adjust_merit`(), it contains an associated job and the ranking factor. At this point this user exit returns an increased ranking factor. The amount of the increase determines where in the job list URM puts this job. The closer to the top, the more likely this job is to be recommended by URM before others (if no targets are exceeded in `target_check`()).

The following example code (which should be placed in `site_rank.c`) would do this:

```
/*
 *      Variable hold the static value for site_adjust_merit
 */
static float site_rank_adjust;

/*
 *      float site_adjust_merit() - Adjust the ranking of a job
 *                                  depending on specific needs of the site.
 *
 *      Return the rank this job should have.
 */
float
site_adjust_merit(float   rank,
                  const   Object *obj)
{
    if (!obj) {                         /* initialize values */
        site_rank_adjust = 10.0;        /* set adjustment value */
                                        /* This value may need to be */
                                        /* increased at your site. */

    } else {                            /* else make adjustment */
        if (strncmp(obj->ovalue.job->owner,"user1",5) == 0) {
            rank += site_rank_adjust;
        }                               /* If ’user1’ is the owner */
    }                                   /* give a high rank value. */
    return (rank);
}
```

In this example code, `obj->ovalue.job->owner` contains the name of the owner of that job. This example compares name to `user1`, which is assumed to be running the weather model batch job. If true, the rank is adjusted and the new value is returned. By giving a high rank to this job, we are guaranteed that this job will be ordered first (or almost first) on the job list.

Note that if the weather model job does not pass the target checks, it still will not be recommended for execution.

8.5.8.5 URM User Exit Example 2

In this example, a user should be prevented from running batch jobs from 8:00 A.M. to 5:00 P.M. (during peak interactive loads). The URM user exits can be used to prevent this particular user from running batch jobs during this time.

One solution would be to change the ranking factor as in example 1, but instead of adding to the ranking factor, decrease it by a significant amount (make sure the ranking is a positive number). Although this user could still possibly run jobs, those jobs will be very difficult for URM to recommend for initiation.

Another way to prevent the user's batch jobs from running during this time is to use the `site_target_check`() user exit. The following example code (which should be placed in `site_rank.c`) would do this:

```
/*
 *      int peak_inter_time() - determine if between 8:00am and 5:00pm
 */
static int
peak_inter_time()
{
     int rc=0;                          /* 0 = off hours, 1 = peak time */
     time_t timval;                     /* see time(2) man pages */
     struct tm *tmptr;                  /* see time(2) man pages */

     timval = TOTIME(_rtc());           /* TOTIME is a URM macro that acts */
                                        /* like time() system call but uses */
                                        /* the real time clock instead and */
                                        /* does not cause a context switch. */

     tmptr = localtime(&timval);        /* break into components */

     if ((tmptr->tm_hour >= 8)&&(tmptr->tm_hour < 17)) rc = 1;
                                        /* If greater than 8:00am and less */
                                        /* than 5:00pm set the return code. */
     return (rc);
}

/*
 *      int site_target_check() - Extend the target checking function
 *                                according to needs of the site.
 *
 *      Return appropriate flags to the calling routine.
 *
 *      If init_flag is true, this is an initialization call which
 *      may or may not be useful depending on the nature of the
 *      code.
 */
int
site_target_check(int     flags,
                  Object *obj,
                  struct  _indirect_vals *target,
                  struct  _indirect_vals *load,
                  struct  _total_vals    *total,
                  struct  _need_vals     *need,
                  const   int init_flag)
{
    if (init_flag) {                    /* initialize values */
        /* do initialization */         /* set values */

    } else {                            /* else make adjustment */

        if (strncmp(obj->ovalue.job->owner,"user1",5) == 0) {
                                        /* If 'user1' is the owner, */

            if (peak_inter_time()) {  /* and this is peak interactive time, */
                flags |= _SITE1_TAR;  /* prevent the job from running by */
            }                         /* setting the flag.  target_check() */
                                      /* will set the reason code later. */
        }
    }
    return (flags);
}
```

The `site_target_check()` uses `peak_inter_time()` only to say if the system is currently running in peak interactive time. `site_target_check()` checks to see if this job is owned by `user1`, and, if so, it calls `peak_inter_time()` to find out if this is peak interactive time. If all true, set the flag value for `_SITE1_TAR`, meaning that a site-defined target has been exceeded by this job (in this case, the job will **not** be recommended). After returning the flag, `target_check`() knows that this flag means to set the reason code to `UR_SITE1_TAR` and the job will not be recommended to run. If `user1` were to queue up batch jobs at 4:45 P.M., they will sit in the NQS queue until 5:00 P.M., at which time the user exit no longer flags these jobs and URM begins to recommend that they be run.

8.5.8.6 URM User Exit Example 3

In this example, an NQS queue has been designated as a high priority queue called `weather`, into which weather model batch jobs are submitted. The URM user exits can be used to ensure that this queue gets high priority.

This example is similar to example 1, but instead of basing priority on the user ID, priority is established by service priority. An assumption is made that NQS queues have unique service priorities and that the `weather` queue service priority is known inside the user exit. Using `site_adjust_merit`(), the ranking factor for jobs having the service priority for the `weather` queue can be increased. The amount of the increase determines where in the job list URM puts this job. The closer to the top, the better chance this job has of being recommended by URM before others (if no targets are exceeded in `target_check`()).

The following example code (which should be placed in `site_rank.c`) would do this:

```
/*
 *      Variable site_weather_svcpri holds the static value for svcpri.
 *      Variable site_rank_adjust holds the static value for rank adjustment.
 */
static int site_weather_svcpri;
static float site_rank_adjust;

/*
 *      float site_adjust_merit() - Adjust the ranking of a job
 *                                  depending on specific needs of the site.
 *
 *      Return the rank this job should have.
 */
float
site_adjust_merit(float   rank,
                  const   Object *obj)
{
    if (!obj) {                         /* initialize values */
        site_weather_svcpri = 40;       /* set 'weather' queue service pri */
        site_rank_adjust = 10.0;        /* set adjustment value */

    } else {                            /* else make adjustment */
        if (obj->ovalue.job->svcpri == site_weather_svcpri) {
            rank += site_rank_adjust;
        }                               /* If svcpri of 'weather' queue, */
    }                                   /* then give a high rank value. */
    return (rank);
}
```

In the example code, `obj->ovalue.job->svcpri` contains the `svcpri` of the batch queue of that job. This example compares an `svcpri` of 40, which is assumed to be the service priority of the `weather` NQS queue, to what is in `obj->ovalue.job->svcpri` for the job. If they are the same, the rank is adjusted and the new value is returned. By giving a high rank to this job, we are guaranteed that this job will be ordered first (or almost first) on the job list.

If the weather model job does not pass the target checks, it still will not be recommended for execution.

8.5.8.7 URM User Exit Example 4

This example shows how to modify the `site_target_check`() function so that NQS jobs from queues with a specific service priority will always be recommended for initiation.

```
#define EXPRESS_PRI 15

/*
 * This module is reserved for local use.
 *
 * Please see rank.c for the calls to these function.
 */

/*
 * int site_target_check() - Extend the target checking function
 * according to needs of the site.
 *
 * Return appropriate flags to the calling routine.
 * Zero means no targets exceeded.
 * If init_flag is true, this is an initialization call which
 * may or may not be useful depending on the nature of the code.
 */

int
site_target_check(int flags,
                  Object *obj,
                  struct _indirect_vals *target,
                  struct _indirect_vals *load,
                  struct _total_vals *total,
                  struct _need_vals *need,
                  struct _max_vals *max,
                  const int init_flag)
{
    Job     *jp;
    char    msg[132];

    if (init_flag) {
        /*
         *      Do nothing for initialization.
         */
        return(flags);
    }
    jp = obj->ovalue.job;     /* Get the job object pointer */

    if (jp->svcpri == EXPRESS_PRI) {
        /*
         *      If the job’s svcpri is EXPRESS_PRI, format and write
         *      a log message.
         *      Returning zero makes sure this job is not removed
         *      from consideration by the target checking rules.
         */

        sprintf(msg, "Batch job %d in NQS express queue (pri %d) cleared",
                jp->svcid, EXPRESS_PRI);
        write_urm_log(ESTATE_INFO, msg);
        return(0);

    } else {
        /*
         *      Jobs that are not EXPRESS_PRI get no special treatment;
         *      'flags' is returned without change.
         */
        return (flags);
    }
}
```

8.6 Troubleshooting URM

The URM is part of the Cray Message System and provides explicit indications of the cause of many error conditions and statuses. Both the `urmd`(8) daemon process and the `rmgr`(1) user interface command access the message system.

8.6.1 URM Daemon Failures

If the `urmd` process fails to start or aborts during execution, check the log file for an indication of the reason. Reasons for the `urmd` process to fail include:

- The `urm` service name does not exist in the `/etc/services` file. Use the menu system to create the `urm` service port.
- The `/bin/rmgr` command does not exist.
- The `/etc/urmd` process was not initiated by a privileged process.
- Configuration files either have not been installed in `/etc/config/urm` or contain invalid commands.
- Files with the same names as the configuration files exist in `/usr/adm/urm`. To open a configuration file, `rmgr` first checks its current directory (during initialization, `/usr/adm/urm`) before the configuration directory (`/etc/config/urm`). The critical file names are `configuration`, `init`, `local`, `machine`, `res`, `structure`, `urminfo`, and `val`.

The `urmd` process can be restarted on an active system. It reads the session table to determine the current state of the machine and continue from there. It is important to reconnect the NQS daemon to the restarted `urmd` process. To do so, use `qmgr` directives to set URM scheduling on and to set URM restart on. For details of enabling URM services in NQS, see the UNICOS NQS and NQE Administrator's Guide, publication SG-2305.

If the `rmgr` command fails to connect to a `urmd` process on a remote machine, it is likely that the URM configuration on the remote machine does not have this machine in its `/hosts` list.

> **Note:** To execute the following procedure and get the expected results, you must be in the URM authorized administrator list (`/admin/privileged`).

To view which machines are allowed to connect remotely, use the `rmgr` command, followed by the `view /hosts` option:

```
# rmgr
rmgr-> view /hosts
-Vrw--  0    0 Jan  4 13:22 <*          >  "null"
```

In this case, the `*` shows that remote connections are accepted from all hosts (as long as user validation also passes). If the desired host is not in the hosts list, use the menu system to add the host(s) to the URM configuration.

If a non-superuser process fails to obtain a privileged connection to the `urmd` process even though their name is in the URM `/admin/privileged/` list, ensure that the `/bin/rmgr` executable has been installed properly. This program must be installed as `setuid-root` in order to use a privileged socket connection with the `urmd` process.

8.6.2 URM and NQS

The URM and NQS work closely together to maintain the desired system loads. For NQS to work with URM, the NQS daemon must be configured to communicate with the `urmd` process. To verify this communication, first make sure that the `urmd` process is executing (and if not, restart it). Then check the NQS configuration parameters to verify that URM scheduling is turned on (the NQS `set job scheduling` *option* parameter must be set to the correct *option*). For information on displaying the NQS configuration parameters, see the `qmgr`(8) man page. For information on changing NQS configuration parameters, see the UNICOS NQS and NQE Administrator's Guide, publication SG-2305.

To manually establish (or re-establish) communication between NQS and URM on a running system, you can use the `qmgr`(8) command, specifying the appropriate subcommand.

To return NQS to its default mode (disable URM control), enter the following `qmgr` subcommand:

```
set job scheduling nqs normal
```

For more information on the `qmgr` command and subcommands, see the `qmgr`(8) man page.

8.6.3 URM and the Fair-share Scheduler

The URM considers fair-share usage information from the user data base (UDB) when prioritizing batch jobs. If your system has the fair-share scheduler turned on, it must be functioning properly, that is updating the usage information in the UDB.

This section explains the three fair-share components used by URM: share priority weight, share entitlement weight, and usage weight. For more information on fair-share, see "Fair-share Scheduler", Chapter 4, page 193.

8.6.3.1 Share Priority Weight

In a system using the fair-share scheduler, URM evaluates the effective share of each resource group and stores the information in a table named `/share` in the URM object tree. Each of the table entries holds the relative share of its resource group. The groups are converted from the hierarchy into a flat organization for ease of searching and evaluation. Relative shares are not adjusted for usage in the object tree.

URM evaluates the share organization at a rate determined by `/urm/share_eval` (the time to go before the next evaluation is found in `/urm/share_to_go`). Normally, the resource group hierarchy and the relative priorities among the resource groups do not change frequently. If the administrator changes the organization of the resource hierarchy or the relative shares within the hierarchy, URM can be forced to regenerate its internal view of the resource groups by setting `share_to_go` to 0.

When a job is being evaluated, the user's resource group is determined as input to the share evaluation function. The entitlement of the user is determined as described in the following section. Next, usage is determined as described in the "Usage weight" section, Section 8.6.3.3, page 395 (the method depends on the setting of `share_policy`), and the priority is determined from the following expression:

$priority = usage / entitlement^2$

This is the same expression that is used in the kernel to compute `kl_usage`, which is the main component of `p_sharepri`. Numerically smaller values are better. If the fair-share scheduler is not active, the value of this component is 0.

Each job's share priority for ranking is determined from the following expression:

share_wt`*(1.0-(`*job_priority*`/`*max_priority*`))`

Share priority ranking is a combination of usage and entitlement ranking available separately. It is expected that either share priority ranking or a combination of usage and entitlement ranking would be chosen, but not some combination of share ranking and usage or entitlement ranking. This capability offers additional flexibility in ranking choices.

8.6.3.2 Share Entitlement Weight

The share entitlement at each level in the fair-share hierarchy is determined from the following expression:

entitlement=*node_share*/*group_share*

The value *group_share* is the total number of shares allocated in the resource group. This means that a node's entitlement is the fraction of the total shares of the group assigned to that node, without regard to how many nodes in the group have activity. This is not how the fair-share scheduler evaluates a user's entitlement, but an exact method is very difficult to implement when some of the users or resource groups waiting to have jobs initiated may not be active. The intent is to determine the relative entitlements of a number of users competing for initiation.

Each level of the hierarchy is evaluated as described in the preceding paragraph, and the product of the entitlements of each level in the hierarchy becomes the user's entitlement. This value is returned to the share priority calculation and to the ranking evaluator separately to be normalized and weighted with the `entitle_wt` factor.

Numerically larger values are better. If the fair-share scheduler is not active, the value of this component is 0. Each job's entitlement for ranking is determined from the following expression:

entitle_wt`*(`*job_entitlement*`/`*max_entitlement*`)`

8.6.3.3 Usage Weight

Usage is derived from the decayed usage as maintained by the fair-share scheduler in either the lnode or the UDB. When URM examines a job, it checks for current active usage by using the `limits`(2) system call. If the user is active (has an lnode), current usage is returned. If the user is not active, the user's record is read from the UDB, and the current usage value is calculated from the recorded usage decayed over the time period since the user last used the machine.

If `share_policy` is `Standard`, only the terminal node's usage is returned. If `share_policy` is `Fair_ratio`, usage of the subject node, proportional to the sum of the usage in each level of the hierarchy, is multiplied together to form the usage value. Usage supplied to the share priority component is the value determined by `share_policy`; this value is also returned to the ranking evaluator separately to be normalized and weighted with the `usage_wt` factor.

Numerically smaller values are better. If the fair-share scheduler is not active, the value of this component is 0. Each job's usage for ranking is determined from the following expression:

usage_wt`*(1.0-(`*job_usage*/*max_usage*`))`

8.7 URM Architecture

The URM includes three servers: the `urmd` selection server, the `rmgr` query and command server, and the `sdsmgr` job server. The following sections describe these URM servers.

8.7.1 Selection Server (`urmd`(8))

The `urmd` selection server (the URM daemon) accepts job selection requests and responds with recommendations. This is the server that NQS, `login`, and other session initiators use to get initiation recommendations from URM. This server uses information about the state of the system and the backlog to recommend which job or jobs should be initiated. If any of the resources required by the job are not available, a recommendation is not issued and the job should not be started, because it may block or fail during its device reservation process.

The initiating service makes requests of the selection server by presenting a single job to add to the current backlog or by presenting its entire backlog of jobs otherwise eligible to initiate. There is also the capability of removing a job from the URM backlog. These functions are intended to provide enough information internal to URM to know the backlog, while giving flexibility to the service to add, delete, or rearrange jobs as required within the scope of the service policy. Services such as `login` that do not have a backlog simply present a job for evaluation. URM assumes that when a single job is presented for evaluation, a positive recommendation implies job initiation.

Device availability checks must succeed for URM to recommend initiation of a job. Resource requirements known to the selection server do not affect the execution of the job, because nothing is really reserved until the job initiates and makes its own reservation requests. This approach is taken to prevent incomplete or inaccurate resource information from affecting the actual execution of jobs.

The selection server expects that all relevant information about the resources needed by a job will be presented to it when a recommendation is requested. Only in this way are useful recommendations possible. All of the URM attributes associated with the job should be presented so they can also be evaluated. URM evaluates these attributes and recommends initiation only when all of the requirements are satisfied. Other information needed is the identification of the service (NQS, interactive, and so on) and, for each job, the following information:

- User name and / or user ID
- Memory and SDS size limits
- CPU time limit and nice value
- Tape usage information and other resource requirements of the job (batch only)
- Fair-share resource group ID
- Job priority (minimum rank)

8.7.2 Query and Command Server (`rmgr`(8))

The `rmgr` query and command server accepts requests for status information and replies to such requests as appropriate. This server is intended to support the needs of network-level scheduling and local administrators. This server also provides the configuration capability to URM.

For more information about the query and command server, see the `rmgr`(1) man page.

8.7.3 SDS Management (`sdsmgr`)

URM includes the `sdsmgr` job server, which monitors any interactive or batch session that uses secondary data space (SDS) in the SSD. When the `sdsmgr` is enabled, SDS space can be oversubscribed in the same manner as memory can be oversubscribed, by swapping SDS to `swapdev`. SDS in use and SDS changes requested by all sessions are determined from the session table by the kerninfo.c module of URM, which gathers load information. sdsmgr reads the sdsmap information to obtain the amount of SDS that is available but not in use at that time. If the amount of SDS in use at a particular point in time is greater than the physical amount of SDS space available after ldcache allocation, SDS is considered to be oversubscribed.

Changing the state of a session from active use of SDS to waiting for a turn at using this resource is termed *preemption*. This releases the SDS space for use by another session. Preemption of SDS space is done via the suspend () system call, which causes both process memory and SDS space for the session to be written to the swap device. Restoration of a session to a running state is accomplished by using the resume () system call, which removes the PC_SSPND flag from each process. When sched makes each process eligible for running and swapper swaps it in, the SDS space is reallocated. If the required SDS space is not available, the process is not swapped in.

To manage SDS, sdsmgr first makes a list of sessions using SDS; this list is ordered by nice value and time-in-queue. Preference is given to lower nice values and older jobs in order to provide fastest wall-clock turnaround. The amount of SDS in use (s_sdsuse in the session table) is further broken down into SDS in core and SDS swapped out (S_SUSPNDED flag set). The amount of SDS requested (s_sdsreqsz in the session table) is further broken down into additional SDS needed for swapin and SDS needed to grow in core. As long as no session needs more SDS, and no session needs to be swapped in to continue running, sdsmgr takes no actions.

If, after accounting for swapped sessions, there is a session trying to ssbreak for more SDS space, sdsmgr looks for a preemption candidate. Sessions with SDS are guaranteed not to be preempted for a specific length of wall-clock time (known as a *residence interval*), as long as they do not change their SDS allocation. Sessions in core that are waiting on ssbreak (that is, the kernel cannot allocate a larger SDS area) are preempted first. Exceptions to this rule are sessions that have just been initiated (sp->s_sdsuse == 0 and sp->s_sdsreqsz > 0). These sessions are given a chance to acquire SDS and start running before being preempted. Because NQS jobs acquire all their SDS specified on the qsub on the first ssbreak, NQS jobs should not be preempted when they first start running, as happened with qfdaemon. This behavior also assumes that users have not changed the environment variables SDSLIMIT, SDSINCR, and SDSMAXFR.

If no sessions trying to grow their SDS space are found, sessions that have had SDS allocated longer than the configured SDS residence interval are considered for preemption next. Sessions are preempted until enough SDS space has been freed up to satisfy the in-core SDS requested. If SDS space is available, and there are swapped out sessions, the job that has been out the longest, has the most favorable (lowest) nice value, and fits in the available SDS is swapped in. As long as as there are no sessions needing SDS to be swapped in, sessions currently using SDS are not preempted, even if they have exceeded the SDS residence interval. The `sdsmgr` does only one preempt for residence time exceeded and one restore per URM cycle.

8.8 URM Resources

This section describes the resources that URM controls. These are the resources usually associated with physical objects and with such notions as available devices or capacity.

URM resources include the following:

Resources	Description
CPU time	Like memory, this resource can be used to control job initiation but is more likely to be used to group jobs of similar duration for selection decisions.
Memory	Memory is not strictly a controlled resource, but URM manages oversubscription (swap control) and recommends against initiating jobs that require memory usage above a specified maximum. This is mostly needed for administrative controls, such as preventing jobs needing more than a specified amount of memory from being initiated during periods of heavy interactive use. Another use of this resource is to group jobs of similar usage, so that jobs with large memory requirements need not compete against one another during execution.
MPP	The MPP resources can be configured to allow multiple user applications to run concurrently (space sharing). The MPP hardware resources that must be shared are the barrier bits and the processing elements (PEs). URM monitors the MPP barrier synchronization mechanism (barrier pools for user-designated bits and for operating-system-designated bits). URM monitors the MPP administrative resource pools (sets of PEs designated for use by batch, interactive, or both job types).
SDS	Allocation of SDS space is evaluated both to deliver service to deserving jobs in the proper order and to manage resource conflicts to avoid oversubscription beyond manageable limits.

User limits on SDS space are controlled through the UDB limits on batch and interactive.

Share	Although shares, as used in the fair-share scheduler, are not resources in the sense of tapes, each user has potential to do work based on a priority derived from the share. Because this priority depends on the user's history, as well as the set of users active on the machine at a given time, URM must consider the effective priority of each request in the initiation recommendation.
Tape	URM supports tape resources as defined by the UNICOS tape subsystem. Eight different user limits are available to control tape resources.

8.9 URM Checkpointing

URM can be configured to checkpoint interactive sessions at shutdown, if requested by the owner of the session. The resulting restart images are kept in the URM `chkpnt` directory.

URM can also be configured to do periodic checkpoints of interactive or batch sessions, if requested by the owner of the session. URM sends checkpoint notices to NQS when a batch session requires periodic checkpointing; the resultant restart files are kept in the NQS `chkpnt` directory. URM manages restart files for interactive sessions in a special URM `chkpnt` directory.

8.9.1 Configuring URM Checkpointing

Checkpointing is done only on an individual session basis. The `chkptint`(1) command provides the mechanism by which any user is able to specify that checkpointing be done at shutdown for an interactive session or that periodic checkpointing is desired for an interactive or a batch session.

This command has one required option, of the form `-s` *sec*, which specifies the requested checkpoint frequency in seconds. When URM is running in checkpoint only at shutdown mode, all that is required is that the `chkptint -s #` be nonzero; `chkptint -s 0` can be used to turn off the automatic or periodic checkpoint request for that session. The `chkptint` command can be part of a script or placed in a user's `.cshrc` or `.profile` file.

The following URM objects (as taken from a `rmgr-> view /urm` display) are used to configure URM checkpoint features:

```
LVrwr-  0    0 Dec 28 08:13 <restart_switch>  "Force"
LVrwr-  0    0 Dec 28 08:13 <chkpnt_switch >  "Auto"
LVrwr-  0    0 Dec 28 08:13 <min_interval  >   Int, 1800
LVrwr-  0    0 Dec 28 08:13 <interval_type >  "CPU, Clock"
LVrwr-  0    0 Dec 28 08:13 <retain_chkpnt >   Int, 432000
LVr-r-  0    0 Dec 28 08:13 <shutdown      >   Int, 0
LVr-r-  0    0 Dec 28 08:13 <shut_done     >   Int, 0
LVrwr-  0    0 Dec 28 08:13 <restart_cmd   >  "UPATH/irstart"
LVrwr-  0    0 Dec 28 08:13 <chkpnt_path   >  "PPATH/chkpnt"
LVrwr-  0    0 Dec 28 08:13 <chkpnt_cmd    >  "UPATH/intchkpt"
```

The `restart_switch` object indicates whether or not to use the `RESTART_FORCE` flag on the restart system call when restarting an interactive session.

The `chkpnt_switch` object defines the type of checkpointing desired. `No` turns both features off; `Shutdown` turns on checkpointing for interactive sessions at shutdown; and `Auto` turns on periodic checkpointing of batch and interactive sessions and specifies checkpointing for interactive sessions at shutdown as well.

The `min_interval` object defines the minimum interval (in seconds) between checkpoints with which URM will comply. If a user requests a checkpoint interval less than this minimum, the minimum will be used. This constraint applies only to periodic checkpoint behavior.

The `interval_type` object defines the checkpoint criteria. `CPU` means user-plus-system CPU time; `Clock` means elapsed wall-clock time. This constraint applies only to periodic checkpoint behavior.

The `retain_chkpnt` object defines the length of time (in seconds) URM will keep a restart image before deleting it. Deleting these images prevents taking up disk space with unwanted restart files.

The `shutdown` and `shut_done` flags indicate whether or not shutdown is in progress or completed. When `shutdown` and `shut_done` are both 0, no shutdown is in progress. If `shutdown` is equal to 1, URM goes through the process of looking for sessions to checkpoint, but does not terminate when completed. If `shutdown` is equal to 2, a checkpoint and terminate sequence has been requested. When all shutdown checkpointing is complete, `shut_done` is set to 1.

The `restart_cmd` object defines the name of the command `rmgr` executes to restart a selected interactive session.

The chkpnt_path object defines the path to the URM checkpoint directory. The default is /usr/adm/urm/chkpnt.

The chkpnt_cmd object defines the name of the command URM spawns as a separate process to checkpoint an interactive session.

For periodic checkpointing of interactive sessions, URM scans the session table, notes when a checkpoint interval has been set by using the chkptint command, and sets a timer for that session. When the requested CPU or wall-clock interval has elapsed, URM will fork and exec a process to do the checkpoint for the interactive session. The timer is reset after a successful checkpoint.

When URM determines that it is time to do a periodic checkpoint of a batch job, URM sends a chkpnt request, defined as part of the interface between NQS and URM, along with job sequence number, MID, and SID. NQS creates the restart file in the NQS checkpoint directory, as if a qchkpnt command had been issued by the user.

8.9.2 Managing Restart Images

Several rmgr subcommands aid in the management of restart images. Upon logging in, an interactive user can query URM, using the rmgr utility view restart subcommand, to see if the user has any checkpointed sessions available.

```
rmgr-> view restart kcz
Restart images belonging to User <kcz>, UID 343 on path /ptmp/urm/chkpnt/kcz
        <1> 12281123.2640
        <2> 12281153.2640
        <3> 12281223.2640
User <kcz> has 3 restart images.
```

To decide which of the saved restart images to try to restart, a user can use the rmgr utility view restart subcommand to obtain a detailed description of the restart image.

```
rmgr-> view restart kcz 2
Restart file /tmp60/kcz/chkpnt/kcz/12281153.2640 belongs to uid 343
Restart file 12281153.2640 for session 2640 created  Dec 28 11:53

  Restart file session characteristics:
  #procs = 5;  #tasks = 5;  pgrp inheritance = 69747;  nice value = 20.
  PID of session parent = 69746;  PID of foreground process group = 76223.
  User cputime = 118212137; system cputime = 196492829
        cpulimit[0] = 4611686018427387903
        cpulimit[1] = 4611686018427387903
        cpulimit[2] = 4611686018427387903
  Memory in use = 868; memhiwat = 2047; memlimit = 35184372088831
  SDS in use = 0; sdshiwat = 0; sdslimit = 900000
  Max #of procs allowed = 100

  Mtask    1: name: csh pid: 69747  #sibs:    1.
        process group pid = 102032, parent pid = 69746
        Memory size= 124 klics, swap_image size= 0 klics.
        SDS allocated= 0 klics, SDS requested= 0 klics.

  Mtask    2: name: zup pid: 70005  #sibs:    1.
        process group pid = 102032, parent pid = 69747
        Memory size= 156 klics, swap_image size= 0 klics.
        SDS allocated= 0 klics, SDS requested= 0 klics.

  Mtask    3: name: csh pid: 70030  #sibs:    1.
        process group pid = 102032, parent pid = 70005
        Memory size= 96 klics, swap_image size= 0 klics.
        SDS allocated= 0 klics, SDS requested= 0 klics.

  Mtask    4: name: man pid: 76223  #sibs:    1.
        process group pid = 102032, parent pid = 70030
        Memory size= 148 klics, swap_image size= 0 klics.
        SDS allocated= 0 klics, SDS requested= 0 klics.

  Mtask    5: name: more pid: 76224  #sibs:    1.
        process group pid = 102032, parent pid = 76223
        Memory size= 112 klics, swap_image size= 0 klics.
        SDS allocated= 0 klics, SDS requested= 0 klics.
User <kcz> has 2 restart images.
```

The `rmgr` utility `delete restart` subcommand allows users to delete unwanted saved restart images. For example:

```
rmgr-> delete restart kcz 1
Deleted restart image <1> for user kcz
rmgr->
```

To request that a previously saved session be restarted, a user first views the saved images and then specifies that a particular image be restarted. The existing session with `rmgr` is eliminated, and the terminal connected to the login shell is saved in the restart session.

The following example restarts a saved session that was in the middle of a `man ps` command. Output resumes where it left off when the job was saved.

```
rmgr-> view restart kcz
Restart images belonging to User <kcz>, UID 343 on path /tmp60/kcz/chkpnt/kcz
        <1> 12281153.2640
        <2> 12281223.2640
User <kcz> has 2 restart images.

rmgr-> restart kcz 1
     When a process has exited and has a parent, but the parent has not
     waited for it, the process is marked <defunct>.

   Format Specification
     The -o option allows the output format to be specified under user
     control.  The format specification consists of a list of field names
--More--
```

8.9.3 Checkpointing at Shutdown

The `rmgr` subcommands used in a controlled shutdown are `set shutdown stopdaemon`, or `set shutdown` and `stopdaemon` used separately and sequentially. When `set shutdown stopdaemon` is used, `urmd` terminates after all checkpointing has completed. When `set shutdown` is used, any requested checkpoints are performed, but `urmd` continues. When `stopdaemon` is used, `urmd` terminates without performing any checkpointing.

8.10 Tuning URM

This section provides information specific to tuning the Unified Resource Manager (URM). It covers the following topics:

- Tuning URM control settings
- Tuning URM job selection criteria
- Using URM with NQS

The examples in this section demonstrate changing URM settings with the `rmgr`(1) command. Any changes made with `rmgr` affect the running version of URM only; when the URM daemon is stopped, these changes are lost. To make permanent changes to URM settings, use the menu system (`install`(8) command), as described in Section 8.4, page 363.

8.10.1 Tuning URM Control Settings

URM is released with a set of default control settings, including the following:

- Machine target values (`/machine/target`)
- Monitoring cycles (`/urm`)
- Group scheduling control (`/machine/target/group_sched`)
- Load smoothing factors (`/machine/rate`)

The remainder of this section describes how to change each of these control settings.

8.10.1.1 Machine Target Values

The URM machine target values set system-wide resource usage boundaries. URM does not recommend initiation of jobs specifying resources that exceed these targets.

When determining the recommendation status of each job, URM compares system resource loads against the following targets:

- Memory oversubscription target
- SDS oversubscription target
- Maximum session target
- Tape limit target
- MPP limit targets

The values for the tape and MPP limit targets are automatically updated by URM through its automatic configuration capability and should not be changed manually. However, if your site does not have tapes, the automatic configuration of tape usage targets should be disabled.

Use the `rmgr` command to view the current machine target values, as shown in the following example:

```
rmgr => view /machine/target
        .
        .
        .
 LVrwr- 0 0 Sep 16 06:21 <sds_os > Float, 2.000000
 LVrwr- 0 0 Sep 16 06:21 <memory_os > Float, 2.000000
        .
        .
        .
 LVrwr- 0 0 Sep 16 06:21 <tape > Int, 2
 LVrwr- 0 0 Sep 16 06:21 <jobcount > Int, 300
 LVrwr- 0 0 Sep 16 06:21 <pe > Int, 0
 LVrwr- 0 0 Sep 16 06:21 <bb > Int, 0
```

The `/machine/target` display also shows other URM values, such as the group scheduling controls; these values are discussed in following sections.

To change a machine target value, use the following `rmgr` directive:

`/machine/target/`*resource*=*value*

To permanently change the configuration, use the following menu in the menu system:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Machine target values
```

The remainder of this section describes the configurable machine targets for the memory, SDS space, and active jobs.

8.10.1.1.1 Memory Oversubscription Target

The amount of memory in use is called the *memory load*. URM computes the memory load by adding the size of all running jobs and recommended jobs. The size of a running job is calculated from the actual size in the kernel session table, *smoothed* by the requested size of the job (see Section 8.10.1.4, page 414). The size of a recommended job is the amount of memory the job requested. If a job did not request memory requirements (that is, interactive jobs), the default memory usage is used; see Section 8.10.2.2.2, page 425, for more information.

The memory oversubscription target, `/machine/target/memory_os`, sets an upper limit on the total amount of user memory (including swap space) that can be in use at the same time. If the actual memory load matches or exceeds the target value, no jobs are initiated until the load drops. This limitation includes only jobs from session initiators that have been configured to allow URM control of job initiation; for more information, see Section 8.10.2.2.2, page 425.

Estimating memory use in this manner allows URM to plan ahead for future memory use, if smoothing factors are used, because most jobs do not use the full amount of declared memory until they have been running for a period of time.

The memory oversubscription target is a multiplier, or percentage value, instead of an absolute value. The default multiplier, 2.0, sets the memory target to twice the total amount of user memory configured.

If a high amount of swapping is affecting system performance, consider decreasing the memory oversubscription multiplier. For example, to change the memory oversubscription target to 1.5, enter the following `rmgr` directive:

```
/machine/target/memory_os = 1.5
```

To permanently change the configuration, use the following menu in the menu system:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Machine target values ->
        Memory oversubscription multiplier
```

Note: The maximum value for the memory oversubscription target is limited by the amount of swap space configured on the system. Do not set this target to a value larger than the amount of available swap space. (If SDS space is configured, the sum of the memory oversubscription target and the SDS oversubscription target should be less than or equal to the amount of available swap space.)

The following example demonstrates how to calculate a value for memory oversubscription that corresponds to previous usage of the system by using the data from the system `sar`(1) logs. For example, assume that the `sar` log for a typical day shows the following average values for memory use:

- 53730 clicks of total user memory available (*umemtot*)
- 38144 clicks of user memory in use (*umemuse*)
- 46769 clicks of swap space in use (*swapuse*)

Use the following equation to calculate the memory oversubscription target value for URM:

memory_os = (*umemuse* + *swapuse*) / *umemtot*

```
          = (38144 + 46769) / 53730

          = 1.6
```

Setting `/machine/target/memory_os` to 1.6 will result in memory use with URM that is similar to previous system behavior.

8.10.1.1.2 SDS Oversubscription Target

The SDS oversubscription target, `/machine/target/sds_os`, sets an upper limit on the total amount of SDS space that can be in use at the same time. If the actual SDS load matches or exceeds the target value, no jobs requesting SDS space are initiated until the load drops.

The SDS oversubscription target is a multiplier, or percentage value, instead of an absolute value. The default multiplier, 1.5, sets the SDS target to one and one-half times the amount of available SDS space. This target could be increased to allow URM to recommend more jobs using SDS space (which might increase swapping as well), as in the following example:

```
/machine/target/sds_os = 2.0
```

To permanently change the configuration, use the following menu in the menu system:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Machine target values ->
        SDS oversubscription multiplier
```

Note: The maximum value for the SDS oversubscription target is limited by the amount of swap space configured. Do not set this target to a value larger than the amount of available swap space. (If SDS space is configured, the sum of the memory oversubscription target and the SDS oversubscription target should be less than or equal to the amount of available swap space.)

In addition to monitoring SDS load, URM manages SDS space by preempting the jobs using SDS to allow other jobs a chance to use SDS space. (Both batch and interactive jobs are affected by this feature.) For more information, refer to Section 8.10.1.2.5, page 411.

8.10.1.1.3 Maximum Session Target

The maximum session target, `/machine/target/jobcount`, determines the boundary at which URM stops recommending the initiation of any more jobs. By default, this target is set to the size of the kernel session table (set by the `NSESS` parameter).

Note: Setting the active job target to too small a value could result in wasted CPU resources, because URM would not recommend jobs even when adequate resources are available. To control job count and prevent the system from becoming overcommitted, use the job targets for the individual session initiators. Refer to Section 8.10.2, page 418, for more information on these targets.

8.10.1.2 Monitoring Cycles

The URM monitoring cycles control the frequency of load monitoring, batch job ranking, and SDS preemption. These cycles define the minimum possible delay for initiation of batch jobs. (Interactive jobs are always handled immediately.) The controlling monitoring cycle is called the *main loop*; this cycle calls all other monitoring cycles. Therefore, all other cycles cannot occur more frequently than the main loop cycle.

The monitoring cycles can be changed to increase or decrease URM's sensitivity to fluctuations in job load. This section describes changing the interval, or delay, for the main loop and the subsidiary cycles.

8.10.1.2.1 URM Main Loop

The URM main loop consists of the following operations:

Operation	Description
Kernel information check	Checks system load and configuration information, monitors SDS usage
Job scheduling check	Ranks and recommends any batch jobs that are waiting in its backlog

Share evaluation check	Reevaluates the fair-share hierarchy
SDS residence management	Manages SDS usage

The main loop is controlled by the main loop delay, `/urm/sleep_time`; it is set to 10 seconds by default. This delay specifies the amount of time URM waits before performing the subsidiary cycles. Each subsidiary cycle in the main loop has its own delay; these should be set to a multiple of the main loop delay. If they are not, the main loop delay will take precedence.

To increase the main loop delay to 20 seconds, for example, enter the following `rmgr` directive:

```
/urm/sleep_time = 20
```

To permanently change the configuration, use the following menu in the menu system. However, it is recommended that you **not** change this value.

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      URM control settings ->
        Main loop sleep period
```

8.10.1.2.2 Kernel Information Check

The kernel information delay, `/urm/info_delay`, controls the frequency of checks for resource loads and configuration information; it is set to 10 seconds by default. Changing this value is **not** recommended. Refer to Section 8.10.1.4, page 414, for more information on the kernel information delay.

8.10.1.2.3 Job Scheduling Check

The job scheduling check controls the frequency of batch job ranking and batch job recommendation. During the job scheduling check, URM ranks all jobs in the backlog and makes any possible recommendations.

The job scheduling delay, `/urm/sched_delay`, is set to 10 seconds by default. This delay, along with the main loop delay, defines the actual minimum delay for batch jobs to receive an initiation recommendation after they are registered with URM.

8.10.1.2.4 Share Evaluation Check

If the fair-share scheduler is enabled, URM periodically evaluates the machine share (normalized share) of each resource group or shareholder (account ID) for use in the batch job ranking calculation.

The share evaluation cycle is controlled by the share evaluation delay, `/urm/share_eval`, which is set to 900 seconds (15 minutes) by default. Resource group hierarchies and relative priorities among the resource groups tend to change infrequently, so this evaluation is not performed as often as the other cycles.

The countdown timer for the fair-share evaluation period, `/urm/share_to_go`, contains the amount of time remaining until the next evaluation. Use the following `rmgr` directive to view this value:

```
view /urm/share_to_go
```

If relative fair-share priorities change on a one-time basis (for example, when the share allocations are changed in the UDB), set `/urm/share_to_go` to 0; this ensures that share evaluation will be performed during the next main loop cycle.

For more information on the fair-share scheduler, see "Fair-share scheduler," Chapter 4, page 193.

8.10.1.2.5 SDS Residence Management

URM controls SDS oversubscription by preempting jobs, both batch and interactive, to ensure that other jobs waiting for SDS space get a chance to use it. Once during each main loop, URM checks to see if SDS space is in use, then calls the `sdsmgr` program to handle the actual preemption.

The SDS residence interval is controlled by `/urm/sds_residence`; this value is set to 900 seconds (15 minutes) by default. You can disable URM management of SDS space by setting `/urm/sds_residence` to 0.

To permanently change the configuration, use the following menu in the menu system:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      URM control settings ->
        SDS residency time in secs
```

The following example shows the `view sds` directive to the `rmgr` command on a system with SDS oversubscription:

```
rmgr-> view sds

SDSwork: SDS residence interval set to 900
Number of sessions using SDS  = 2
Number of preempted SDS jobs = 0

SDS jobs:
         preempted   = 0
         restoring   = 0
         swapped     = 1
         in core     = 1

SDS units:
         physical size            12288
         allocated                23296
         requested                0
         available                128
         in memory                12160
         swapped out              11136
         needed by running   jobs 0
         needed by suspended jobs 0
         being preempted out      0
         being restored to memory 0
SDS state is OVERSUBSCRIBED by 11008
```

8.10.1.3 Group Scheduling Control

During the batch job ranking phase (see Section 8.10.2.1.1, page 419), the default behavior for URM is to select the jobs with the best (highest) rank to recommend for initiation. However, some jobs might receive a consistently low rating because of atypical resource usage. Using rank alone as a selection criterion can prevent these jobs from being recommended for initiation on busy systems. To prevent this situation, URM has a *group scheduling control* feature to control job recommendations by group characteristics, or similarity of resource usage, as well as the batch job ranking calculation.

By default, group scheduling control is disabled. All jobs fall into the same default group (also called the *batch job pool*). Group scheduling control is enabled by setting `/machine/target/group_sched` to 1. When this feature is enabled, URM divides all batch jobs into four groups (defined by computing the average rank and standard deviation) based on how similar each job is to the average job. Group 1 contains jobs closest to the average, and group 4 contains jobs farthest from the average. URM stores a count of jobs for each group in four `/machine/target/jobs_in_sg` values.

URM also monitors the recommendation history for each group, that is, the number of jobs selected, or *picked*, from each group. The group scheduling control feature allows you to change the job selection percentage to favor unusual or low ranked jobs by setting the *pick value* (selection percentage) for each group. The pick values define how often a job is selected from each group. URM stores integer values in four `/machine/target/pick_in_sg` values; these are converted to relative values, or percentages. By default, a total value of 20 is used to divide the group pick values; the pick value for group 1 is 9 (45% of jobs are selected from this group), group 2 is 7 (35%), group 3 is 3 (15%), and group 4 is 1 (5%). The recommendation history is stored in four `/machine/target/rec_from_sg` values.

The following `rmgr` directive displays sample URM values used for group scheduling control:

```
rmgr => view /machine/target
                .
                .
                .
 LVrwr- 0 0 Sep 20 07:23 <group_sched > Int, 1
                .
                .
                .
 LVr-r- 0 0 Sep 20 07:23 <jobs_in_sg1 > Int, 20
 LVr-r- 0 0 Sep 20 07:23 <jobs_in_sg2 > Int, 11
 LVr-r- 0 0 Sep 20 07:23 <jobs_in_sg3 > Int, 4
 LVr-r- 0 0 Sep 20 07:23 <jobs_in_sg4 > Int, 2
 LVr-r- 0 0 Sep 20 07:23 <rec_from_sg1 > Int, 45
 LVr-r- 0 0 Sep 20 07:23 <rec_from_sg2 > Int, 35
 LVr-r- 0 0 Sep 20 07:23 <rec_from_sg3 > Int, 15
 LVr-r- 0 0 Sep 20 07:23 <rec_from_sg4 > Int, 5
 LVrwr- 0 0 Sep 20 07:23 <pick_in_sg1 > Int, 9
 LVrwr- 0 0 Sep 20 07:23 <pick_in_sg2 > Int, 7
 LVrwr- 0 0 Sep 20 07:23 <pick_in_sg3 > Int, 3
 LVrwr- 0 0 Sep 20 07:23 <pick_in_sg4 > Int, 1
                .
                .
                .
```

The line for `group_sched` shows that group scheduling control is enabled. The number of jobs in each group are displayed in the values `jobs_in_sg1`, `jobs_in_sg2`, `jobs_in_sg3`, and `jobs_in_sg4`. The recommendation history is shown in the values `rec_from_sg1`, `rec_from_sg2`, `rec_from_sg3`, and `rec_from_sg4`. The default pick values are shown in `pick_in_sg1`, `pick_in_sg2`, `pick_in_sg3`, and `pick_in_sg4`.

To change the percentages for group selection, modify the `pick_in_sg` values. The following example enables group scheduling control and sets pick values so that jobs are selected from each group on an equal basis:

```
/machine/target/group_sched = 1
/machine/target/pick_in_sg1 = 1
/machine/target/pick_in_sg2 = 1
/machine/target/pick_in_sg3 = 1
/machine/target/pick_in_sg4 = 1
```

Note: The sum of the `pick_in_sg` values should be less than or equal to the maximum number of initiations for the session initiator (that is, the value set by `/machine/jobmax/` *initiator* `/start_max`). See Section 8.10.2.1.3, page 423, for more information.

To permanently enable group scheduling control and make permanent changes to the pick values, insert the appropriate `rmgr` directives in the local configuration file by using the following menu in the configuration menu system:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      URM control settings ->
        Full pathname of local URM config file
```

8.10.1.4 Load Smoothing Factors

Each resource that URM monitors has a specific *load* associated with it (stored in `/machine/load/` *resource*) that indicates the amount of the resource in use. URM tracks the load for the following resources:

- Memory
- Total active sessions
- SDS usage
- Tape usage

- MPP resource usage
- Individual session initiators: `batch`, `login`, `rsh`, `rexec`, `null`, `ftp`, and `cron`.
- Site-specific session initiators `site1`, `site2`, and `site3`, if defined (see Section 8.4.5, page 369, for more information)

Use the `rmgr` directive `view /machine/load` to display the current load values, as in the following example:

```
rmgr => view /machine/load

 LVrwr- 0 0 Sep 17 08:22 <memory > Int, 41905
 LVrwr- 0 0 Sep 17 08:22 <shared_txt > Int, 10660
 LVrwr- 0 0 Sep 17 08:22 <sesstab_mem > Int, 54222
 LVrwr- 0 0 Sep 17 08:22 <sds > Int, 0
 LVrwr- 0 0 Sep 17 08:22 <tape > Int, 1
 LVrwr- 0 0 Sep 17 08:22 <bb > Int, 0
 LVrwr- 0 0 Sep 17 08:22 <pe > Int, 0
 LVrwr- 0 0 Sep 17 08:22 <site3 > Int, 0
 LVrwr- 0 0 Sep 17 08:22 <site2 > Int, 0
 LVrwr- 0 0 Sep 17 08:22 <site1 > Int, 0
 LVrwr- 0 0 Sep 17 08:22 <rsh > Int, 1
 LVrwr- 0 0 Sep 17 08:22 <rexec > Int, 0
 LVrwr- 0 0 Sep 17 08:22 <null > Int, 31
 LVrwr- 0 0 Sep 17 08:22 <login > Int, 35
 LVrwr- 0 0 Sep 17 08:22 <ftp > Int, 0
 LVrwr- 0 0 Sep 17 08:22 <cron > Int, 1
 LVrwr- 0 0 Sep 17 08:22 <batch > Int, 1
```

Instead of using actual load values to estimate available system resources, URM applies a *smoothing function* to each resource load before using the load in its recommendation calculations. The smoothing function prevents overreaction to fluctuations in resource use and helps protect users from an erratic system response. (The load smoothing function is also called a *moving average* or *rolling average*.) The amount of smoothing is controlled by smoothing factors.

Each resource has its own smoothing factor, which can be set to a proportional value between 0.0 and 1.0, depending on how accurately you want it to represent changes in actual resource loads. The value 1.0 specifies that URM should match load values to actual changes; the value 0.0 specifies no change to existing load values. The closer a smoothing factor is to 1.0, the more quickly load changes will be adjusted. The closer to 0.0, the less effect actual resource usage has on job selection. For example, the smoothing factors for the session initiators (batch, `ftp`, `login`, and so on) are set to 1.0 by default; this allows URM to enforce any session boundaries (`jobcount` target values).

Note: Smoothing occurs only when resource use drops. Increases in resource use are reflected immediately.

The graph in Figure 9 shows how this adjustment takes place. This example compares the results of four different smoothing factors for the memory load:

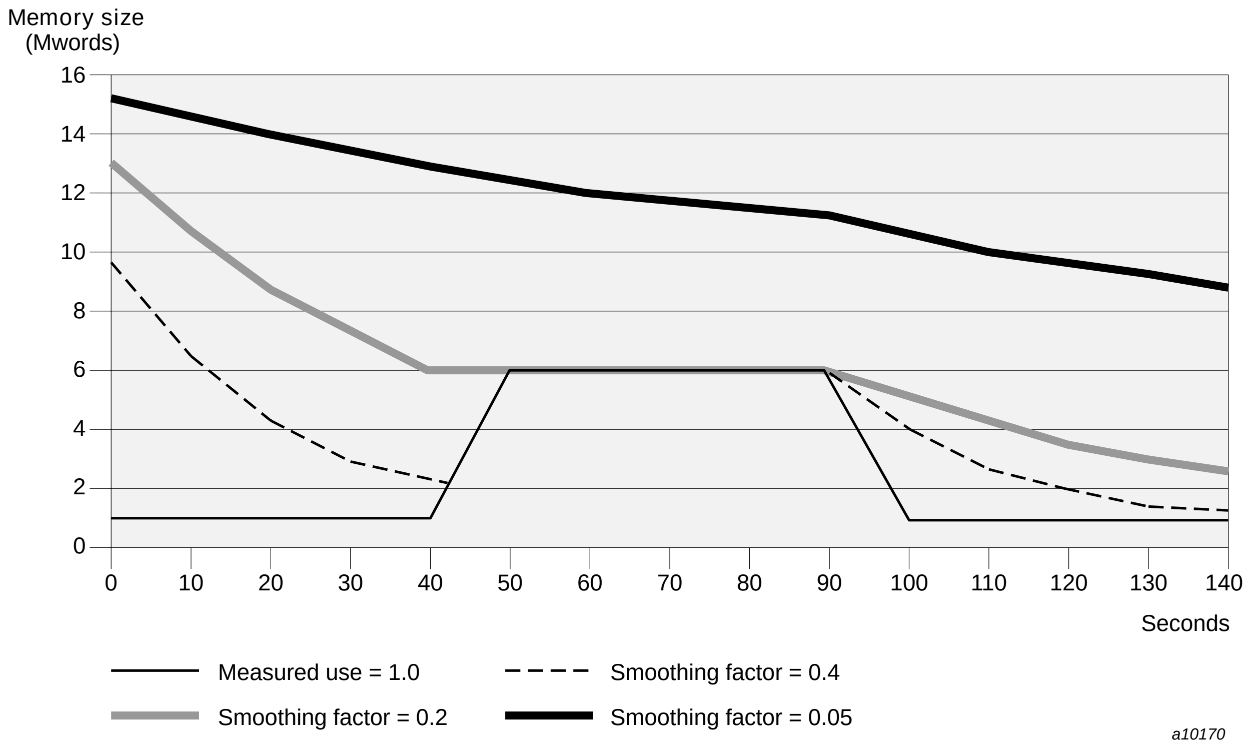


Figure 9. Example of Different Smoothing Factors

Note: Smoothing factors of values at or near 0.0 is not recommended. Setting a smoothing factor below 0.1 would cause URM to cease updating the load values. Using a very low value for a smoothing factor can cause URM to react too slowly to changes in resource loads.

The graph in Figure 10 shows the effect of the default smoothing factor for memory (0.8) on URM calculations of memory load over time:

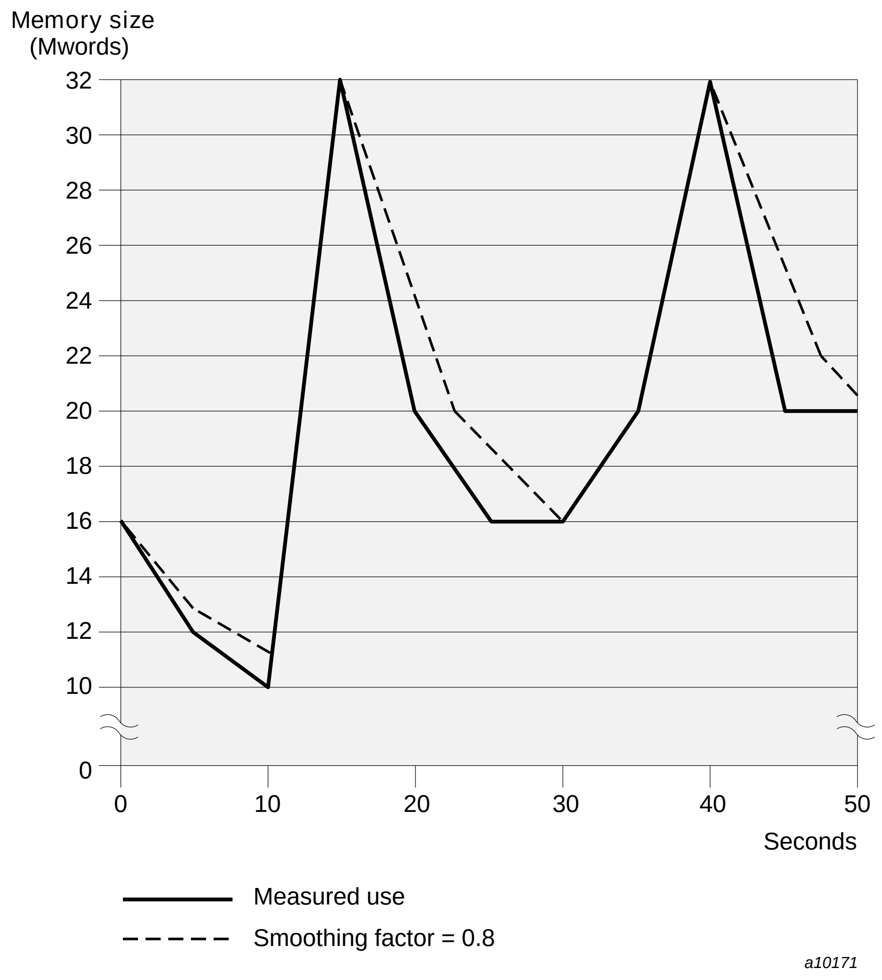


Figure 10. Default Smoothing Factor for Memory Load

Use the `rmgr` command to display the current smoothing factors, as in the following example:

```
rmgr =>view /machine/rate
```

```
    LVrwr-  0    0 Oct 25 23:59 <memory         > Float, 0.800000
    LVrwr-  0    0 Oct 25 23:59 <sds            > Float, 0.800000
    LVrwr-  0    0 Oct 25 23:59 <tape           > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <bb             > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <pe             > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <site3          > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <site2          > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <site1          > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <rsh            > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <rexec          > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <null           > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <login          > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <ftp            > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <cron           > Float, 1.000000
    LVrwr-  0    0 Oct 25 23:59 <batch          > Float, 1.000000
```

To change any of the load smoothing factors, enter the following `rmgr` directive:

`/machine/rate/`*resource* `=` *factor*

The value *resource* specifies the resource, such as `memory`, and *factor* specifies a value between 0.1 and 1.0.

To permanently change the configuration, use the following menu in the menu system:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      URM machine load evaluation rates
```

8.10.2 Tuning URM Job Selection Criteria

In addition to controlling system resource targets, URM controls the job selection criteria for each session initiator. (The job selection criteria are called *individual session initiator targets* in the menu system.)

For the batch session initiator, URM monitors the session maximum targets (in `/machine/jobmax` values) and the resource loads (in `/machine/load` values). For nonbatch session initiators, URM monitors the defaults (in `/machine/default` values) and the resource loads (in `/machine/load` values) for each session initiator.

The following sections describe the job selection criteria used for batch jobs and for interactive jobs.

8.10.2.1 Batch Jobs

URM tracks the following job selection criteria for batch jobs:

- Job count; total number of active sessions allowed for the batch session initiator
- Maximum number of batch job initiations per scheduling cycle
- Memory request for a batch job
- CPU request for a batch job
- Tape request for a batch job
- SDS request for a batch job
- MPP requests for a batch job

In addition to the job selection criteria, URM checks the machine target values and batch job ranking before recommending a job for initiation. (Refer to Section 8.10.1.1, page 405, for more information about target values; see the following section for information on the batch job ranking calculation.)

Use the following `rmgr` directive to change any of the job selection criteria for the batch session initiator:

`/machine/jobmax/batch/`*resource* = *value*

To permanently change the configuration, access the following menu and select `batch` as the session initiator name:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      URM control settings ->
        Individual session initiator targets
```

The following sections discuss each of the batch job selection criteria.

8.10.2.1.1 Batch Job Ranking Calculation

URM assigns a rank to each job based on the job's *resource attributes*, or declared amount of resources requested by the job (see "Using URM with NQS," Section

8.4.13, page 376, for more information). During batch job ranking, each resource attribute is normalized (converted to a proportional value between 0.0 and 1.0) and multiplied by a weighting factor. Each resource has an associated weighting factor that can be configured to a smaller or larger value based on the desired importance for that resource in the batch job ranking calculation.

By default, all weighting factors are equal (set to 0.5). Increasing the value of a weighting factor increases its relative importance in the ranking; setting a weighting factor to 0 effectively removes that resource from the calculation.

URM uses following equation for batch job ranking:

```
rank = (share * share_wt) +
       (usage * usage_wt) +
       (service_pri * service_wt) +
       (age * age_wt) +
       (cpu * cpu_wt) +
       (memory * mem_wt) +
       (tape * tape_wt) +
       (SDS * sds_wt) +
       (bb * bb_wt) +
       (PE * pe_wt) +
       (PE_time * petime_wt) +
       (prevrun_boost) +
       (min_rank)
```

The resources and associated weighting factors are represented in this equation as follows:

share * `share_wt`

Fair-share component. If the fair-share scheduler is not enabled, the value of this component is 0. If fair-share is enabled, *share* represents the normalized share value of the job owner's resource group or account; `share_wt` represents the fair-share weighting factor. URM evaluates the system share hierarchy at a rate determined by `/urm/share_eval` (default 900 seconds). This weighting factor can be adjusted by changing `/urm/share_wt` with the `rmgr` command.

usage * `usage_wt`

CPU usage component. The value *usage* represents the normalized usage maintained for the job's user in the `shrusage` field in the UDB; `usage_wt` represents the usage weighting factor. If the user already has jobs active on the system, the current usage is obtained; otherwise, the decayed usage from the UDB is used. The usage weighting factor can be adjusted by changing `/urm/usage_wt` with the `rmgr` command.

service_pri * `service_wt`

NQS queue priority component. The value *service_pri* represents the interqueue priority from NQS; `service_wt` represents the service weighting factor. If `service_wt` is set to 0, the NQS interqueue priorities no longer take effect in the URM priority calculation. This weighting factor can be adjusted by changing `/urm/service_wt` with the `rmgr` command.

age * `age_wt`

Age-in-queue component. The value *age* represents the length of time the job has been waiting for initiation from the URM queue; `age_wt` represents the age weighting factor. The age of the oldest job is 1.0, and the age of the youngest job is 0.0. The age weighting factor can be adjusted by changing `/urm/age_wt` with the `rmgr` command.

cpu * `cpu_wt`

Requested CPU resource component. The value *cpu* represents the normalized CPU requirement specified with each batch job; `cpu_wt` represents the CPU weighting factor. The `cpu_wt` value can be adjusted by changing `/urm/cpu_wt` with the `rmgr` command.

memory * `mem_wt`

Requested memory resource component. The value *memory* is the normalized memory requirement specified with each batch job; the largest job is 0.0, and the smallest job is 1.0. The value `mem_wt` represents the memory weighting factor. This weighting factor can be set by changing `/urm/mem_wt` with the `rmgr` command.

tape * `tape_wt`

Requested tape resource component. The value *tape* is the normalized tape device requirement specified with each batch job; *tape* is set to 0.0 for the job requesting the most tape resources, and to 1.0 for the job requesting the least resources.

The value `tape_wt` represents the tape weighting factor. This weighting factor can be set by changing `/urm/tape_wt` with the `rmgr` command.

SDS `* sds_wt`

Requested SDS resource component. The value *SDS* represents the normalized SDS requirement specified with each batch job; *SDS* is set to 0.0 for the job requesting the most SDS space, and to 1.0 for the job requesting the least resources. The value `sds_wt` represents the SDS weighting factor. This weighting factor can be set by changing `/urm/sds_wt` with the `rmgr` command.

BB `* bb_wt`

PE `* pe_wt`

PE_time `* petime_wt`

These components define the MPP resource attributes for MPP barriers, processor elements, and requested PE time job attributes, respectively.

prevrun_boost

Value to improve the rank of a previously checkpointed job.

min_rank

Minimum rank (NQS job priority).

To make a permanent configuration change to the weighting factors, use the following menu in the menu system:

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      Weighting factors for the selector
```

8.10.2.1.2 Batch Job Count

The job count, or maximum number of active batch sessions, is controlled by `/machine/jobmax/batch/jobcount`. By default, this value is set to the size of the kernel session table (set by the `NSESS` parameter) during URM's automatic configuration process. Changing this value affects the relative proportions of batch and interactive jobs on the system.

8.10.2.1.3 Batch Maximum Initiation

The maximum number of batch initiations allowed during a single scheduling cycle is controlled by the `/machine/jobmax/batch/start_max` value. When this limit is reached, URM does not recommend any more batch jobs until its next scheduling cycle. The default for this value is 10.

If group scheduling control is enabled, the sum of the `pick_in_sg` values should be less than or equal to this value; see Section 8.10.1.3, page 412, for more information.

8.10.2.1.4 Batch Memory Request Target

The memory request target for batch jobs is controlled by `/machine/jobmax/batch/memory`. This value determines the maximum size, in clicks, of a batch job. Jobs requesting more memory than this value will not be recommended.

By default, this value is configured automatically to the maximum amount of user memory available. Consider increasing this value if your site runs batch jobs that have two or more processes whose total memory requirements exceed the amount of available user memory.

8.10.2.1.5 Batch CPU Request Target

The CPU request target is controlled by `/machine/jobmax/batch/cputime`. This value determines the maximum amount of CPU time, in seconds, for a batch job. Jobs requesting more CPU usage than this value are not recommended for initiation. By default, this value is set to 9999999 seconds.

8.10.2.1.6 Batch Tape Request Target

The batch tape request target is controlled by `/machine/jobmax/batch/tape`. This value specifies the maximum number of tape devices allowed for a batch job. Jobs requesting more devices than this value will not be recommended. This value is set to the number of online tape devices during URM's automatic configuration process.

8.10.2.1.7 Batch SDS Request Target

The SDS request target is controlled by `/machine/jobmax/batch/sds`. This value specifies the maximum amount of SDS space, in clicks, that a batch job can use. Jobs requesting a larger SDS usage than this value are not recommended for initiation. This value is set to the amount of available SDS space during URM's automatic configuration process. Consider increasing this value if your site runs jobs whose total SDS requirements exceed the amount of available space.

8.10.2.2 Interactive Jobs

Interactive jobs (jobs initiated from all session initiators except `batch`) are evaluated for initiation by URM immediately. Unlike batch jobs, no backlog queue is maintained by URM. In most cases, if URM does not recommend the job for initiation, the session initiator will terminate it. This is the case for jobs submitted by `login`.

In order for URM to recommend an interactive job for initiation, the requested resources for the job must not exceed the following target and maximum values:

- Active job target for the system (`/machine/target/jobcount`). For more information, refer to Section 8.10.1.1.3, page 409.
- Interactive job count; maximum number of active sessions for the specified interactive session initiator.
- Memory oversubscription target for the system.

8.10.2.2.1 Interactive Job Count

The job count maximum for each interactive session initiator is controlled by `/machine/jobmax/` *initiator* `/jobcount` The default values for each interactive session initiator are determined during the automatic configuration process for URM. For example, the maximum job count for `login` sessions is set to the size of the kernel session table by default. Use the `rmgr` command to change this value, as in the following example:

```
/machine/jobmax/login/jobcount = 100
```

To permanently change the configuration, access the following menu and select the desired session initiator name (for example, `login`):

```
UNICOS 10.0 Installation / Configuration Menu System ->
  Configure system ->
    URM configuration ->
      URM control settings ->
        Individual session initiator targets
```

8.10.2.2.2 Interactive Memory and CPU Defaults

For interactive jobs, URM does not monitor session initiator values, as is done for the `batch` service initiator, because the interactive service initiators do not supply resource requirement information to URM. Instead, URM uses the default values for memory and CPU use in `/machine/default/` *initiator* `/memory` and `/machine/default/` *initiator* `/cputime`. For example, to determine if a login session will exceed the memory oversubscription target, URM compares the memory default for the job (in `/machine/default/login/memory`) to the current system memory load.

By default, these values are set to 0; this specifies that URM should recommend all interactive jobs for initiation. To enable URM control of the number of active interactive sessions for a specific initiator, both the `memory` and `cputime` defaults must be set to a nonzero value. For example, use the following `rmgr` directives to enable URM control of login sessions:

```
/machine/default/login/memory = 200
/machine/default/login/cputime = 10
```

The value 200 specifies the average size of memory, in clicks, allowed for login sessions. The value 10 specifies the average amount of CPU time in seconds. URM does not consider the `cputime` value when recommending interactive sessions, but both `memory` and `cputime` must be set to a nonzero value to enable URM to control jobs from a session initiator.

> **Note:** The session initiator defaults are used only for URM job initiation recommendations. They do not affect the actual memory and CPU limits of the executing session.

Automatic Incident Reporting Tests [A]

This appendix contains descriptions of the tests available with the automatic incident reporting (AIR) feature.

> **Note:** The AIR feature is not part of a Cray ML-Safe configuration of the UNICOS system.

The following tests are included:

Test	Description
`msgd.exist`	Message daemon existence test
`msgd.response`	Message daemon response test
`nqs.exist`	NQS daemon existence test
`nqs.funct`	NQS functional test
`nqs.response`	NQS daemon response test
`nqsnet.exist`	NQS networking existence test
`tape.exist`	UNICOS tape subsystem existence test
`tape.func`	UNICOS tape subsystem functional test
`tape.response`	UNICOS tape subsystem response test
`tapeavr.exist`	UNICOS tape subsystem AVR existence test
`tcp.exist`	TCP/IP existence test
`tcp.funct`	TCP/IP functional test
`tcpgated.exist`	TCP/IP `gated` existence test
`tcplpd.exist`	TCP/IP `lpd` existence test
`tcpnamed.exist`	TCP/IP `named` existence test
`tcpntpd.exist`	TCP/IP `ntpd` existence test
`tcpsmail.exist`	TCP/IP `sendmail` existence test
`tcpsnmpd.exist`	TCP/IP `snmpd` existence test

`urm.exist`	URM daemon existence test
`urm.funct`	URM functional test
`urm.response`	URM daemon response test

Index