Operator Workstation (OWS) Guide

SN–3030

Cray Research, Inc.

Requests for copies of Cray Research, Inc. publications should be sent to the following address:

Cray Research, Inc.
Distribution Center
2360 Pilot Knob Road
Mendota Heights, MN 55120

RECORD OF REVISION

CRAY RESEARCH, INC.

PUBLICATION NUMBER SN-3030

Each time this manual is revised and reprinted, all changes issued against the previous version are incorporated into the new version and the new version is assigned an alphabetic level.

Every page changed by a reprint with revision has the revision level in the lower righthand corner. Changes to part of a page are noted by a change bar in the margin directly opposite the change. A change bar in the margin opposite the page number indicates that the entire page is new. If the manual is rewritten, the revision level changes but the manual does not contain change bars.

Requests for copies of Cray Research, Inc. publications should be directed to the Distribution Center and comments about these publications should be directed to

CRAY RESEARCH, INC.
1345 Northland Drive
Mendota Heights, Minnesota 55120

Revision	Description
	October 1988 - Original printing supporting the OWS 1.0 release.

PREFACE

This manual describes the Cray Research, Inc. (CRI) Operator Workstation (OWS). The OWS is a VME-based microcomputer system used for CRAY Y-MP and CRAY X-MP EA computer system operation and monitoring and is intended as a replacement for the I/O Subsystem (IOS) Peripheral Expander. On CRAY-2 computer systems, the OWS has the priviledges of any UNIX station.

Section 1 provides an overview of the OWS and describes the differences between the OWS and the Peripheral Expander. Section 2 describes the OWS commands. Appendix A provides information on booting the system. Appendix B provides the CRI OWS man pages. Appendixes C and D describe the FEI-3 installation and CRAY-2 usage of the OWS respectively. Appendix E describes the deadstart diagnostics **dsdiag** and **cleario**.

Information on the OWS Station Interface is contained in the Operator Workstation (OWS) Station Interface Reference Manual, CRI publication SN-3031. IOS information pertaining to the OWS is contained in the IOS Software Addendum - Operator Workstation (OWS) Technical Note, CRI publication SN-0313. Detailed information on the IOS software is contained in the IOS Software Internal Reference Manual, CRI publication SM-0046, and the IOS Table Descriptions Internal Reference Manual, CRI publication SM-0007.

CONVENTIONS

The following conventions are used throughout this manual unless indicated otherwise:

Convention	Description
bold	Lowercase bold identifies UNIX command verbs or literal parameters.
Italic	Italics defines generic terms that represent words or symbols to be supplied by the user.
[]	Brackets enclose optional portions of command lines.
A|B	A vertical bar indicates that one item (A) or the other (B) should be chosen.

READER COMMENTS

If you have any comments about the technical accuracy, content, or organization of this manual, please tell us. You have several options that you can use to notify us:

- Call our Technical Publications department directly at (612) 681-5729 during normal business hours (Central Time)
- Send us UNICOS electronic mail at:

  bungia!cray!publications or **sun!tundra!hall!publications**

- Send a facsimile of your comments to the attention of "Publications" at FAX number: 612-681-5602
- Use the Reader Comment form at the back of this manual
- Write to us at the following address:

  Cray Research, Inc.
  Technical Publications Department
  1345 Northland Drive
  Mendota Heights, Minnesota 55120

We value your comments and will respond to them promptly.

CONTENTS

FIGURES

TABLES

1. INTRODUCTION

The Cray Research, Inc. (CRI) Operator Workstation (OWS) is a VME-based microcomputer running the UNIX operating system. The OWS provides for the following functions:

- System operator interface
- System deadstart
- System monitoring functions
- Software maintenance utilities
- Remote access for software service
- System time-of-day clock
- Local tape and printer
- Optional local removable streamer tape

The OWS replaces the Peripheral Expander and system consoles used on CRAY-1 and CRAY X-MP computer systems. The OWS communicates with the Cray mainframe through a low-speed channel from the I/O Subsystem's (IOS) Master I/O Processor (MIOP), as shown in figure 1-1. The OWS peripheral devices are available to the mainframe through the MIOP.

This section provides information on hardware and software configuration, an OWS overview, the differences between the OWS and the Peripheral Expander, and related documentation.

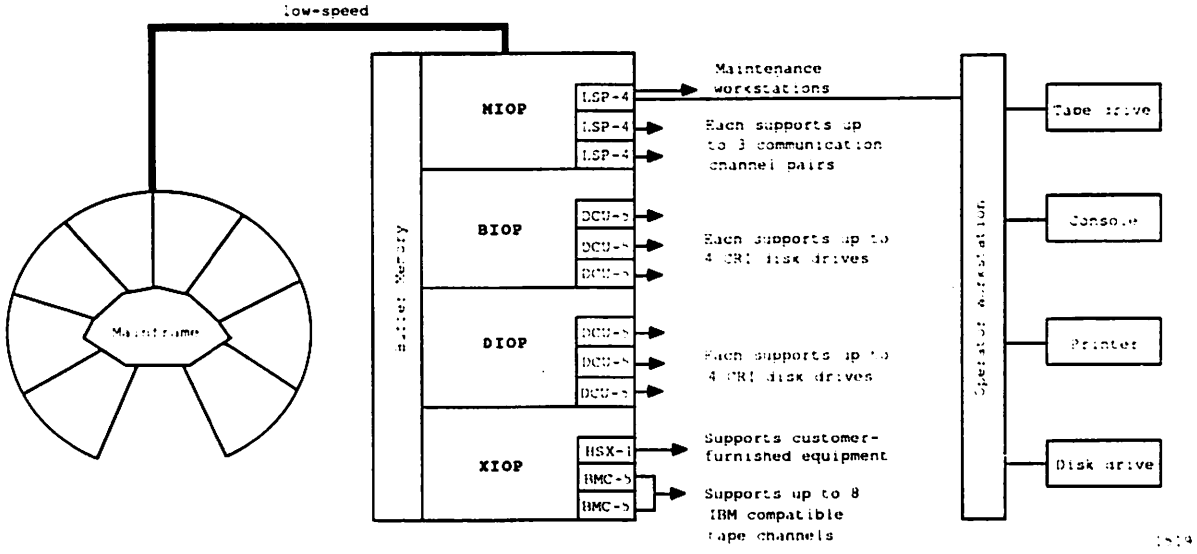

Figure 1-1. OWS Connection to IOS

1.1 HARDWARE CONFIGURATION

The OWS hardware (figure 1-2) consists of the following equipment:

- 2016 microcomputer with 8 Mbytes of memory:
  - Two 182-Mbyte disk drives
  - 8 port multiplexer for terminal support (one for each IOS)
  - One 60-Mbyte streamer tape
  - Battery backed-up time-of-day clock
- Tri-density Telex Tape Drive
- Ethernet Controller: Used to connect OWS to the local Ethernets
- 19" Color Graphics Monitor: A bit mapped, color graphics terminal used as an external terminal. Comes with an AT compatible keyboard and mouse.
- AMPEX Video Display Terminal: Used as a serial system console for the microcomputer
- Hewlett-Packard LaserJet II Printer: Used as workstation printer.
- CRI Interface Boards: Used to connect the low-speed channel to the microcomputer's VMEbus.

Detailed information on these peripherals can be found in the documentation supplied with the OWS.

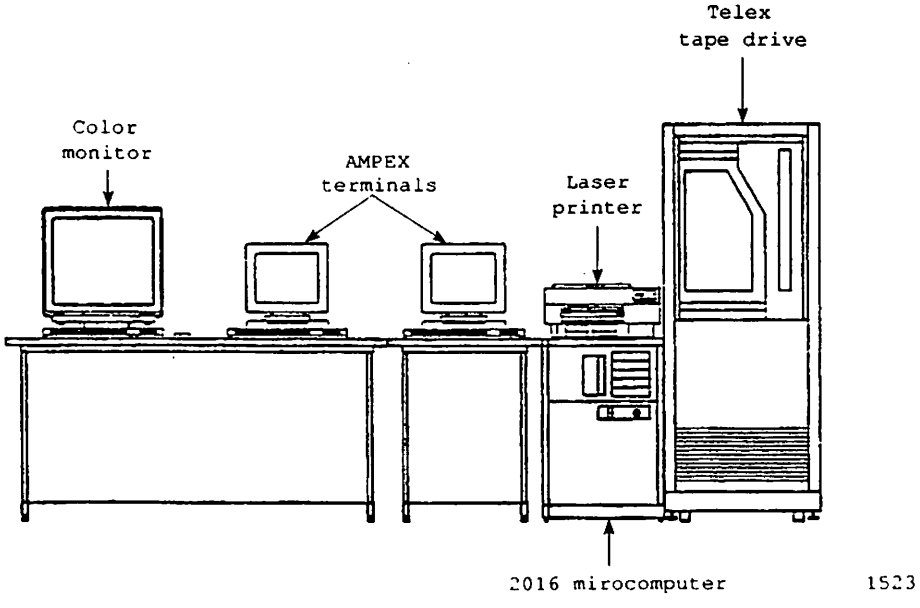

Figure 1-2. Operator Workstation Hardware

1.2 SOFTWARE CONFIGURATION

The software of the OWS (figure 1-3) consists of the following components:

- Microcomputer software
- Protocols and drivers
- CRI OWS software
- IOS software

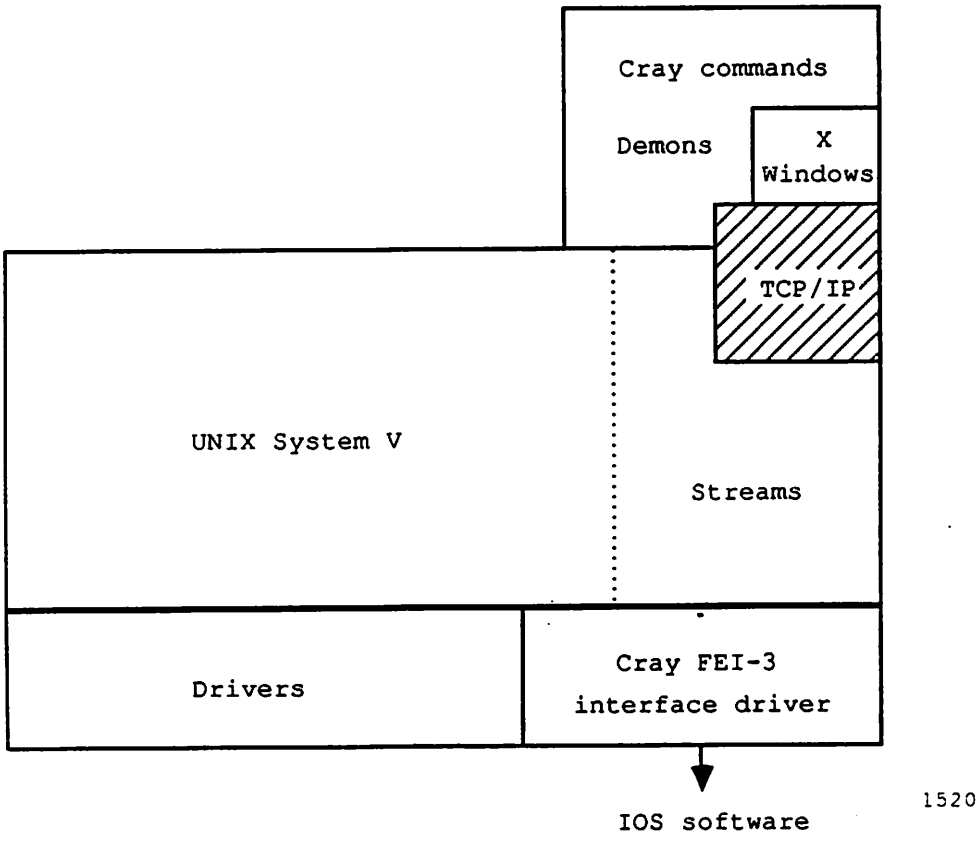

Figure 1-3. Software Components

1.2.1 MICROCOMPUTER SOFTWARE

The software of the microcomputer is UNIX System V with drivers for all peripherals except the FEI-3 VMEbus interface. The software allows for administration support, software installation, and operation of the CRI mainframe. Many functions of the Peripheral Expander commands have been replaced by these UNIX commands.

1.2.2 PROTOCOLS AND DRIVERS

The CRI FEI-3 Interface provides a hardware connection, through the backplane of the OWS, between the VMEbus and the Cray low-speed channel. The **scy** driver provides the software interface to this hardware. **scy** is a raw (headerless) mode, single-user driver for the FEI-3M hardware. Ordinarily users access **scy** through the multiplexor module, **cymux**. For more information on **scy** and **cymux**, refer to subsection C.2, Test Man Pages.

1.2.3 OWS SOFTWARE

The software of CRI's OWS provides additional functionality that the microcomputer software does not. Commands are available through the OWS itself, and the OWS's Station Interface.

Specifically, the OWS software provides for boot starting and dumping the Cray system. A number of commands are provided which allow the OWS to issue functions to the IOS for command and control.

1.2.4 IOS SOFTWARE

The software of the IOS provides the link between the OWS and the Cray mainframe. The IOS handles and routes all requests from the OWS.

The standard IOS FEI-3 driver handles the SCP, USCP, and TCP/IP protocols on a "logical-path" basis to enable communication between the OWS and the mainframe. In addition, a new V-packet protocol is handled by the IOS VMEOPR overlay to support communication between the OWS and the IOS, which needs no mainframe intervention. Examples of tasks supported by the V-packet are deadstarting and deaddumping.

1.3 OVERVIEW

By moving the operator environment from the IOS Peripheral Expander to the OWS, a more diverse and user-friendly operator environment is provided, as well as freeing up substantial amounts of IOS resources which can be better used for Cray I/O.

The OWS is only available for connection to the MIOP of IOS Model C or later. Through the IOS, OWS supports the CRAY X-MP, CRAY X-MP/se, CRAY X-MP EA, and CRAY Y-MP mainframes. The microcomputer is a 32-bit machine with 8-Mbytes of memory and a VMEbus backplane. It is into this bus that the FEI-3 connects to the low-speed (6 or 12 MB/sec) channel from the MIOP. The microcomputer allows connections to an Ethernet network and to a modem, and runs a UNIX System V operating system, with CRI written driver, IOS support, and optional station code.

The peripherals of the OWS (laser printer, disk, tape drives, and time-of-day clock) are accessible to the mainframe through the low-speed connection to MIOP. These peripherals replace the peripherals that once were on the Expander channel (also connected to MIOP). The Expander channel does not exist in the OWS environment.

The OWS has one color graphics display that provides the user interface to all Cray IOPs and CPUs. (This display, with the microcomputer's networking and X Windows graphics software, provides a windowing system through which UNICOS is able to communicate.) In addition, the multiple IOP consoles that were used for communicating to the IOPs and CPUs are replaced by this one graphics display at the workstation.

1.3.1 IMPACT

The main goal of the OWS is to remove the handling of operator requests from the CPU/IOS software as much as possible. Some of the benefits of this action have been discussed previously. The first difference that a Cray operator will notice is that instead of rows of IOP consoles, there is one large display screen, capable of supporting multiple windows, as opposed to the simple line-at-a-time displays seen in expander systems. There is the capability of many windows being open on this screen, each displaying a different type of information about the Cray system.

NOTE

The first few prereleases of the OWS systems will still have AMPEX terminals as the OWS console.)

The commands that the operators now enter are different from the current command syntax used on the expander systems. The basic login and user commands syntax is pure UNIX System V. In addition, there are OWS and OWS Station Interface commands described in section 2 of this manual.

TCP/IP and X Windows are the preferred vehicles for communicating with UNICOS. They allow true graphics representation of system information. For COS systems, or systems without TCP/IP, the OWS SCP Station Interface is provided.

The IOS station ("AP" batch station ID and "II" interactive station ID) no longer exist. This is not much different from today's UNICOS environment, where the IOS station exists, but cannot communicate with the Cray mainframe. For COS systems and UNICOS systems without TCP/IP, the IOS station has been replaced by the OWS SCP Station Interface in the microcomputer. This station is based on the standard UNIX station and enhanced with the functionality required of the master operator.†

Also unused in the OWS configuration are the Kernel commands that were used to control front-end communication device connections, expander device functions, editing, maintenance and debugging capabilities. New commands with similar capabilities have been implemented in the OWS for all of these areas except the expander device handling and editing. Device handling and editing are part of the normal UNIX environment on the microcomputer and therefore can be accessed with standard UNIX device structures and commands to do things such as reading, writing, editing, and copying files on these devices (see section 2).

Because the Cray system master operator functionality lies in a machine that may be connected to an Ethernet, the possibility of a remote Cray system master operator has become a reality. This will not only aid a site in allowing the location of the operator's environment to be detached from that of the Cray hardware, it will allow for more complete hardware and software support over long distances (that is, from a regional or Mendota Heights office, to a site).

A parallel effort to the VME-based Operator Workstation is a VME-based Maintenance Workstation (MWS) to replace the current Maintenance Control Unit (MCU). The MWS is also being developed in a microcomputer which may be connected to the MIOP. Copies of both the Maintenance and Operator software systems are kept in each of these workstations. In this way, if

† The master operator station has the ability to status and manipulate any job in the system, as well as the entire system itself. A non-master operator station can only alter jobs which belong to its station ID. By default, the IOS station is declared as the master station in COS systems, although a site may change the default to a different machine's station. In a UNICOS system, the master operator functionality belongs to anyone with "super user" permissions.

one workstation should experience a hardware failure, the other workstation may then be used to carry on the functions of both the Operator and Maintenance workstations until the hardware is repaired.

For the foreseen future, the IOS debugger user interface and Kernel displays are being handled across RS232 lines, one from each IOP into serial ports on the workstation terminal.

1.3.2 ABSENCE OF THE IOS STATION

As noted earlier, the IOS master station under COS (both batch and interactive modes) is no longer available. It has been replaced with master station functionality in the OWS. For COS systems, the OWS Station Interface is available for obtaining status of, and manipulating Cray jobs and the system. This means that the COS operator needs to be familiar with the commands that are an extended set of those commands found in the UNIX station. These commands are described in section 2 of this manual, and in detail in the Operator Workstation (OWS) Station Interface Reference Manual, publication SN-3031.

In a COS environment, the new station code in the workstation communicates with the COS SCP task through the SCP protocol, just as the IOS Station and all other front-end stations have always done. No new code was written in COS to support the master operator station in the workstation instead of in the IOS.

For a UNICOS system, there are two protocols available for use in fulfilling the statusing and manipulating capabilities provided to the OWS by the COS SCP protocol. If the UNICOS system has the TCP/IP protocol already implemented and licensed, TCP/IP will be used by the OWS, to communicate with the UNICOS TCP in providing the master operator functions. TCP/IP is the preferred protocol to use with UNICOS because it fits naturally into the UNIX environment and makes bit-mapped graphic displays possible through the X Windows.

Because TCP/IP is not a standard feature of UNICOS (due to its cost and the fact that it must be licensed separately from the UNICOS system), it cannot be required of our customers who must run the OWS. Without TCP/IP, the OWS uses the station software to communicate with the UNICOS USCP daemon through the USCP protocol.

Whichever protocol is used, the standard IOS FEI-3 channel driver treats transfers as any other transfers of these protocol types from other VME-based front end stations (SCP or USCP) and from other users logged in interactively to UNICOS (TCP/IP). In other words, these requests will be received on a "logical path" that is defined as belonging to the chosen protocol, and therefore, the request packets will be sent on to the responsible CPU daemon without the IOS actually acting on the request.

1.3.3 V-PACKET PROTOCOL

The Kernel initialization code will begin to use a newly-defined protocol that is unique to transfers involving the portion of MIOP that specifically deals with the OWS. This protocol defines an 8-Cray-word packet, called the "V-packet." The first four words of the V-packet define such things as the destination and source addresses of the packet, the requested function of the packet, the response status, and information about data associated with this packet.

The first word of the V-packet is defined exactly as the first word of the Message Proper used in other NSC and VME communication). The last seven words of the packet are defined according to the specific function request types. Each transfer in this protocol will consist of a V-packet, optionally followed by a data block. In this subsection, this protocol will be referred to as the "V-packet protocol."

In the past, the IOS Station software made available to the operator a few commands and displays which allowed the monitoring of activities within the IOS and of the channels connected directly to the IOS. These commands are now handled completely within the IOS, and not by any of the previously mentioned protocols.

Because this type of information is valuable to an operator, especially in diagnosing problems in the Cray system, these functions are implemented using the same V-packet protocol used at deadstart and deaddump time (refer to the IOS Software Addendum - Operator Workstation (OWS), publication SN-0313). Operator requests for these functions are received by the VMEOPR overlay. The requested information is then gathered from various IOS internal tables, and a response V-packet (with optional data block(s)) is sent back to the OWS for formatting and display. The commands implemented in this way are as follows:

CONC **DKERR** **DSPL** **LPATH** **MONITOR**
STP

It is of interest to note that these few commands are the only "Station" commands that the current IOS Station supports on a UNICOS system. This is because the IOS Station does not support either the USCP or TCP/IP protocols. It is more logical to perform operator type functions through a direct UNICOS interactive login than through a station. Thus, a UNICOS system with a Peripheral Expander has all the AMPEX non-Kernel consoles initialized as "UNICOS interactive terminals" rather than as "IOS Station consoles." It is possible, with an expander-UNICOS environment, to switch a terminal from UNICOS interactive mode to station mode when one of the above commands is desired. (See the I/O Subsystem (IOS) Operator's Guide for UNICOS, publication SG-2005, Revision A, for details).

The Operator Workstation takes care of display matters such as refresh rates, clearing the screen, etc. IOS station commands (for these types of functions) that have been assumed by the UNIX system running in the microcomputer are as follows:

@filename@	**CLEAR**	**DEFAULT**	**DELAY**	**HELP**
IACON	**PAUSE**	**STATINIT**		

The **IACON** command is included in this list because the IOS is no longer directly involved in initiating interactive sessions to the Cray system.

1.3.4 REPLACEMENT OF IOS KERNEL COMMANDS AND DISPLAY INTERFACE

This subsection describes other functions of the OWS that replace IOS Kernel commands (commands entered at an IOP's Kernel console). The OWS commands that fall into this category are as follows:

boot†	**configure**	**console**	**dboot**†	**ddmp**†
dkdmp†	**endconc**	**halt**	**iosconf**†	**iosdisk**†
iospeek†	**iosping**†	**iospoke**†	**listo**	**listp**
man†	**mfclr**	**mfinit**	**mflosp**	**mfpeek**†
mfpoke†	**mfsystem**†	**nsc**	**nscend**	**sentry**†
start	**sysdmp**	**trdmp**†	**trevt**†	**vme**
vmend				

All of these commands are implemented through the same V-packet protocol referenced earlier for deadstarting, deaddumping, and supporting the station commands of above.

As described earlier, the V-packet protocol involves sending an 8-word request packet from the OWS to the MIOP. The VMEOPR overlay in the MIOP receives the request and calls on the necessary parts of the IOS to fulfill the request. VMEOPR then sends a response packet and optional data block(s) back to the OWS. This can continue with the OWS sending a packet and potential data, and the IOS responding with a packet and data (normally data flows in one direction only). When a "Cray system halt" V-packet is received by the OWS on its "critical path," a operator message is displayed. The OWS sysdmp program may then be executed by a operator entering the OWS **sysdmp** command. This is equivalent to entering the CONTROL-D sequence on the MIOP Kernel console in an expander based system.

---

† OWS commands replacing Kernel commands functionality, but not corresponding to actual Kernel commands

The IOS Kernel commands that are not supported because they no longer require IOS or CPU action are as follows:

ABORT	**CLEAR**	**COPY**	**DDUMP**	**DEF**
DELETE	**DISABLE**	**DLOAD**	**DSTAT**	**EDIT**
ENABLE	**ERRDMP**	**ERROR**	**FDUMP**	**FLAW**
FLOAD	**FORMAT**	**FSCOPY**	**FSTAT**	**HELP**
HPLOAD	**IAIOP**	**INIT**	**PART**	**PLOT**
PROC	**PRTAPE**	**RENAME**	**RESTART**	**RESUME**
UBTAPE	**XDK**	**XMT**	**XPR**	

These commands deal with the obsoleted expander devices (clock, tape, disk, printer, plotter), the IOS expander disk file system, and the IOS editor. Since the functionality of the expander devices is replaced with the peripherals of the OWS, the commands to interface with these peripherals are supported with standard UNIX system commands such as **cp**, **cd**, and **vi**, and are described in the Motorola SYSTEM V/68 Release 3 User's Reference Manual, publication MU43810UR/D2. Also included in this list is the IAIOP command, which is associated with the IOS interactive ("II") station.

The full Operator Workstation implementation is being conducted in stages. Currently, there will still be RS232 lines from each IOP leading directly into serial ports on the OWS. Kernel commands and console messages being sent to and from IOPs other than MIOP will go through these RS232 lines until all of the software is in place to remove the RS232 support, and multiplex traffic to all IOPs through the single FEI-3 connection on MIOP. This means that some of these commands will remain in the IOP Kernel software and not be moved out to the OWS until later releases of the OWS.

1.3.5 DEADSTARTING THE MAINFRAME

The first step in deadstarting the mainframe is to have the OWS software up and running. For more information on deadstarting, refer to Appendix A, Booting the System. Before booting the IOS, the IOS deadstart diagnostics may be run (for more information on the deadstart diagnostics, refer to appendix E, IOS Deadstart Diagnostics). Next, the operators boot the IOS. In the past, operators were accustomed to pushing the Master Clear and Deadstart buttons on the IOS chassis to send the control signals that start the booting of the system. These chassis buttons are no longer used in this process. A similar set of control lines run from the MIOP directly into the OWS. Now the operators simply enter the **boot** command from the OWS. When the workstation processes this command, it internally sends these deadstart control signals to the MIOP. See section 2 for a description of the **boot** command syntax.

The control signals cause the low-speed interface to perform a read on the channel associated with the control lines. The control signals are accepted by the IOS if the switch for the channel is enabled. The boot command causes the OWS to send a standalone IOS program, the Loader, across the low-speed channel into MIOP Local Memory location zero. The first portion of this Loader file contains several parameters that the workstation sets for the IOS's use. These parameters are: the word size of the IOS Kernel binary; the sector size of the IOS overlay binary (all overlays concatenated into one file); the low-speed input channel number that connects the MIOP to the OWS; a flag indicating if IOS debug mode is to be entered; and the maximum transfer size, in sectors, that the OWS can handle. This last parameter is needed because the microcomputer system software currently is limited to transfers of less than 8 sectors. For transfer of files greater than the limit, the OWS and MIOP must break the file into multiple transfers of the limiting block size.

The low-speed input channel number parameter is needed for the IOS to build internal tables and so that the channel connecting the MIOP to the OWS can change without having to rebuild the IOS.

The Loader will first read the overlay file from the OWS by doing several inputs of the above noted maximum block size. As each block is read in, it is written to the overlay area of Buffer Memory. Next the IOS Kernel is read from the OWS into MIOP Local Memory, starting at location zero. The Loader then forces the MIOP to start executing at location zero. Up to this point, all transfers across the low-speed channel to and from the OWS have been in "raw" mode; that is, there has been no protocol involved, just portions of raw data going across the channel.

Once it has begun to execute at location zero, the MIOP kernel sends one of these packets to the OWS to indicate that the MIOP has begun initialization and is waiting for more information from the OWS. When the OWS receives this packet, it sends another V-packet back to the MIOP that contains the current time and date (from the microcomputer system clock) and the *critical path* number. The critical path number is basically a logical address in the OWS that the IOS should use as the destination for any unsolicited packets that it sends which require immediate attention from the OWS. These types of packets include those that indicate an IOP or CPU halt, and IOS Kernel-type console messages. The messages received at this address are formatted for display at the operator's console and are also written to a Cray log file kept on the workstation. The time and date information received by MIOP in the V-packet, replaces the IOS call to the expander clock for time and date.

After receiving this information, the IOS finishes deadstarting itself. At this time the standard FEI-3 channel driver is created in MIOP as well as the layer of code that sits on top of this driver and handles further V-packet protocol exchanges. The heart of this V-packet handling code is the VMEOPR overlay. (For more information on the IOS VMEbus overlays, refer to the VMEbus Driver Technical Note, CRI publication SN-0244.)

The standard FEI-3 channel driver, centered around the VMERD and VMEWT overlays, is used by MIOP to handle all connections to VME-based machines, whether that machine is the Operator Workstation or just another front-end machine. This driver uses a means of routing transfers of different user protocols in the Cray mainframe (i.e. SCP, USCP, TCP/IP, NETEX, etc.) by assigning different "logical paths" or different logical addresses to each protocol. The VMEOPR activity and its V-packet protocol is also assigned a unique logical path and therefore can be treated as "just another protocol" by this low-level channel driver.

When VMEOPR is created, it sends a V-packet, via the standard FEI-3 channel driver, to the OWS informing it that the IOS is completely operational and is waiting for further instructions. At this point the start command may be entered at the OWS causing a "start CPU" request V-packet to be sent to the MIOP. Included in this packet is the sector length of the CPU binary file. This file consists of the concatenation of the CPU system binary and the CPU parameter files. The OWS start command is the equivalent of the IOS Kernel START command, which was entered on expander systems at the MIOP Kernel console to start the mainframe. See section 2 for a description of start. When VMEOPR receives this request, it optionally (based on a flag included in the "start CPU" V-packet) executes the mainframe verification code (MFINIT) and reports the results to the OWS through another V-packet transfer. (In the OWS, MFINIT only reports errors.) On the current expander systems, MFINIT reports the results to the MIOP Kernel console with:

```
MFINIT:  COMPLETE
```

The next step is for the IOS to send another packet to the OWS indicating that it is ready to receive the CPU binary file. (Recall that the IOS is given the sector length of this file in the "start CPU" request packet.) The OWS then starts a sequence of sending a V-packet followed by a block of binary data, and waiting for a response V-packet from the MIOP. These exchanges take place until the entire CPU binary has been transferred from the OWS. As the MIOP receives each block of the file, it transfers that data to the mainframe (CPU) over the IOS<->CPU High-speed channel before sending the response packet back to the OWS. After the IOS has transferred the last block of binary data to the mainframe, it waits to send the final response packet to the OWS until it has sent the time/date stamp to the mainframe (as a separate transfer), and has verified that the mainframe is deadstarted and able to communicate with the IOS. It is at this point, in an expander-based operator environment, that the following messages appear on the MIOP Kernel console:

```
CPU <-> MIOP CHANNEL INIT
CPU <-> MIOP LINKAGE COMPLETE
```

In the OWS environment, this last response packet will generate a prompt at the OWS console, indicating that the system startup is complete.

1.3.6 DEAD DUMPING THE MAINFRAME

A deaddump, or "rawdump," may be taken when the IOS is so incapacitated that it cannot execute the standard **sysdmp** request. The deaddump is initiated by the OWS **ddmp** command (refer to section 2 for a detailed description). This command has an effect similar to pressing IOP-0 Master Clear and Deadstart buttons on the IOS, and then responding to the prompts at the console that executes the dump program residing on the expander tape or expander disk. When the **ddmp** command is entered at the OWS, the workstation issues an IOS master clear and a CPU master clear to the MIOP, which causes it to begin reading a file from the OWS into MIOP Local Memory location zero. This file is the same IOS Loader file that is read in to initiate deadstart processing.

As in the case of deadstarting, there are a few parameters at the start of the deaddump Loader file: the word size of the IOS Dead Dump program; a debug flag; and the input channel number of the low-speed channel that connects the MIOP to the OWS. The Loader will save some volatile MIOP information and then read the IOS Dead Dump program into MIOP Local Memory location zero, and begin executing it.

The idle state of the Dead Dump program is when the MIOP is waiting to input a V-packet from the OWS. The first packet the MIOP will receive is a packet requesting initialization of the deaddump. This causes the MIOP to gather all volatile information from every configured IOP and in addition, 40 sectors of Buffer Memory, and write it to the OWS as a response V-packet. After this initial exchange, all further exchanges are based on **ddmp** subcommands (see section 2) which the operator enters at the OWS. Each exchange consists of the OWS sending a request V-packet, then the MIOP sending a response V-packet and data block(s). Because formatting of the dump will take place in the OWS not the mainframe, a deaddump will be able to be viewed in a legible, formatted manner by an analyst.

1.4 DOCUMENTATION

There are several publications that support the OWS. These publications can be divided into three groups: Cray Research, Inc., Motorola, and peripheral hardware vendors. The Motorola and peripheral hardware vendor documentation should have been included with the hardware shipment. The CRI (OWS) documentation is provided with the OWS software.

1.4.1 OWS MANUALS

In addition to this manual, there are two other CRI manuals that provide information on the OWS software. The CRI manuals are as follows:

- Operator Workstation (OWS) Guide (SN-3030): This manual is intended for operators and system administrators and provides a description of the OWS, an overview of the system, commands, installation instructions, and OWS and FEI-3 man pages.

- Operator Workstation (OWS) Station Interface Reference Manual (SN-3031): This manual describes the commands that COS sites and UNICOS sites without TCP/IP have available to them in using the OWS.

- IOS Software Addendum - Operator Workstation (OWS) (SN-0313): This addendum provides IOS information as it relates to the OWS.

In addition to these manuals, the VMEbus Driver Technical Note, CRI publication SN-0244, may also be helpful.

1.4.2 MOTOROLA MANUALS

The Motorola manuals describe the software used by the Motorola microcomputer, and they come as a set shipped with the Motorola hardware. There are eight manuals in this set:

- SYSTEM V/68 Release 3 User's Reference Manual (Part Number MU43810UR/D2): This manual describes the commands that constitute the basic software running on the Motorola microcomputer.

- SYSTEM V/68 Release 3 User's Guide (Part Number MU43811UG/D2): This manual describes the basic principles of Motorola's SYSTEM V/68 operating system and provides a set of tutorials on the main tools provided on this system: the **ed** and **vi** text editors, the **shell** command language and programming language, and the electronic communication tools.

- System Administrator's Reference Manual (Part Number MU43812SAR/D2): This manual describes system maintenance commands and application programs, special files, and system maintenance procedures.

- System Administrator's Guide (Part Number MU43813SAG/D2): This manual describes procedures used in the administration of a Motorola microcomputer running the SYSTEM V/68 Release 3 operating system.

- Programmer's Reference Manual (Part Number MU43814PR/D2): This manual describes the commands, function calls, file formats and miscellaneous facilities of interest primarily to programmers on the Motorola microcomputer.

- SYSTEM V/68 Release 3 STREAMS Programmer's Guide (Part Number MU43817SPG/D2): This manual provides information about the use of the STREAMS mechanism at user and kernel levels. STREAMS was incorporated in SYSTEM V/68 Release 3 to augment the existing character input/output mechanism and to support development of communication services.

- X Window System Programmer's Reference Manual (Part Number 68NW9209F46A): This manual describes the C Language programming interface to the X Window System, the X library (Xlib).

- X Window System Programmer's Guide (Part Number 68NW9209F47A): This manual provides a conceptual introduction to the Xlib functions and includes programming examples and tutorial material. It is intended to be an introduction to all the basic concepts of X programming and also to be a useful reference for many of the most common programming techniques.

2. COMMANDS

This section provides information on the commands used with the Operator Workstation (OWS). Detailed information is provided on the OWS commands, and those commands using the RS232 lines. Commands are listed alphabetically in each subsection. Detailed information on the OWS Station Interface commands and Motorola commands are documented in the Operator Workstation (OWS) Station Interface Reference Manual, CRI publication SN-3031; and SYSTEM V/68 Release 3 User's Reference Manual, Motorola part number MU43810UR/D2; respectively.

All commands are entered in lowercase unless otherwise noted.

Table 2-1 provides a comparison of the commands (IOS Kernel and Station) used by the Peripheral Expander and their replacement on the OWS. The *italicized* descriptions in the "Expander Function" column represent the OWS command function if different from the Peripheral Expander command.

The Operator Workstation Station Interface Reference Manual describes additional commands not presented in table 2-1.

The references used in the table are as follows:

M	UNIX command implemented through the Motorola microcomputer
O	Command implemented through the Operator Workstation (OWS)
U	Command implemented through the OWS Station Interface
UC	Command implemented through the OWS Station Interface for COS systems only
†	Command uses RS232 lines of the OWS and should be entered uppercase

---

NOTE

The Operator Workstation Station Interface commands will not be supported until the next release of the OWS.

Table 2-1. Peripheral Expander/OWS Command Comparison

Expander	OWS	Expander Function
@filename@	Use shell scripts	Invokes the peripheral disk drive command file
ABORT	Ctrl C or D at executing terminal. kill[M] from another terminal	Terminates input or output on peripheral devices
CHANNEL	channel[UC]	Turns Cray mainframe I/O channel on or off. Also turns IOP channel on or off.
CLASS	class[UC]	Turns job classes on or off
CLEAR	rmdir[M]	Deletes directory(s) on expander disk (Kernel command)
CLEAR	clear[O]	Clears status display area of screen (Station command). *Clears terminal.*
COMMENT	Not Available	Allows you to enter comments into the screen display area
CONC	CONC†	Initiates FEI concentrator (Kernel command)
CONC	CONC†	Monitors Concentrator Table information for a specified concentrator (Station command)
CONFIG	iosconf[O]	Displays configured IOS channels
CONFIGURE	configure[U]	Alters the status of device

M Motorola command
O OWS command
U OWS Station Interface command
UC OWS Station Interface COS only command
† Uses RS232 lines; enter uppercase

Table 2-1. Peripheral Expander/OWS Command Comparison (continued)

Expander	OWS	Expander Function
CONSOLE	cs[U]	Adds another MIOP console to the station software *Initiates a user console mode session interface to the station; also used to preface station commands entered from the shell.*
COPY	cat[M], cp[M], cpio[M], dd[M], tar[M]	Copies a file from expander tape to expander disk; or from expander disk to expander tape.
CRAY	mflosp[O]	Initializes IOS and Cray mainframe link
DATASET	dataset[U]	Displays dataset status
DDUMP	cpio[M], tar[M]	Dumps directories from expander disk to tape
DEF	cd[M]	Enables setting of defaults for expander disk file references
DEFAULT	lpstat[M] ls[M]	Sets or displays default device, volume, and directory attributes for devices used for station staging operations
DELAY	sh[M], sleep[M]	Suspends processing of a command for a specified time
DELETE	rm[M], rmdir[M]	Deletes file(s) or directory(s) on peripheral disk
DISABLE	Use standard UNIX conventions	Places expander device off-line
DISK	iosdisk[O]	Initiates the disk statistics display
DKERR	DKERR†	Displays detailed disk error status

M Motorola command
O OWS command
U OWS Station Interface command
† Uses RS232 lines; enter uppercase

Table 2-1. Peripheral Expander/OWS Command Comparison (continued)

Expander	OWS	Expander Function
DLOAD	cpio[M], tar[M]	Loads directories from expander tape to disk
DROP	drop[U]	Immediately ends processing of job at the Cray mainframe but does not delete output datasets
DSPL	DSPL†	Displays a CPU formatted screen
DSTAT	ls[M]	Displays expander disk directories information
EDIT	vi[M], ed[M]	Invokes IOS editor
ENABLE	lpsched[M]	Places expander device on-line
END	end[U], exit[U], quit[U]	Terminates console operation (and logs off if no other active consoles) *Terminates console interface session*
ENDCONC	ENDCONC†	Terminates FEI concentrator
ENTER	enter[UC]	Changes parameters associated with a job or queued dataset
ERRDMP	Not Available	Dumps error log buffer
ERROR	Not Available	Turns MIOP Error Log channel on or off (Kernel command)
ERROR	Refer to the error logger	Displays information from the error Log Table if errors are sensed (Station command)
FDUMP	cpio[M], tar[M]	Dumps files from expander disk to tape

M Motorola command
O OWS command
U OWS Station Interface command
UC OWS Station Interface COS only command
† Uses RS232 lines; enter uppercase
†† Refer to the Motorola System Administrator's Guide (MU43813SAG/D2)

Table 2-1. Peripheral Expander/OWS Command Comparison (continued)

Expander	OWS	Expander Function
FLAW	††[1]	Disables an expander disk cylinder until next INIT is done
FLOAD	cpio[M], tar[M]	Loads files from expander tape to disk
FLUSH	flush[UC]	Copies data from a volatile device to a permanent mass storage device
FORMAT	††[1]	Formats expander disk
FSTAT	ls[M]	Displays information about files within an expander disk directory
GETIME	date[M]	Displays date and time
HELP	usage[M] help[U] man[O]	Displays information for all Kernel, Station, or interactive commands
IACON	statinit -p inter[U] ias[U]	Initializes MIOP console for use as interactive station console
		Initiates an interactive station; initiates a user interactive station session.
IAIOP	ias[U]	Allows interactive concentrator communication
		Allows communication with interactive station concentrator
INIT	dinit[M], usage[M], ††[1]	Initializes expander disk for IOS use

M Motorola command
O OWS command
U OWS Station Interface command
UC OWS Station Interface COS only command
† Uses RS232 lines; enter uppercase
†† Refer to the Motorola System Administrator's Guide (MU43813SAG/D2)

Table 2-1. Peripheral Expander/OWS Command Comparison (continued)

Expander	OWS	Expander Function
JOB	Not Available	Displays job status message
JSTAT	jstat[U]	Displays extended status information for a job
KILL	kill[U]	Depending on the status of the job, either deletes the job input dataset from the input queue (if processing has not yet begun), terminates processing (if processing has begun), or deletes an output dataset from the output queue
LIMIT	limit[UC]	Sets the maximum number of jobs that COS can process at one time
LINK	link[U]	Displays the status of the link between the station and the mainframe *Displays OWS stations link statistics*
LISTO	listo[O]	Prints list of all overlays in the system
LISTP	listp[O]	Prints list of all currently defined IOS halt codes
LOGOFF	logoff[U]	Terminates communications between an IOS station and COS *Terminates communications between the OWS station and the Cray operating system. Also terminates user console session from which it was entered.*
LOGON	Done when OWS is started	Establishes communications between an IOS station and COS
LPATH	LPATH†	Displays NSC/FEI logical path

M Motorola command
O OWS command
U OWS Station Interface command
† Uses RS232 lines; enter uppercase

Table 2-1. Peripheral Expander/OWS Command Comparison (continued)

Expander	OWS	Expander Function
MESSAGE	message[U]	Enters message into the job log file, system log file, or both.
MFCLR	mfclr[O]	Writes a pattern to Central Memory
MFINIT	mfinit[O]	Performs mainframe confidence test
MONITOR	MONITOR†	Monitors specified aspects of COS or IOS
MSG	msg[U]	Sends unsolicited operator message to a job
NSC	NSC†	Initiates NSC concentrator
NSCEND	NSCEND†	Terminates NSC concentrator
OPERATOR	operator[UC]	Changes the ID of the master operator station
PAUSE	sh[m]	Interrupts command-file processing and reads commands from the console keyboard
PLOTIT	Not Available	Prints plot data from expander tape
POLL	Must be configured	Sets the rate at which control messages are exchanged with the Cray mainframe if no staging is in progress
PROC	sh[M]	Executes Kernel console command file from expander disk
PRTAPE	Not Available	Prints unblocked tape files
RECOVER	recover[UC]	Lifts suspension from jobs suspended by a SHUTDOWN or system interruption

M Motorola command
O OWS command
U OWS Station Interface command
UC OWS Station Interface COS only command
† Uses RS232 lines; enter uppercase

Table 2-1. Peripheral Expander/OWS Command Comparison (continued)

Expander	OWS	Expander Function
REFRESH	refresh[U]	Sets the interval between screen update requests
RELEASE	release[U]	Releases a dataset which has been put into the HOLDING state
RENAME	mv[M]	Renames an expander disk file
REPLY	reply[U]	Replies to a specific station message
RERUN	rerun[UC]	Immediately ends processing of job at the Cray mainframe and reruns the job if it can be rerun; if not rerunnable, command is rejected.
RESTART		Terminates input or output on expander devices; allows output staging to be postponed and reinitiated later.
RESTORE	restore[U]	Places data on preemptable generic resource, or reinstates device availability
RESUME	Not Available	Resumes input or output on expander device (Kernel command)
RESUME	resume[UC]	Reschedules for processing jobs that were suspended by the SUSPEND or SHUTDOWN commands (Station command)
ROUTE	route[U]	Reroutes datasets intended for disposition at one station to another station by changing the job source ID or dataset destination ID
RSTAT	rstat[U]	Displays current information on generic resource availability and use

M Motorola command
O OWS command
U OWS Station Interface command
UC OWS Station Interface COS only command
† Uses RS232 lines; enter uppercase

Table 2-1. Peripheral Expander/OWS Command Comparison (continued)

Expander	OWS	Expander Function
SAVE	save[U]	Queues a file for staging to Cray mass storage, where the file is made a permanent dataset
SCROLL	scroll[U]	Changes the entire display area to a command/response area *Changes the size of the command/response area; comparably adjusts the size of the display area.*
SET	Not Available	Modifies the default value associated with station's ID and TID parameters
SETIME	date[M]	Sets date and time
SHUTDOWN	shutdown[UC]	Idles down job activity as part of the system deactivation process
SNAP	*command*|lp	Copies the display screen image to a line printer
STAGE	Not Available	Halts or resumes initiation of dataset staging between Cray mass storage and the IOS station
START	start[O]	Deadstarts the Cray mainframe
STATCLASS	statclass[U]	Displays status of defined job classes for current job class structure
STATION	station[U] statinit[U]	Initializes the IOS station software (Kernel command) **station** *initializes the batch station software and a user console session.* **statinit** *initiates the station software; either batch or interactive.*

M Motorola command
O OWS command
U OWS Station Interface command
UC OWS Station Interface COS only command
† Uses RS232 lines; enter uppercase

Table 2-1. Peripheral Expander/OWS Command Comparison (continued)

Expander	OWS	Expander Function
STATION	Not available	Displays station console status (Station command)
STATUS	status[U]	Displays status of all jobs in job input queue, datasets in output staging queue, and all executing jobs known to COS
STMSG	stmsg[U]	Displays station messages
STORAGE	storage[UC]	Initiates mass storage status display
STP	STP†	Displays statistics on the COS System Task Processor (STP)
STREAM	stream[UC]	Changes link activity by changing the number of input and output streams defined for a link
STRSTAT	strstat[U]	Displays information for individual station streams
SUBMIT	submit[U]	Queues a file for staging to Cray mass storage, where the dataset is entered into the job input queue
SUSPEND	suspend[UC]	Suspends processing of one or more jobs
SWAP	swap[U]	Displays information on the availability of swap space for preemptable generic resources
SWEEP	sweep[U]	Removes data from preemptable generic resource, or turns off device availability
SWITCH	switch[UC]	Sets or clears a job sense switch

M Motorola command
O OWS command
U OWS Station Interface command
UC OWS Station Interface COS only command
† Uses RS232 lines; enter uppercase

Table 2-1. Peripheral Expander/OWS Command Comparison (continued)

Expander	OWS	Expander Function
SYSTEM		Identifies the Cray mainframe resident operating system
TAPE	tape[U]	Displays configuration information for each tape device
TJOB	tjob[U]	Displays status of all jobs using generic resources, including those on the input, output, and executing queues
TLINK	tlink[U]	Links any tape that is already premounted to a user's request for a scratch tape (Automatic Volume Recognition (AVR) systems only
TTY	TTY†	Assigns a UNICOS interactive terminal
UBTAPE	Not Available	Unblocks a single, specified, blocked file on expander tape and sends it to the expander printer
UNLOAD	unload[U]	Unloads a premounted tape from the tape drive and rearms the drive for another mount (AVR)
VME	VME†	Initiates a VME-type channel driver
VMEND	VMEND†	Terminates the VME-type channel driver

M Motorola command
O OWS command
U OWS Station Interface command
UC OWS Station Interface COS only command
† Uses RS232 lines; enter uppercase

2.1 OWS COMMANDS

This section describes, in alphabetical order, the commands developed by Cray Research, Inc. (CRI) in support of the OWS. Additional on-line information on these commands is provided when the **man** command is used.

The OWS commands can be divided into the following categories:

- Initialization
- Maintenance
- Status
- Termination
- Utility

Table 2-2 summarizes the OWS commands.

Table 2-2. OWS Commands

Type	Command	Description
Initialization	boot	Boots the IOS from the Operator Workstation
	dboot	Boots the deadstart diagnostics into the IOS from the OWS
	mfclr	Requests the IOS to clear mainframe memory
	mfinit	Requests the IOS to initialize the mainframe
	mflosp	Requests the IOS to initialize the mainframe Lowspeed channel
	mfsystem	
	start	Starts the mainframe
Maintenance	listo	Requests overlay names and sizes from the IOS
	listp	Requests punt codes (IOP halt codes) from the IOS
	sentry	Waits on specific logical path for messages from the IOS and mainframe
	trdmp	Requests IOS trace information from a specified IOP
	trevt	Turns on or off IOS tracing

Table 2-2. OWS Commands (continued)

Type	Command	Description
Status	iosconf	Requests configuration information from specified IOP
	iosdisk	Requests current IOP disk data from specified IOP
	mfmon	Displays Cray CPU utilization
Termination	halt	Idles the Cray mainframe
Utility	ddcp	Copies data to or from the ROOT file system on a Cray disk
	ddmp	Initiates an IOS dead dump
	dkdmp	Dumps specified number of sectors from an IOS configured disk device
	eft	Displays the Engineering Flaw Table for a Cray disk
	flaw	Initializes the UNICOS Flaw Table on a Cray disk
	iospeek	Requests a read of IOS memory
	iosping	Sends a packet to IOS and records the time for it to be echoed back
	iospoke	Writes a specified pattern to IOS memory
	man	Displays on-line information on OWS commands (Refer to subsection C.2, Test Man Pages.)
	mfpeek	Requests the IOS to read mainframe memory
	mfpoke	Requests the IOS to write a word into mainframe memory
	sysdmp	Requests a Cray system dump

boot - BOOT THE IOS FROM THE OPERATOR WORKSTATION

FUNCTION: Boots the IOS using a Kernel and overlays on the Operator Workstation. **boot** uses the VME/FEI-3 device interface to deadstart the IOS. The "deadstart enable switches" for the connected channel on the IOS must be enabled, and the channel should not be used for anything else at this time. The **boot** command does not check to see if there is a running IOS or mainframe. Ensure that the mainframe is shut down before booting the IOS.

FORMAT: **boot** [**-b** *vme.boot*] [**-c** *channel*] [**-d**|**-D**] [**-I** *ios*] [**-k** *kernel*] [**-P** *path*] [**-z** *vmedev*]

-b *vme.boot* Specifies the binary file (*vme.boot*) that loads the IOS Kernel and overlays. If **-b** is not specified, the default is **/usr/cray/os/ios/vme.boot**.

-c *channel* Specifies the IOS channel number (*channel*) in octal that is to be used to communicate with the Motorola microcomputer. The default channel number is configured in the header file **ows.h**.

-d Sets debug mode in the IOS

-D Sets debug mode in the **boot** program

-I *ios* Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.

-k *kernel* Specifies IOS path name (*kernel*). The overlays path name is the same as the *kernel* path name, but suffixed with ".ov". The default is set to **/usr/cray/os/ios/kernel**.

-P *path* Specifies the default path for all boot files; currently it is **/usr/cray/os/ios**. The default is not used when **-b** or **-k** are specified, as these options require full path names.

-z *vmedev* Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

dboot - BOOT THE DEADSTART DIAGNOSTICS INTO THE IOS FROM THE OPERATOR WORKSTATION

FUNCTION: Boots the IOS using a diagnostic Kernel and diagnostic overlays on the Operator Workstation. **dboot** uses the VME/FEI-3 device interface to deadstart the IOS. The "deadstart enable switches" for the connected channel on the IOS must be enabled, and the channel should not be used for anything else at this time. The **dboot** command does not check to see if there is a running IOS or mainframe. Ensure that the mainframe is shut down before booting the deadstart diagnostics. Unlike the **boot** command, **dboot** requires that the **-k** option be used to specify the diagnostic to run. For more information on the diagnostics themselves, refer to appendix E, IOS Deadstart Diagnostics.

FORMAT: **dboot** [**-b** *vme.boot*] [**-c** *channel*] [**-d**|**-D**] [**-I** *ios*] **-k** *diag* [**-P** *path*] [**-z** *vmedev*]

-b *vme.boot* Specifies the binary file (*vme.boot*) that loads the diagnostic Kernel and overlays. If **-b** is not specified, the default is **/usr/cray/os/ios/vme.boot**.

-c *channel* Specifies the IOS channel number (*channel*) in octal that is to be used to communicate with the Motorola microcomputer. The default channel number is configured in the header file **ows.h**.

-d Sets debug mode in the IOS

-D Sets debug mode in the **boot** program

-I *ios* Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.

-k *diag* Specifies the diagnostic path name (*diag*). **-k dsdiag** specifies that the IOS deadstart diagnostic control program is run. **-k cleario** specifies that the IOS deadstart utility is run. The default is set to **/usr/cray/os/ios/kernel**.

-P *path*	Specifies the default path for all boot files; currently it is **/usr/cray/os/ios**. The default is not used when **-b** or **-k** are specified, as these options require full path names.
-z *vmedev*	Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

ddcp - COPY DATA TO OR FROM THE ROOT FILE SYSTEM ON CRAY DISK

FUNCTION: Copies data to or from the **ROOT** file system on a Cray disk (see subsection A.2.2). For a flawed block in **ROOT**, **ddcp** reads or writes the corresponding block in the **SPARES** area.

FORMAT: **ddcp** [**-r**] [**-w**] [**-b block**] [**-n** *count*] [**-I** *ios*] **-d** *file*

-r	Copies data from the Cray disk to **stdout**
-w	Copies data from **stdin** to the Cray disk
-b	Block offset into the file system. Default is 0.
-c *count*	Maximum number of blocks to copy. If not specified, a read will copy the **ROOT** file system from the initial block offset to the end of the file system. A write will copy data until an input EOF is encountered or the **ROOT** file system has been written from the initial block offset to the end of the file system.
-I *ios*	IOS interface number for systems with more than one IOS. Default is 0.
-d *file*	The file containing the Cray disk description (see subsection A.2.2).

ddmp - INITIATE, FORMAT, AND DISPLAY AN IOS DEADDUMP

FUNCTION:

Initiates an IOS deaddump. Commands at the keyboard (standard in) provide for a formatted display of selected portions of the IOS. The default output is directed to the standard out, but may be redirected to a ‹*pathname*› for later use.

The commands are issued from a terminal in response to the "›" prompt, rather than redirected from a file. **ddmp** creates a temporary file of volatile data. The temporary file is not removed when the program terminates.

The commands can be issued in any order and will be accepted if enough characters are present to distinguish it from other commands. (For example: **q** is all that is needed to quit the program.) If a command is not accepted, the informative message "?" is issued. The commands are not case sensitive, nor are parameters that are character data, except for the **redir** command. The **redir** command uses the ‹*pathname*› parameter for the open, exactly as specified. The parameters ‹*addr*› ‹*length*› are octal numbers that specify the starting address and length of the dump. Addresses are rounded down to the next lower word address, and lengths are rounded up to the next word address when parcels are being dumped.

The **mode** is used to determine whether the address and length are in words or in parcels. The default mode is to use the natural size for the object: the natural size for an IOP is a parcel; the natural size for Buffer Memory or Central Memory (Cray memory) is a word. The default mode is used when in **automode**. The **mode** command is in force until another **mode** command, or the **automode** command, is issued.

The **trace** command requires only the ‹*iop*› parameter. If the ‹*entries*› parameter is not specified, the default is all the available trace for that *iop*. **ddmp** uses the VME/FEI-3 device interface to communicate with the IOS.

The current version does not save the first 600 parcels of IOP0.

FORMAT:

ddmp [-b *vme.boot*] [-c *channel*] [-D] [-I *ios*] [-d *vmedump*] [-P *path*] [-z *vmedev*]

FORMAT: (continued)

-b *vme.boot*	Specifies the binary file (*vme.boot*) that loads the IOS Kernel and overlays. The default is **/usr/cray/os/ios/vme.boot**.
-c *channel*	Specifies the IOS channel number (*channel*) in octal that is to be used to communicate with the Motorola microcomputer. The default channel number is configured in the header file **ows.h**.
-D	Sets debug mode in the **ddmp** program
-I *ios*	Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.
-d *vmedump*	Specifies IOS deaddump path name. The default is set to **/usr/cray/os/ios/vmedump**.
-P *path*	Specifies the default path for *vmeboot* and *vmedump* files; currently, it is **/usr/cray/os/ios**. The default is not used when **-b** or **-d** are specified, as these options require full path names.
-z *vmedev*	Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

Commands that can be used with **ddmp** areas follows (values in octal):

cpu *addr length*	Dumps CPU memory starting at address *addr* for a length of *length*.
bmem *addr length*	Dumps Buffer Memory starting at address *addr* for a length of *length*.
iop0 *addr length*	Dumps IOP0 memory starting at address *addr* for a length of *length*.
iop1 *addr length*	Dumps IOP1 memory starting at address *addr* for a length of *length*.

FORMAT:
(continued)

iop2 *addr length*	Dumps IOP2 memory starting at address *addr* for a length of *length*.
iop3 *addr length*	Dumps IOP3 memory starting at address *addr* for a length of *length*.
mode [parcel|word]	Sets the dump format to parcel or word mode.
automode [on|off]	Allows the natural size for the memory type to be used: parcel for IOPs, word for Central or Buffer Memory.
echo	Echoes command to output file(s).
noecho	Stop previous **echo** command.
print	Prints the dump at the system console (standard out).
noprint	Does not print the dump at the system console.
regs *iop*	Dumps the registers of IOP *iop*.
redir *pathname*	Redirects (prints copy) output to *pathname*.
noredir	Stops previous **redir** command.
trace [*iop*] *entries*	Dumps *entries* of *iop* trace. If *entries* is not specified, the default is all of the available traces for the specified IOP.
quit	Terminates **ddmp**.
?	Help function, prints the commands and explanatory text.

dkdmp - IOS DISK DUMP INTERFACE

FUNCTION: Requests a specified number of sectors (in octal) to be dumped from an IOS configured disk device. **dkdmp** formats the returned information, and writes it to standard out. **dkdmp** has an interactive interface activated by the -a parameter or by not specifying the -c and -l parameters.

FORMAT: **dkdmp** -c *chan* [-C *cyl*] [-D] [-h *head*] [-i *iop*] [-I *ios*] [-l *len*] [-s *sector*] [-u *unit*] [-z *vmedev*]

-c *chan*	The IOS channel connected to the disk device. There is no default.
-C *cyl*	The starting cylinder on the disk device to dump. The default is zero.
-D	Sets debug mode in the **dkdmp** program.
-h *head*	The starting head on the disk device to dump. The default is zero.
-i *iop*	The IOP connected to the disk device. Default is zero (MIOP).
-I *ios*	Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.
-l *len*	The length, in sectors, to dump. The default is zero.
-s *sector*	The starting sector to dump on the disk device. The default is zero.
-u *unit*	The unit number of the disk device to dump. Note that some disk types do not have a unit number and this parameter is not necessary for those devices. The default is zero.
-z *vmedev*	Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

eft - DISPLAYS THE ENGINEERING FLAW TABLE FOR A CRAY DISK

FUNCTION: Displays the Engineering Flaw Table (EFT) for a Cray disk. The location of the eft is determined by the disk type.

FORMAT: **eft** [**-b**] [**-I** *ios*] **-d** *file*

-b	Prints block numbers only (this output may be used as input to the **flaw** command).
-I *ios*	IOS interface number for systems with more than one IOS. Default is 0.
-d *file*	The file containing the Cray disk description (see subsection A.2.2).

flaw - INITIALIZES THE UNICOS FLAW TABLE FOR A CRAY DISK

FUNCTION: The **flaw** command initializes the UNICOS flaw table on a Cray disk; and adds, deletes, or lists flaws. At least one of the the options -i, -a, -r, or -l must be specified; the -a and -r options may not be specified at the same time.

FORAMT: **flaw** [-i] [-a] [-r] [-l] [-I *ios*]
-d *file* [*blocks*]

-i	Initializes the flaw table
-a	Adds flaws
-r	Removes flaws
-l	Lists flaws
-I *ios*	IOS interface number for systems with more than one IOS. Default is 0.
-d *file*	The file containing the Cray disk description (see subsection A.2.2).
blocks	List of blocks to add (-a) or remove (-r)

halt - HALTS THE IOS

FUNCTION: Loads and starts the vme boot loader in the IOS. The boot loader forces the IOS to loop, waiting on the VME channel. This command is useful only when the Cray system needs to be idled.

FORMAT: **halt** [**-b** *vme.boot*] [**-D**] [**-I** *ios*] [**-z** *vmedev*]

-b *vme.boot* Specifies the path name for the IOS boot loader. The default is **/usr/cray/os/ios**.

-D Sets debug mode in the **halt** program.

-I *ios* Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.

-z *vmedev* Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

iosconf - LIST IOS CONFIGURATION

DESCRIPTION Requests the IOS configuration from a specified IOP. It formats the returned information, and writes it to standard out.

FORMAT: **iosconf** [**-D**] [**-i** *iop*] [**-I** *ios*] [**-z** *vmedev*]

-D Sets debug mode in the **iosconf** program.

-i *iop* Specifies the target IOP to retrieve configuration information from.

-I *ios* Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.

-z *vmedev* Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

iosdisk - LISTS IOS DISK INFORMATION

FUNCTION: Requests the current IOS disk data from a specified IOP. It formats the returned information, and writes it to standard out.

FORMAT: **iosdisk** [**-D**] [**-i** *iop*] [**-I** *ios*] [**-z** *vmedev*]

-D	Sets debug mode in the **iosdisk** program.
-i *iop*	Specifies the target IOP to retrieve disk data from.
-I *ios*	Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.
-z *vmedev*	Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

iospeek - READ IOS MEMORY

FUNCTION: Requests the IOS to read IOS memory. The memory is formatted by the program and written to standard out.

FORMAT: **iospeek -a** *add,addl* [**-b**] [**-D**] [**-i** *iop*] [**-I** *ios*] [**-m** *mode*] [**-o** *overlay*] [**-z** *vmedev*]

-a *add,addl* Specifies the address range to read from IOS memory. *add* is the starting address, and *addl* is the ending address. IOS memory addresses are in parcels, and, if buffer memory is specified, addresses are in words.

-b Peeks into Buffer Memory. The default mode is changed to "word". This option excludes the use of the **-i** and **-o** options.

-D Sets debug mode in the **iospeek** program.

-i *iop* Specifies the IOP to peek into. This option excludes the use of the **-b** option.

-I *ios* Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.

-m *mode* Specifies the mode of the dump, either parcel or word. The default is word.

-o *OVERLAY* Specifies a specific IOS overlay to be used with the **-a** option. *OVERLAY* is an ASCII string that matches an overlay name of the IOS. Note that overlay names are in uppercase only.

-z *vmedev* Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

SEE ALSO: **iospoke**

iosping - TEST THE IOS

FUNCTION: Sends a packet to the IOS, and the IOS then echoes the packet back. **iosping** times the round trip and writes the information to the system console (standard out). (The output is modeled after the BSD command **ping**(1).)

FORMAT: **iosping** [**-D**] [**-I** *ios*] [**-n** *npkts*] [**-r** *rfreq*] [**-z** *vmedev*]

-D Sets debug mode in the **iosping** program.

-I *ios* Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.

-n *npkts* Specifies the number of packets to send to/echo back (ping) from the IOS. The default is a very large number. The program can be interrupted by a **control C** at anytime.

-r *rfreq* Specifies the report frequency for output from **iosping** in packets. **iosping** will write a line to standard out every *rfreq* packets.

-z *vmedev* Specifies the operator workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

iospoke - WRITE IOS MEMORY

FUNCTION: Requests that the IOS write a specified pattern to IOS memory. **iospoke** should be used with caution, since writing into the memory of a running system can have unpredictable results.

FORMAT: **iospoke -a** *add* **[-b] [-D] [-i** *iop***] [-I** *ios***]**
[-o *OVERLAY***] [-p** *pattern***] [-z** *vmedev***]**

-a *add* Specifies the starting address of IOS memory where write is to occur. IOS memory addresses are in parcels. (If Buffer Memory is specified (**-b**), the address should be in words.)

-b Write (poke) Buffer Memory with the pattern (**-p**). The default mode changes to word. This option excludes the use of the **-i** and **-o** options.

-D Sets debug mode in the **iospoke** program.

-i *iop* Specifies the IOP to write (poke) into. This option excludes the use of the **-b** option.

-I *ios* Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.

-o *OVERLAY* This option specifies a specific IOS overlay address range (**-a** option). *OVERLAY* is an ASCII string that matches an overlay name in the IOS. Note that overlay names are in uppercase only.

-p *pattern* *pattern* is a 22 octal digit pattern to write into memory. (For Local Memory, it's a 6-digit pattern.) There is a group of predefined patterns of all zero bits, all one bits, or alternating ones and zeros. These can be specified as "zero", "one", "oz", "zo", "01", "10". (The preset patterns can be specified with alphabetic specifications in lowercase or uppercase.)

FORMAT: (continued)	-z *vmedev*	Specifies the operator workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

SEE ALSO: **iospeek**

listo - LIST IOS OVERLAYS

FUNCTION: Requests the overlay names and sizes from the IOS. It formats the returned information, and writes it to standard out.

FORMAT: **listo** [**-D**] [**-I** *ios*] [**-z** *vmedev*]

-D Sets debug mode in the **listo** program.

-I *ios* Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.

-z *vmedev* Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

SEE ALSO: **listp**

listp - LIST IOP HALT CODES

FUNCTION: Requests the punt codes (IOP halt codes) from the IOS. It formats the returned information, and writes it to standard out.

FORMAT: **listp** [**-D**] [**-I** *ios*] [**-z** *vmedev*]

-D	Sets debug mode in the **listp** program.
-I *ios*	Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.
-z *vmedev*	Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

SEE ALSO: **listo**

man - DISPLAYS ON-LINE MANUAL ENTRIES

FUNCTION: Displays information on a specific command. The format style is appropriate for terminals or printers. The **man** command further processes the entries to make the output more suitable for display terminals. Some information may be lost because of display limitations.

FORMAT: **man** *command*

command The desired command

mfclr - CLEAR MAINFRAME MEMORY WITH PATTERN

FUNCTION: Requests the IOS to clear mainframe memory, with the specified pattern, within the specified address. The pattern may be one of a group of preset patterns, or a pattern defined by the administrator. The use of this command could be detrimental to running systems.

FORMAT: **mfclr** [**-a** *add,addl*] [**-D**] [**-I** *ios*] [**-p** *pattern*] [**-z** *vmedev*]

-a *add,addl* Specifies the address range to clear in mainframe memory. Mainframe memory addresses are in words. Default is all of memory.

-D Sets debug mode in the **mfclr** program.

-I *ios* Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.

-p *pattern* *pattern* is a 22 octal digit pattern to write into memory. (For Local Memory it is a 6-digit pattern.) There is a group of predefined patterns of all zero bits, all one bits, or alternating ones and zeros. These can be specified as "zero", "one", "oz", "zo", "01", "10". (The preset patterns can be specified with alphabetic specifications in lowercase or uppercase.)

-z *vmedev* Specifies the Operator Workstation device name. The default is /dev/cray/cy0_00 for the Motorola microcomputer.

SEE ALSO: **mfpeek, mfpoke**

mfinit - INITIALIZE THE MAINFRAME

FUNCTION: Requests the IOS to initialize the mainframe. Note that **mfinit** will stop any running mainframe system.

FORMAT: **mfinit** [**-D**] [**-I** *ios*] [**-z** *vmedev*]

-D	Sets debug mode in the **mfinit** program.
-I *ios*	Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.
-z *vmedev*	Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

SEE ALSO: **mflosp**

mflosp - INITIALIZE THE MAINFRAME LOW-SPEED CHANNEL

FUNCTION: Requests the IOS to initialize the mainframe low-speed channel. This is typically done by the **start** command. This command should be used carefully, as normal low-speed communications between the mainframe and the IOS can be disrupted. **mflosp** has the same effect and functionality as the operating system console command **CRAY**.

FORMAT: **mflosp** [**-D**] [**-I** *ios*] [**-z** *vmedev*]

-D	Sets debug mode in the **mflosp** program.
-I *ios*	Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.
-z *vmedev*	Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

SEE ALSO: **mfinit**

mfmon - DISPLAYS CRAY CPU UTILIZATION

FUNCTION: Displays Cray CPU utilization information. It reads mainframe memory and extracts data from the Processor working storage (PWS) data structure. If the address of the PWS is not known, it can be found using the -f option.

FORMAT: **mfmon** [-f] [-p *pws*] [-I *ios*]

-f	Find PWS and print address
-p *pws*	PWS address; default is 0440
-I *ios*	Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.

EXAMPLES: The first **mfmon** command demonstrates the use of the -f option to obtain the PWS address. The second **mfmon** command uses that PWS address to display CPU utilization.

```
ows1001$ mfmon -f
pws address=0420
ows1001$ mfmon -p 0420

        CPU utilization

   cpu  unicos  user   idle  syswait
    -     --     --     --     --
    0     1      0      98     0
    1     10     2      88     0
    2     9      5      85     0
    3     12     1      86     0
    4     0      0      100    0
    5     0      99     0      0
    6     0      0      100    0
    7     0      0      100    0
ows1001$
```

mfpeek - READ MAINFRAME MEMORY

FUNCTION: Requests the IOS to read mainframe memory. The memory is formatted by the program and written to standard out.

FORMAT: **mfpeek** -a *add,addl* [-D] [-I *ios*] [-m *mode*] [-z *vmedev*]

-a *add,addl* Specifies the address range (in octal) to read from mainframe memory. *add* is the starting address and *addl* is the last address. Mainframe memory addresses are in words.

-D Sets debug mode in the **mfpeek** program.

-I *ios* Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.

-m *mode* The dump mode. Can be "word" or "parcel". The default is word.

-z *vmedev* Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

SEE ALSO: **mfpoke, mfclr**

mfpoke - WRITES A WORD INTO MAINFRAME MEMORY

FUNCTION: Requests the IOS to write a word into mainframe memory with the specified pattern, starting at the specified address. The pattern may be one of a group of preset patterns, or a pattern defined by the administrator. The use of this command may be detrimental to running systems.

FORMAT: **mfpoke -a** *add* **[-D] [-I** *ios***] [-p** *pattern***] [-z** *vmedev***]**

-a *add* Specifies the starting address (in octal) to write to in mainframe memory. Mainframe memory addresses are in words.

-D Sets debug mode in the **mfpoke** program.

-I *ios* Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.

-p *pattern* *pattern* is a 22 octal digit pattern to write into memory. There is a group of predefined patterns of all zero bits, all one bits, or alternating ones and zeros. These can be specified as "zero", "one", "oz", "zo", "01", "10". (The preset patterns can be specified with alphabetic specifications in lowercase or uppercase.)

-z *vmedev* Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

SEE ALSO: **mfpeek, mfclr**

mfsystem - CHANGE THE MAINFRAME SYSTEM TYPE IN THE IOS

FUNCTION: Requests the IOS to change the system type to either COS or UNICOS. This command is for IOS internal needs only. If no system type is specified, the current type is returned and the program formats the information and writes it to standard out.

FORMAT: **mfsystem** [-D] [-I *ios*] [-s *system*] [-z *vmedev*]

-D Sets debug mode in the **mfpoke** program.

-I *ios* Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.

-s *system* Specifies the system type as either COS or UNICOS. If this parameter is not used, then the current type is returned and written to standard out.

-z *vmedev* Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

SEE ALSO: **listp**(1M) in the UNICOS Administrator Commands Reference Manual, publication SR-2022

sentry - WAIT FOR IOS AND MAINFRAME MESSAGES

FUNCTION: Waits on a specific logical path, **/dev/cray/cy0_02**, for messages from the IOS and mainframe. The IOS punt messages, mainframe panic messages, and other unsolicited messages are sent on this path. The information is formatted and written to the system console and to the log file **/usr/cray/logs/bootlog**. **sentry** is started from the IOS boot script.

FORMAT: **sentry** [**-D**] [**-I** *ios*] [**-z** *vmedev*]

-D Sets debug mode in the **sentry** program.

-I *ios* Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.

-z *vmedev* Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

start - START THE CRAY MAINFRAME

FUNCTION: Starts the Cray mainframe using a mainframe binary and parameter file on the Operator Workstation. The **start** command does not check to see if there is a running IOS or mainframe system. Ensure that the mainframe is shut down before starting a new system. **start** uses the VME/FEI-3 device interface to communicate with the IOS.

FORMAT: **start** [**-D**] [**-I** *ios*] [**-m**] [**-u** *path*] [**-p** *param*] [**-P** $path_2$] [**-z** *vmedev*]

-D	Sets debug mode in the **start** program
-I *ios*	Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.
-m	Specifies not to run **mfinit** before starting the mainframe
-u *path*	Specifies the mainframe binary path name. The default is currently set to **/usr/cray/os/uts/unicos.**
-p *param*	Specifies the parameter file path name. The default is currently set to **/usr/cray/os/uts/param.**
-P $path_2$	Specifies the default path for all boot files; currently it is **/usr/cray/os/ios.** The default is not used when **-b** or **-k** are specified, as these options require full path names.
-z *vmedev*	Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

SEE ALSO: **boot**

sysdmp - CRAY SYSTEM DUMP INTERFACE

FUNCTION: Requests the IOS take a Cray system dump. A parameter block of system dump parameters are presented and may be modified by the user. Parameters consist of: CPU memory ranges, IOS Buffer Memory ranges, SSD memory ranges, dump device parameters, operating system, inclusion of the IOS, inclusion of CPU user tables, the system type, and the number of processors. Some of the errors that occur while the IOS is taking the dump can be retried. **sysdmp** prompts the user for a decision on whether to retry and returns that to the IOS.

FORMAT: **sysdmp** [-a] [-D] [-I *ios*] [-v] [-z *vmedev*]

-a	Sets auto mode. Does not ask questions, but uses the default IOS configuration for **sysdmp** parameters.
-D	Sets debug mode in the **sysdmp** program
-I *ios*	Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.
-v	Sets verbose mode. Prints useful information on standard out as required.
-z *vmedev*	Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

trdmp - REQUEST IOS TRACE INFORMATION

FUNCTION: Requests IOS trace information from a specific IOP. It formats the returned information, and writes it to standard out.

FORMAT: **trdmp** [-c *count*] [-D] [-i *iop*] [-I *ios*] [-l *local*] [-m *mos*] [-z *vmedev*]

-c *count*	Specifies the count of trace entries to print. Default is all.
-D	Sets debug mode in the **trdmp** program.
-i *iop*	Specifies the IOP from which to obtain trace information from. Default is 0 (MIOP).
-I *ios*	Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.
-l *local*	Get IOP Local Memory trace buffers. One or both of the -l or -m options must be specified.
-m *mos*	Get MOS trace buffers. One or both of the -l or -m options must be specified.
-z *vmedev*	Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

SEE ALSO: **trevt**

trevt - IOS TRACE EVENT CONTROL

FUNCTION: Allows the administrator to turn IOS tracing on or off.

FORMAT: **trevt** [**-d** *evt*] [**-D**] [**-e** *evt*] [**-i** *iop*] [**-I** *ios*] [**-m** *mos*] [**-s** *tsc*] [**-z** *vmedev*]

-d *evt*	Disables tracing of event *evt*.
-D	Sets debug mode in the **trevt** program.
-e *evt*	Enables tracing of event *evt*.
-i *iop*	Applies trace control command to IOP *iop*.
-I *ios*	Specifies the IOS interface to use for multiple IOS systems. The interface number corresponds to the IOS number, the default is zero.
-m *mos*	Enables or disables MOS trace. The option accepts the strings "on" or "off".
-s *tsc*	Specifies the trace function subcode. This option is used in conjunction with the **-d** and **-e** options.
-z *vmedev*	Specifies the Operator Workstation device name. The default is **/dev/cray/cy0_00** for the Motorola microcomputer.

SEE ALSO: **trdmp**

2.2 SERIAL LINE COMMANDS

This section describes the commands that use the serial (RS232) lines of the Operator Workstation and are entered at the AMPEX terminal (refer to table 2-3). The IOS commands are entered in uppercase.

Table 2-3. Serial Line Commands

Command	Description
CONC	Initiates FEI concentrator
DKERR†	Displays detailed disk error status
DSPL†	Displays a CPU formatted screen
ENDCONC	Terminates FEI concentrator
LPATH†	Displays NSC/FEI/VME logical path
MFCLR	Writes a pattern to Central Memroy
MFINIT	Performs confidence on mainframe hardware before deadstarting
MONITOR†	Monitors specified aspects of the CPU or IOS
NSC	Initiates NSC concentrator
NSCEND	Terminates NSC concentrator
STATION	Initiates the IOS station software
STP†	Displays statistics on the COS System Task Processor (STP)
TTY	Assigns a UNICOS interactive terminal
VME	Initiates the VMEbus interface driver
VMEND	Terminates the VMEbus interface driver

† The STATION command must be invoked before using

CONC - INITIATES CONCENTRATOR

FUNCTION: Initiates the front-end concentrator. Enter the CONC command to reinitiate the concentrator once it has been terminated by the ENDCONC command. If the command is successful, initialization is recognized with the following message:

CONCENTRATOR NUMBER *cn* INITIALIZED

FORMAT: CONC [*cn*]

cn Concentrator number which corresponds to a channel pair connecting the IOS to the front-end computer system; default is 0. Contact a your site administrator for the IOS configuration.

DKERR - DISPLAY DISK ERRORS

FUNCTION: Initiates the disk error display; will display approximately the last 25 disk error packets sent to be logged in the mainframe. The IOS station software must be initialized first through the STATION command.

FORMAT: DKERR,[*iop*]

iop	IOP number:	
	1	BIOP (default)
	2	IOP 2
	3	IOP 3 Only if DIOPs

AVAILABILITY: No prerequisites

EXAMPLE: DKERR

```
                Disk error display for IOP 1   Frame  0
OUTSTANDING DISK REQS:     3                                        FRAME  0

15:24:32  DD29 CHN 24 IDERR SK REC RTR   1 CYL   0 HED 0 SEC    21
          FLT      0 INL      0

15:29:13  DD39 CHN 33 UNT 0 WRIT  WRIT  REC RTR  3 CYL   10 HED 2 SEC    1
          CTL    611 GEN   5400 FCT    401 FGE  1400 FFC= *****
          C3M      0 C3O      0 C2M      0 C2O     0 OES= BUSY  FES= OK
          C1M      0 C1O      0 C0M      0 C0O     0 EXP      0 ACT      0
          S00  15400   6400   1000  70000      0

15:30:52  DD29 CHN 24 INTLK WR UNR RTR   0 CYL  15 HED 1 SEC    20
          FLT      0 INL      0

15:32:56  DD49 CHN 21 SLCT  SLCT  REC RTR    1 CYL  0 HED 3 SEC    0
          CTL  52200 GEN 177777 FCT  52003 FGE     0 FFC= ***** VSF      0
          B2M      0 B2O      0 B1M      0 B1O     0 OES= TOUT  FES= OK
          A2M      0 A2O      0 A1M      0 A1O     0 EXP      0 ACT      0
          S00 177777 177777 177777 177777  S04 177777 177777 177777 177777
          S08 177777 177777 177777 177777  S12 177777 177777 177777 177777
          S16 177777 177777 177777 177777  S20 177777 177777 177777 177777

                              End of data
```

DSPL - DISPLAY A CPU FORMATTED SCREEN

FUNCTION: Initiates the display of a screen that is built by the CPU. Displays are used to monitor Job Scheduler activity levels and general performance. The IOS station software must be initialized first through the STATION command.

FORMAT: DSPL,*display*

display The 2-character display name; valid names are determined by the CPU. If an invalid display name is entered, a help screen will be displayed showing the valid display names.

ENDCONC - TERMINATES CONCENTRATOR

FUNCTION: Terminates the concentrator initiated with the CONC command. When all resources are released, the following message appears:

CONCENTRATOR NUMBER *cn* TERMINATED

FORMAT: ENDCONC [*cn*]

cn Concentrator number which corresponds to a channel pair connecting the IOS to the front-end computer system; default is 0. Contact your site administrator for the IOS configuration.

LPATH - DISPLAY LOGICAL PATH

FUNCTION: Displays information on the Network Systems Corporation (NSC), Front-end Interface (FEI) logical path, or VME (FEI-3) connection. This command was developed for use with UNICOS/GOS and details protocols other than SCP. The IOS station software must be initialized first through the STATION command.

FORMAT: LPATH,*ichn*,*lp*

ichn	MIOP input channel number (octal)
lp	NSC/FEI/VME logical path number (octal)

EXAMPLE 1 (NSC display): LPA,34,2

IOP NSC/FEI LOGICAL PATH MONITOR

Input Channel - 34 Local Adapter Address (hex) - 00C1
Logical path - 2 Number of Messages Buffered - 0

Status - Read queued
Read active
Write active

	Read	Write
Errors detected (hex) -	00000000	00000000
Number of messages (hex) -	00000E37	00000E38
Bytes transferred (hex) -	00E3A000	00E3A000
NSCIO table address -	104010	

EXAMPLE 2 (FEI display): LPA,30,2

```
                    IOP NSC/FEI LOGICAL PATH MONITOR

  Input Channel -  30
  Logical path  -   2              Number of Messages Buffered -      0

  Status -  Read queued
            Read active

                                          Read            Write

  Number of messages (dec) -                 0                4
  Bytes transferred  (dec) -                 0                0
  Errors detected    (dec) -                 0                4
```

EXAMPLE 3 (VME display): LPA,30,5

```
                    IOP LOGICAL PATH MONITOR - VME

  Input Channel -  30
  Logical path  -   5              Number of Messages Buffered -      0

  Status -  Read queued
            Read active

                                          Read            Write

  Number of messages (dec) -                 0              211
  Bytes transferred  (dec) -                 0              320
  Errors detected    (dec) -                 0                0
  Last error code    (oct) -                 0                0

  VMERD table address      -            102414
  VMEWT table address      -            102434
```

MFCLR - WRITE PATTERN TO CENTRAL MEMORY

FUNCTION: The IOS MFCLR command writes a pattern to Cray Central Memory. This can be used to zero all of memory or to write zeros or some other pattern to selected areas of memory.

FORMAT: MFCLR [*fwa*] [*lwa*] [*pattern*]

fwa	Address of first word to write; default is 0
lwa	Address of last word to write; default is the last word of Central Memory
pattern	Pattern to write; default pattern is 0

NOTE

If an asterisk (*) is specified as the value for any of the MFCLR parameters, the default is used.

MFINIT - PERFORM CONFIDENCE TEST ON THE MAINFRAME HARDWARE

FUNCTION: The IOS MFINIT command performs a confidence test on the mainframe hardware components prior to deadstarting the operating system in the mainframe.

FORMAT: MFINIT

MONITOR - MONITOR SYSTEM PARAMETERS

FUNCTION: Generates a display to monitor a particular aspect of the Cray computer system. The IOS station software must be initialized first through the STATION command.

FORMAT: MONITOR,*display*

display Monitored information:

BMEM Percent of Buffer Memory currently in use by each IOP

BMIO Percent of maximum I/O to the 100 Mbyte channel connecting the IOP to Buffer Memory

BMX Percent of maximum data transfer, current device number, and mode on each configured Block Multiplexer channel

CMEM Percent of Cray Central Memory use by:
- System (fixed)
- User
- System buffers
- Unused

CPU† Percent of Cray mainframe time as follows:
- System time
- Idle time
- User time
- Time blocked for I/O

DISK Percent of maximum data transfer to each disk

DMEM Percent of disk buffers in use in each IOP

FMEM Percent of free Local Memory in use by each IOP

† COS only

FORMAT:
(continued)

HSPIO	Percent of maximum I/O to the 100 Mbyte channel connecting each IOP to the mainframe
HSX	Percent of maximum transfer rate realized over the HSX input and output channels
IOP	Percent of IOP central processor time not idle (for each IOP)
SSDCH†	Percent of maximum transfer rate realized over the SSD channel and the percentage of time the SSD channel is busy

EXAMPLE 1 (for a CRAY-1 computer system): MON,C

```
                    Monitor:  CPU Utilization
   System  31% *HHHHHHHHHHHHHHHHH
               *HHHHHHHHHHHHHHHHH
               *HHHHHHHHHHHHHHHHH
               *
     User  69% *HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
               *HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
               *HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
               *
     Idle   0% *
               *
               *
I/O Block   0% *
               *
               ******************************************************
               0         20         40         60         80         100
```

† COS only

EXAMPLE 2 (for a CRAY X-MP computer system): MON,C

```
                        Monitor:  CPU Utilization
                    *
 System 0    9%  *HHHHH
 System 1   10%  *HHHHH
 System 2   13%  *HHHHHHH
 System 3   15%  *HHHHHHHH
                    *
   User 0   89%  *HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
   User 1   78%  *HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
   User 2   61%  *HHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
   User 3   69%  *HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
                    *
   Idle 0    0%  *
   Idle 1    0%  *
   Idle 2    0%  *
   Idle 3    0%  *
                    *
I/O Block   14%  *HHHHHHH
                    ************************************************
                    0         20         40         60         80         100
```

NSC - INITIATES NSC CONCENTRATOR

FUNCTION: Initiates an NSC concentrator or ordinal. Enter the NSC command to reinitiate an NSC activity if it has been terminated.

FORMAT: NSC [*ord*]

ord	Ordinal that corresponds to a physical IOP channel pair. To determine which ordinal to enter, examine the MIOP configuration display by entering the **iosconf** command. If *ord* is not specified, the ordinal for the NSC concentrator is used.

NSCEND - TERMINATES NSC CONCENTRATOR

FUNCTION: Terminates an active NSC concentrator or ordinal. Either an NCS termination message or an error message (appears if an invalid ordinal number is specified) will be displayed.

FORMAT: NSCEND [*ord*]

ord Ordinal corresponding to a physical IOP channel pair. To determine which ordinal to enter, examine the MIOP configuration display by entering the iosconf command. If *ord* is not specified, the ordinal for the NSC concentrator is used.

STATION - INITIALIZES THE IOS STATION SOFTWARE

FUNCTION: Initializes the IOS station software. The specified MIOP console will then be able to be used for entering IOS station commands.

FORMAT: STATION [*num*]

num Number (0 through 2) of a console attached to the MIOP to be assigned for use as a station console. THe number maps to the physical channel pair connecting the terminal to the IOS as follows:

num	Channel Pair
0	40 and 41
1	42 and 43
2	44 and 45

The default at the Kernel console depends on the IOS configuration. Contact your site administrator for this information. The default at other consoles is the console at which the command was entered.

NOTE

Neither the dedicated MIOP Kernel console nor the UNICOS kernel console can be assigned as an IOS station software console.

STP - DISPLAY SYSTEM TASK PROCESSOR STATISTICS

FUNCTION: Provides the CRI site analyst with statistics pertaining to the COS system task processor (STP). The IOS station software must be initialized first through the STATION command.

FORMAT: STP

The information displayed is as follows:

Information	Description
Base	Base address (octal) of STP in memory
Name	ASCII name of the task or display entry
%CPU	Percentage of total CPU time in use by the displayed entry
%System	Percent of system time (CPU time used by the system as opposed to the user) being used by the displayed entry
Requests	Number of system requests made by the displayed task over the last display interval
Readies	Number of times the displayed task was readied by EXEC over the last display interval
Suspend	If YES is displayed, the displayed task is currently suspended; otherwise, NO is displayed.

EXAMPLE: STP

SYSTEM TASK PROCESSOR BASE: 27000 FRAME: 0

NAME	%CPU	%SYS	REQUESTS	READIES	SUSPEND
EXEC	19.23	47.78			
STP	20.97	52.18	5016	3948	
SCP	0.35	0.88	42	17	YES
EXP	14.88	36.99	3864	3649	YES
PDM	0.3	0.8	15	5	YES
DEC	0.0	0.0	0	0	YES
DQM	1.40	3.49	549	178	YES
MSG	0.4	0.11	12	4	YES
MEP	0.0	0.0	0	0	YES
SPM	0.0	0.0	0	0	YES
JSH	4.24	10.54	525	90	YES
JCM	0.0	0.0	0	0	YES
TQM	0.0	0.0	0	0	YES

TTY - ASSIGNS A UNICOS INTERACTIVE TERMINAL

FUNCTION: Assigns an MIOP console, other than the Kernel console, for use as a UNICOS interactive terminal. Normally, UNICOS automatically assigns consoles attached to channels 40, 42, and 44 as interactive consoles during initialization.

Character echoing and mode of input are controlled by flags received from the UNICOS tty driver. In raw mode, all characters are sent immediately to the Cray mainframe without interpretation.

FORMAT: TTY *dev* [*func*]

dev Octal channel number (for example, 40) of console to be connected to UNICOS as an interactive terminal

func Function to be performed; if not specified, a terminal session is initiated.

- END Stops the terminal session
- RAW Changes terminal to raw mode. This only affects the IOS terminal driver and does not affect the state of the UNICOS tty driver.

VME - INITIATES VMEbus DRIVER

FUNCTION: Initiates the VMEbus driver activities for a channel. Enter the VME command to reinitiate a VME activity if it has been terminated.

FORMAT: VME [*ch*[*mode*]]

ch Physical input channel number (octal) of the low-speed channel pair connecting the IOS to the VMEbus that is to be initiated. The default is the channel pair with the lowest ordinal configured as a VME connection.

mode Mode of execution. Enter G to enable the driver in graphics mode. This mode uses more Local Memory buffers, but guarantees maximum output transfer rate because output will not have to wait for resources. Entering no mode or any character other than G will enable the driver in networking mode. This mode requires fewer Local Memory buffers, but does not guarantee maximum output transfer rate or immediate resource availability.

VMEND - TERMINATES VMEbus DRIVER

FUNCTION: Terminates an active VMEbus driver. Either an VMEbus termination message or an error message (appears if an invalid ordinal number is specified) will be displayed.

FORMAT: VMEND [*ch*]

ch Physical input channel number (octal) of the low-speed channel pair connecting the IOS to the VMEbus that is to be terminated. The default is the channel pair with the lowest ordinal configured as a VMEbus connection.

APPENDIX SECTION

A. BOOTING THE SYSTEM

This appendix provides information on booting the Operator Workstation (OWS). Information is also provided on installing UNICOS from the OWS

A.1 BOOTING THE OWS

The following steps should be performed before the boot procedure can be invoked:

- Enable the external deadstart switch for the OWS channel on the IOS(s) that is to be booted.

  ****

  CAUTION

  The Motorola cannot be powered off while the deadstart switch is being enabled, as it will take down the IOS.

  ****

- Change the file **cl/include/ows.h** in the OWS source to reflect the correct channel and rebuild the code in **cl**. This step is only necessary if the channel for the OWS is not 026, or you do not want to hide the use of the -c option in a script.

- Put the IOS boot files (vme.boot, vmedump, kernel, and kernel.ov) in the directory **/usr/cray/os/ios**. This can be changed by changing the ows.h file and rebuilding, or by using or hiding the use of options on the **boot** command.

- Repeat the last procedure with the mainframe binary and parameter file. The default directory for the mainframe file is **/usr/cray/os/uts**.

The next step is to boot the system:

1. Login to the Motorola as **ops**, or **cray**, or some user that is in the **os** group.

2 Optional IOS deadstart diagnostics may be run at this point. Type **dboot -k dsdiag** to run the DSDIAG diagnostic, or **dboot -k cleario** to run the CLEARIO diagnostic. Refer to appendix E, IOS Deadstart Diagnostics, for more information.

3. Type the **boot** command. Include any necessary parameters that your site may need, see the **start** command in section 2. Wait for the prompt to be displayed. Any errors will be printed on the screen. You can abort the process by typing **control C** or **control D**.

4. Type the **start** command. Include any necessary parameters that your site may need. Some **start** errors will ask for user input before the procedure can continue. When the prompt is displayed, it signals that boot process has completed successfully.

A.2 INSTALLING UNICOS

A set of OWS utilities provide a mechanism for installing UNICOS. The utilities establish a UNICOS file system on a Cray disk drive. A standard UNICOS system, configured to use this file system, can then be run.

A.2.1 UTILITIES

The OWS utilities are described in section 2 and are provided to manage flaw tables and file systems on Cray disk drives. For all numeric input and output, a leading zero signifies an octal number. The utilities are as follows:

- **eft** Displays the Engineering Flaw Table
- **flaw** Initializes the UNICOS flaw table; adds, deletes, and lists flaws
- **ddcp** Copies data to or from the ROOT file system

A.2.2 CRAY DISK DESCRIPTION

The Cray disk type, location, and layout are contained in the file designated by the **-d** option on the **eft**, **flaw**, and **ddcp** commands. The information and format is similar to that describing disks in the UNICOS configuration and parameter files. It consists of a line

describing the disk, followed by lines describing the disk layout:

device iop channel [**UNIT=***unit*]
cylinder: slice
cylinder: slice
.
.
.

device — Types may be: DD19, DD29, DD39, DD40, DD49, or DD10

iop — IOP types are: BIOP, DIOP, or XIOP

channel — IOP channel number

unit — Unit designator (default is A):
DD-39 - A, B, C
DD-40 - A, B

cylinder — Starting cylinder number of the slice. Slices must be defined in ascending order.

slice — Type of slice:

ROOT — File system. This slice is read and written by **ddcp**

SPARES — Spares area. This slice is managed by the **flaw** command.

OTHER — Reserved areas. These slices are not read or written by **flaw** or **ddcp**. **eft** reads the EFT from a predefined location on disk that usually is in one of the reserved areas.

One **ROOT** and one **SPARES** slice must be defined. Multiple **OTHER** slices may be defined. Null lines in the file are ignored. For all numeric fields, a leading zero signifies an octal number.

A.2.2.1 Examples

This is an example of a configuration file, $0-A!-20. The disk is a DD-40 attached to channel 020 of the BIOP. The file system occupies cylinders 1 through 22, 20064 sectors, and the spares area is defined to be cylinders 706 and 707.

```
DD40 BIOP 020

0:    OTHER
1:    ROOT
23:   OTHER
706:  SPARES
708:  OTHER
```

This example shows the output of the **eft -d 40-A1-20** command.

```
EFT for device DD40 BIOP 020 UNIT=A

cylinder head sector  block count
  0616    010   031   363385   1
 01332     06   022   666066   1
 01716     06   021   888593   1
```

This example shows the output of the **flaw -l -d 40-A1-20** command.

```
Flaw list for device DD40 BIOP 020 UNIT=A

            flawed                          alternate
block  cylinder head sector     block  cylinder head sector
363385   0616    010   031     643878    01302    0    06
666066  01332     06   022     643879    01302    0    07
888593  01716     06   021     643880    01302    0   010
```

A.3 SYSTEM INSTALLATION

Installing a UNICOS system from the OWS involves establishing flaws on a Cray disk, copying a file system onto the disk, and booting a UNICOS system configured to use the installed file system. The example of the installation procedure described in this subsection assumes the following components:

- An IOS with the installation disk configured in files **kernel** and **kernel.ov**.
- A UNICOS system appropriate for the mainframe type, but with nothing built in disk configuration contained in file **unicos**.
- A 20000-block UNICOS root file system contained on tape device r40. This file system is a copy of a file system from a similar mainframe type dumped through the **ddcp** command.

The following steps illustrate the installation procedure. Input is shown in ***bold italic*** type.

1. Start the IOS.

```
ows$ boot -P.
ows$
```

2. Choose a disk to be used for the installation and decide on a layout. For this example, we are using the configuration file **40-A1-20** shown in the examples in subsection A.2.2.1.

3. Initialize the flaw table and add the flaws listed in the EFT.

```
ows$ flaw -i -d 40-A1-20
Initializing flaw table for device DD40 BIOP 020 UNIT=A
ows$ eft -b -d 40-A1-20 | xargs flaw -a -d 40-A1-20
Adding flaws to device DD40 BIOP 020 UNIT=A
ows$
```

4. Copy data to the **ROOT** file system.

```
ows$ ddcp -w -d 40-A1-20 < /dev/r40
20000 blocks written to device DD40 BIOP 020 UNIT=A
ows$
```

5. Generate a parameter file **param** to target **unicos** to the installed file system. The definitions for the file system and spares areas must correspond to the description in the file **40-A1-20**. File **param** contains:

```
CONFIG {
     49-A1-20:=DD49 BIOP 020(
               0:    CE120_1
               1:    root
               23:   fs1
               706:  SP120
               708:  CE120_2
               710:  fs2
               1415: CE120_3
          );

          root:=(
                root
          );

          ROOTDEV = root;
          SWAPDEV = root;
          PIPEDEV = root;
}
```

6. Boot the UNICOS system.

```
ows$ start -P.
ows$
```

B. OPERATOR WORKSTATION MAN PAGES

This appendix provides the Operator Workstation (OWS) man pages (see the man command in subsection 2.1). The man pages can be divided into three areas: OWS commands (subsection 2.1), FEI-3 driver, and miscellaneous commands for use on the OWS. Only the latter two are described in this appendix.

B.1 FEI-3 DRIVER

The man pages presented in this subsection describe the FEI-3 interface driver. The drivers described are: **cymux**, **ipcy**, and **scy**.

CYMUX()

NAME

cymux – FEI-3M Streams Multiplexor

DESCRIPTION

The FEI-3M Interface provides a hardware connection between the VME bus and the Cray Low Speed Channel. The *scy* driver (see scy(4)) provides the software interface to that hardware. However, *scy* is a raw (headerless) mode, single user driver for the FEI-3M hardware. Ordinarily, users would access *scy* through the multiplexor module called *cymux*. The linkage between *cymux* and *scy* would be set up by a daemon process, typically *tpid*, during the boot sequence. To the users, *cymux* appears as a UNIX device, accessible to the well known UNIX primitives, open, close, read, write and ioctl.

User interface.

Open. The user must be aware of some differences between *cymux* and other UNIX devices. The multiplexor is installed with many names, each name representing a logical channel, which identifies the user to *cymux*. For example, *cymux* could be installed with the following names:

```
/dev/cray/cy0_00
/dev/cray/cy0_01
/dev/cray/cy0_02
  etc.
```

The user then could attempt to open each installed name in turn until a valid file descriptor is returned, and use that for the duration of his session. Or, the system administrator could assign a user or group of users to a given driver name. If the open returned an **EBUSY** error, then that name was already in use.

Write. Data written to *cymux* must have:
1) a byte count < 32767 plus header (see below), and
2) a byte count divisible by 8.

Write data must be preceded by a 64 byte header, containing the logical channel number of the destination application in the Cray. Consequently, the user must know that application's logical channel number, something which is site defined.

This is the format of the 64 byte header used by the FEI-3M multiplexor module *cymux*. Fields marked (driver) are always filled in by the driver. Fields marked (user) must be filled in by the user.

```
unsigned        :15;        /* unused */
unsigned        ad:1;       /* Associated Data (driver) */
unsigned        seccnt:16;  /* sector count for IOS (driver) */
unsigned char   to[2];      /* logical to*  (user) */
unsigned char   from[2];    /* logical from (driver) */
unsigned        :16;        /* unused */
unsigned char   CYtype;     /* TCP/IP type */
unsigned char   CYoffset;   /* TCP/IP offset */
unsigned        :32;        /* unused */
unsigned char   data[48];   /* data (user) */
```

CYMUX()

*Logical TO:

The user must fill in the logical TO field expected by the receiving end (the FROM field is always filled in by cymux).

Because of the streams architecture, there is no way to inform a user if a write fails. *Cymux* does not guarantee delivery of data, consequently, error recovery is the responsibility of the user.

Read. Read data must satisfy the same length requirements as write data. As write data must be preceded by a header, read data will also be received with a header in front of it.

The streams mechanism by default will queue any unread data (ie. the user read was smaller than the amount of data received). The stream head will deliver the unread data on the next read. Applications expecting a header as the first portion of data must set the stream read options to discard unread data:

```
fd = open("/dev/cray/cy0_00", O_RDWR); /* open multiplexor */
r = ioctl(fd, I_SRDOPT, RMSGD);              /* discard unread data */
```

Cymux sets no timer when a user read is received. It is the application's responsibility to time out reads. The following code fragment highlights how:

```
#include <sys/signal.h>
-
-
        int rvitalarm();
        signal(SIGALRM, rvitalarm);
        alarm(10);                          /* ten second timer */
        r = read(fdcymux, buf, len);
        alarm(0);                           /* cancel timer */
        if (r < 0) {                        /* error or timer expired */
                -
                -
        }
-
rvitalarm() {}
-
```

Ioctl. The ioctl commands to *cymux* are to establish and break the multiplexor linkage between *cymux* and *scy*. The ioctls allowed are:
1) those defined by the streams mechanism for multiplexors, I_LINK and I_UNLINK: The following code fragment shows how a daemon process would do this:

```
if (fork()) exit;                   /* run in background */
fdm = open("/dev/cray/cy0_00", O_RDWR);     /* open multiplexor */
fdd = open("/dev/cy0", O_RDWR);     /* open driver */
r = ioctl(fdm, I_LINK, fdd);        /* link driver under mux */
close(fdd);                                 /* fdd not needed */
pause();                            /* hold mux open forever */
```

CYMUX()

2) CYRAW - this ioctl sets "raw" mode for the logical path; there is no modification of the user's header.

```
fd = open("/dev/cray/cy0_00", O_RDWR);  /* open multiplexor */
cyiocblk.ic_cmd = CYRAW;                /* set command */
cyiocblk.ic_timout = 2;                 /* set timeout value */
cyiocblk.ic_len = 0;                    /* length of data */
cyiocblk.ic_dp = 0;                     /* addr to put data */
r = ioctl(fd, I_STR, &cyiocblk);        /* send ioctl to scy */
```

3) CYDSTART - this ioctl sets "raw" mode and is also handled by the scy driver.

```
fd = open("/dev/cray/cy0_00", O_RDWR);  /* open multiplexor */
cyiocblk.ic_cmd = CYDSTART;             /* set command */
cyiocblk.ic_timout = 2;                 /* set timeout value */
cyiocblk.ic_len = 0;                    /* length of data */
cyiocblk.ic_dp = 0;                     /* addr to put data */
r = ioctl(fd, I_STR, &cyiocblk);        /* send ioctl to scy */
```

HARDWARE INSTALLATION

Installing the Cray-VME Interface hardware is covered in other documentation, see the list of publications below for details. It is worth noting that to run the Interface loopback tests, the Interface must be configured as a 50 Mbit channel master. In binary, without explanation, the switch settings would be:

```
control board sw1    1110111x    /* 1 is open, off, active */
control board sw2    10xxx000    /* x is unused */
control board sw3    1110000x    /* base address 0xe000 - could vary */
```

INSTALLATION

Compiling and archiving.

The *cymux* driver will typically be in a special directory, say /usr/cray/src/uts/streams. The multiplexor name is *cymux.c*. The FEI3 Streams driver *scy.c* and the TCP/IP interface module *ipcy.c*, are also installed in /usr/cray/src/uts/streams. See *scy (4)* and *ipcy (4)* for more details. The driver, multiplexor, and the IP interface module can simply be compiled and archived in lib.io by issuing 'make install' or 'make' in the /usr/cray/src/uts/streams directory. The multiplexor can be compiled and archived by issuing 'make cymux.o'.

Changes to the **makefile** *file..*

Set CFLAGS (near line 4 of the makefile) as follows:

```
CFLAGS=-DINTFS=2 -DIOS
```

INTFS is the number of FEI-3 board sets installed; IOS is for Cray X/MP and Y/MP systems only - remove this definition for Cray 2 systems.

Changes to the **master** *file..*

Add the following lines to */usr/src/uts/m68k/cf/m68k/master*, being sure that the major numbers are unique for *cymux*. Also insure there are at least 16 data blocks of size 4096, and the maximum stream message size is at least 32767:

```
cymux          0 1000   244 cymx    0    0    66    8    0

nblk4096       NBLK4096          16
strmsgsz       STRMSGSZ          32767
```

Changes to the **dfile** *file.*

Add the following line to the *dfile*:

```
cymux      0        0        0   16
```

Build a new kernel.

Build a new kernel while in the configuration directory /usr/src/uts/m68k/cf/m68k.

```
make
mv /stand/sysV68 /stand/osysV68  # save old binary
mv sysV68 /stand
```

Define the driver names to UNIX. This is done by typing "make devs" or just "make" in the streams directory.

```
mknod /dev/cray/cy0_00 c maj 0
mknod /dev/cray/cy0_01 c maj 1
mknod /dev/cray/cy0_02 c maj 2
        etc.
```

Finally, boot new kernel.

ERRORS

Open

Open *cymux* will fail if one of the following are true:

If the minor device number is out of bounds. [EINVAL]
If the device is already in use. [EBUSY]

Read

Read will fail if one of the following are true:

If the byte count is less than a header. [EINVAL]
If the byte count is greater than the maximum. [EINVAL]
If the user interrupts the read. [EINTR]

Write

Write will fail if one of the following are true:

If the byte count is less than a header. [EINVAL]
If the byte count is greater than the maximum. [EINVAL]
If the byte count is not a multiple of 8. [EINVAL]
If cymux is not linked to to scy. [ENODEV]

Ioctl

Ioctl will fail if one of the following are true:

If the command is unknown. [EINVAL]
If the address to copy data to is invalid. [EFAULT]
If cymux is not linked to to scy. [ENODEV]

SEE ALSO

Cray VMEbus Interface Reference Manual, CRI publication HR-0322
Cray VMEbus Driver Technical Note, CRI publication SN-0244

scy(4), ipcy(4), scyadmin(8), scytest(8)

BUGS

Changing the number of logical channels defined to cymux requires modifying the value of CYMX-MAX in cymux.c and recompiling.

Writes can fail silently for a variety of reasons.

IPCY()

NAME

ipcy – TCP/IP to Cray Low Speed Channel Interface Module

DESCRIPTION

The FEI-3M Interface provides a hardware connection between the VME bus and the Cray Low Speed Channel. The *scy* driver and *cymux* multiplexor provide the software interface to that hardware. Ipcy provides an interface between Motorola's Network Services Extension (NSE) INTERNET protocol module (IP) and the FEI-3M software. Ipcy appears as a streams module and is pushed above cymux, (which is linked above scy), and below the IP multiplexor. Note: ipcy is not installed as a streams device driver. The following code fragment shows how the a program could link ipcy for the Cray channel interface (note: this is normally done by tpid):

```
int fdd,fdm;
int r;
fdd = open("/dev/inet/if/scy", O_RDWR);
fdm = open("/dev/inet/if/cymux01", O_RDWR);
r = ioctl(fdm, I_LINK, fdd);          /* link driver under mux */
close(fdd);
r = ioctl(fdm, I_PUSH, "ipcy");       /* push ip intf above mux */
```

The main function of ipcy is to convert data outbound from IP into a form usable by cymux and scy (and ultimately the Cray), and vice versa convert data inbound from the Cray to a form usable by IP. Data in either direction appears as an mblk_t with a data type of M_DATA. Outbound IP data is preceded by a common interface structure called an *ifarg*. Inbound Ip data is preceded by a Cray channel header structure called a *cy_msg*. Ipcy's job is to convert these headers.

An additional function of ipcy is to respond to network requests from the upstream IP. Some appear as M_PROTO requests and some as M_IOCTL requests. Of importance here is the network name is sent in a response to an M_PROTO IP_INFO request. This request is passed downstream to cymux and the name "cy*n*" is returned, where *n* is the number of the FEI-3M board set 0-3.

INSTALLATION

Compiling and archiving.

The TCP/IP interface module is named *ipcy.c*, and is also installed on a special directory with *scy* and *cymux*, say /usr/cray/src/uts/streams. The driver, multiplexor, and the IP interface module can simply be compiled and archived in lib.io by issuing 'make install' in the /usr/cray/src/uts/streams directory.

Changes to the* sys/streams.h *file. (this is done by typing "make includes" in the /usr/cray/src/uts/streams directory).

Add the following declaration before the fmodsw array:

extern struct streamtab ipcyinfo;

Add the following line to the /usr/include/sys/streams.h file in the fmodsw array:

"ipcy", &ipcyinfo, /* IP to Cray channel interface module */

Build a new kernel.

IPCY()

Build a new kernel while in the configuration directory /usr/src/uts/m68k/cf.

```
make
mv /stand/sys/v68 /stand/osysV68  # save old binary
mv sysV68 /stand
```

Finally, boot new kernel.

CONFIGURATION

changes to /etc/hosts

Select a unique network number for the Cray-Motorola network. Note this is a simple point to point configuration. Here is a sample from the hosts file:

```
87.0.0.01       motorola-fei3
87.0.0.05       sn228-mfei3
```

Changes to /etc/tpid.conf file

This file must be modified to allow inclusion of the Cray channel interface. For routing outbound IP traffic, ipcy uses the lower 16 bits of the destination network number as the destination logical TO field (See *scy -l* for details on logical TO). Likewise, the lower 8 bits of the source network number are the logical FROM field and must match the logical channel of cymux opened by tpid. For the example /etc/hosts file above, this means that tpid must open /dev/cray/cy0_01. This driver name is simply placed in the tpid.conf file like this:

```
#       Cray channel interface(s)
cymux,mux,cy0:/dev/cray/cy0_01::ipcy:
cy0:/dev/cray/cy0:motorola-fei3::up
#
#       second interface if present
#
cymux,mux,cy1:/dev/cray/cy0_01::ipcy:
cy1:/dev/cray/cy1:moto2-fei3::up
```

If you don't want to ifconfig the interface, just push cymux on the stream for deadstart, station, etc., do this:

```
#       Cray channel interface(s)
cymux,mux,cy0:/dev/cray/cy0_01::ipcy:
cy0:/dev/cray/cy0:::down
```

Note that the name "cy*n*" matches the name hard coded in the cymux.c source code, and the name 'uname'-fei3 must match the name in /etc/hosts.

FILES

/usr/adm/inetd	log file for inetd
/etc/hosts	network file
/usr/include/sys/streams.h	contains fmodsw table

SEE ALSO

Cray VMEbus Interface Reference Manual, CRI publication HR-0322
Cray VMEbus Driver Technical Note, CRI publication SN-0244

scy(4), cymux(4), scyadmin(8), scytest(8),

BUGS

Changing the logical channel used by IP requires modifying the tpid.conf file.

The name of the interface must be "cy*n*".

Ifconfig cy0 down does not dismantle the IP-ipcy-cymux-scy stream. The only way to dismantle the stream is to kill inetd.

SCY()

NAME

scy – FEI-3M to Cray Low Speed Channel Interface

DESCRIPTION

The FEI-3M Interface provides a hardware connection between the VME bus and the Cray Low Speed Channel. The *scy* driver provides the software interface to that hardware. *Scy* is a raw (headerless) mode, single user driver for the FEI-3M hardware. Ordinarily, users would access scy through the multiplexor module called *cymux*. The linkage between *cymux* and *scy* would be set up by a daemon process, typically by *inetd*, during the boot sequence. To the users, *scy* and *cymux* appear as a UNIX devices, controlled by the well known UNIX primitives, open, close, read, write and ioctl.

User interface.

Open. The user must be aware of some differences between *cymux* and other UNIX devices. The multiplexor is installed with many names, each name representing a logical channel, which identifies the user to *cymux*. For example, *cymux* could be installed with the following names:

```
/dev/cray/cy0_00
/dev/cray/cy0_01
/dev/cray/cy0_02
  etc.
```

The user then could attempt to open each installed name in turn until a valid file descriptor is returned, and use that for the duration of his session. Or, the system administrator could assign a user or group of users to a given driver name. If the open returned an **EBUSY** error, then that name was already in use.

The interface to *scy* is more straightforward - it is merely opened as */dev/cray/cy0*.

Write. Data written to the Low Speed Channel must
1) begin on a fullword (32 bit) boundary,
2) the byte count must be < 65536 plus header (see below), and
3) the byte count must be divisible by 8.

If using the multiplexor, write data must be preceded by a 64 byte header containing the logical channel number of the destination application in the Cray. Consequently, the user must know that application's logical channel number, which is site defined.

This is the format of the 64 byte header used by the FEI-3M multiplexor module *cymux*. Fields marked (driver) are always filled in by the driver. Fields marked (user) must be filled in by the user.

```
unsigned        :15;        /* unused */
unsigned        ad:1;       /* Associated Data (driver) */
unsigned        seccnt:16;  /* sector count for IOS (driver) */
unsigned char   to[2];      /* logical to*  (user) */
unsigned char   from[2];    /* logical from (driver) */
unsigned        :16;        /* unused */
unsigned char   CYtype;     /* TCP/IP type */
unsigned char   CYoffset;   /* TCP/IP offset */
unsigned        :32;        /* unused */
unsigned char   data[48];   /* data (user) */
```

SCY()

NAME

scy – FEI-3M to Cray Low Speed Channel Interface

DESCRIPTION

The FEI-3M Interface provides a hardware connection between the VME bus and the Cray Low Speed Channel. The *scy* driver provides the software interface to that hardware. *Scy* is a raw (headerless) mode, single user driver for the FEI-3M hardware. Ordinarily, users would access scy through the multiplexor module called *cymux*. The linkage between *cymux* and *scy* would be set up by a daemon process, typically by *inetd*, during the boot sequence. To the users, *scy* and *cymux* appear as a UNIX devices, controlled by the well known UNIX primitives, open, close, read, write and ioctl.

User interface.

Open. The user must be aware of some differences between *cymux* and other UNIX devices. The multiplexor is installed with many names, each name representing a logical channel, which identifies the user to *cymux*. For example, *cymux* could be installed with the following names:

```
/dev/cray/cy0_00
/dev/cray/cy0_01
/dev/cray/cy0_02
  etc.
```

The user then could attempt to open each installed name in turn until a valid file descriptor is returned, and use that for the duration of his session. Or, the system administrator could assign a user or group of users to a given driver name. If the open returned an **EBUSY** error, then that name was already in use.

The interface to *scy* is more straightforward - it is merely opened as */dev/cray/cy0*.

Write. Data written to the Low Speed Channel must
1) begin on a fullword (32 bit) boundary,
2) the byte count must be < 65536 plus header (see below), and
3) the byte count must be divisible by 8.

If using the multiplexor, write data must be preceded by a 64 byte header containing the logical channel number of the destination application in the Cray. Consequently, the user must know that application's logical channel number, which is site defined.

This is the format of the 64 byte header used by the FEI-3M multiplexor module *cymux*. Fields marked (driver) are always filled in by the driver. Fields marked (user) must be filled in by the user.

```
unsigned        :15;        /* unused */
unsigned        ad:1;       /* Associated Data (driver) */
unsigned        seccnt:16;  /* sector count for IOS (driver) */
unsigned char   to[2];      /* logical to*  (user) */
unsigned char   from[2];    /* logical from (driver) */
unsigned        :16;        /* unused */
```

SCY()

```
unsigned char   CYtype;         /* TCP/IP type */
unsigned char   CYoffset;       /* TCP/IP offset */
unsigned         :32;           /* unused */
unsigned char   data[48];       /* data (user) */
```

*Logical TO:

The user must fill in the logical TO field expected by the receiving end (the FROM field is always filled in by the driver).

Because of the streams architecture, there is no way to inform a user if a write fails. *Scy* and *cymux* do not guarantee delivery of data, consequently, error recovery is the responsibility of the user.

Read. Read data must satisfy the same address and length requirements as write data. As write data must be preceded by a header, read data will also be received with a header in front of it.

The streams mechanism by default will queue any unread data (ie. the user read was smaller than the amount of data received). The stream head will deliver the unread data on the next read. Applications expecting a header as the first portion of data must set the stream read options to discard unread data:

```
fd = open("/dev/cray/cy0_00", O_RDWR); /* open multiplexor */
r = ioctl(fd, I_SRDOPT, RMSGD);                 /* discard unread data */
```

Scy and *cymux* set no timers when a user read is received. It is the application's responsibility to time out reads. The following code fragment highlights how:

```
#include <sys/signal.h>
-
-
        int rvitalarm();
        signal(SIGALRM, rvitalarm);
        alarm(10);                              /* ten second timer */
        r = read(fdcymux, buf, len);
        alarm(0);                               /* cancel timer */
        if (r < 0) {                            /* error or timer expired */
                -
                -
        }
-
rvitalarm() {}
-
```

Ioctl. The ioctl commands to *scy* are intended for diagnostic purposes for system administrators and *scy* driver developers. They are:

CYPEEK:	dump readable hardware registers to user
CYMCLR:	Master clear the Interface
CYDISC:	Disable disconnect interrupts for uncabling channel
CYDSTART:	Send IO Master Clear

SCY()

HARDWARE INSTALLATION

Installing the Cray-VME Interface hardware is covered in other documentation, see the list of publications below for details. It is worth noting that to run the Interface loopback tests, the Interface must be configured as a 50 Mbit channel master. In binary, without explanation, the switch settings would be:

```
        control board sw1       1110111x        /* 1 is open, off, active */
        control board sw2       10xxx000        /* x is unused */
        control board sw3       1110000x        /* base address = 0xe000 */

OR      control board sw3       0111000x        /* base address = 0x7000 */
```

INSTALLATION

Compiling and archiving.

The *scy* and *cymux* drivers will typically be in a special directory, say /usr/cray/src/uts/streams. The driver name is *scy.c*, the multiplexor is *cymux.c*. The TCP/IP interface module is named *ipcy.c*, and is also installed on /usr/cray/src/uts/streams. See *ipcy (4)* for more details. The driver, multiplexor, and the IP interface module can simply be compiled and archived in lib.io by issuing 'make' or 'make install' in the /usr/cray/src/uts/streams directory.

Changes to the **master** *file..*

Add the following lines to */usr/src/uts/m68k/cf/master*, being sure that the major numbers are unique for *scy* and *cymux*, Also insure there are at least 16 data blocks of size 4096, and the maximum stream message size is at least 32767:

```
scys      0   0    4 scy   4   0    67    1   0
scy       4 1000 5104 scy   0   0    67    1   4
cymux     0 1000  244 cymx  0   0    66    8   0

nblk4096        NBLK4096        16
strmsgsz        STRMSGSZ        32767
```

Changes to the **dfile** *file.*

Add the following lines to the *dfile*, being sure the address (fffffc000) and interrupt vector numbers (200-20C) do not conflict with any other entries:

```
scys      0       fffffc000 0
scy       200     0       4
scy       204     0       4
scy       208     0       4
scy       20c     0       4
cymux     0       0       0   8
```

SCY()

Ioctl information is passed to *scy* (even if using the multiplexor) by the strioctl structure (in stropts.h) provided by the streams mechanism. Here is an example of its use:

```
#include <sys/fcntl.h>                  /* must be included */
#include <sys/stropts.h>          /* must be included */
#include "scyioctl.h"                   /* must be included */
struct strioctl cyiocblk;
struct cv_peek cv_peek;
        -
        -
        fd = open("/dev/cray/cy0_00", O_RDWR); /* open multiplexor */
        cyiocblk.ic_cmd = CYPEEK;               /* set command */
        cyiocblk.ic_timout = 2;         /* set timeout value */
        cyiocblk.ic_len = sizeof cv_peek; /* length of data */
        cyiocblk.ic_dp = (char *) &cv_peek;     /* addr to put data */
        r = ioctl(fd, I_STR, &cyiocblk); /* send ioctl to scy */
```

The ioctl commands to *cymux* are to establish and break the multiplexor linkage between *cymux* and *scy*. The ioctls allowed are:
1) those defined by the streams mechanism for multiplexors, I_LINK and I_UNLINK. The following code fragment shows how a daemon process would do this:

```
        if (fork()) exit;               /* run in background */
        fdm = open("/dev/cray/cy0_00", O_RDWR);     /* open multiplexor */
        fdd = open("/dev/cray/cy0", O_RDWR);        /* open driver */
        r = ioctl(fdm, I_LINK, fdd);    /* link driver under mux */
        close(fdd);                             /* fdd not needed */
        pause();                        /* hold mux open forever */
```

2) CYRAW - this ioctl sets "raw" mode for the logical path; there is no modification of the user's header.

```
        fd = open("/dev/cray/cy0_00", O_RDWR); /* open multiplexor */
      cyiocblk.ic_cmd = CYRAW;                  /* set command */
      cyiocblk.ic_timout = 2;                   /* set timeout value */
      cyiocblk.ic_len = 0;                      /* length of data */
      cyiocblk.ic_dp = 0;               /* addr to put data */
      r = ioctl(fd, I_STR, &cyiocblk);    /* send ioctl to scy */
```

3) CYDSTART - this ioctl sets "raw" mode and is also handled by the scy driver.

```
      fd = open("/dev/cray/cy0_00", O_RDWR);  /* open multiplexor */
      cyiocblk.ic_cmd = CYRAW;                /* set command */
      cyiocblk.ic_timout = 2;           /* set timeout value */
      cyiocblk.ic_len = 0;              /* length of data */
      cyiocblk.ic_dp = 0;               /* addr to put data */
      r = ioctl(fd, I_STR, &cyiocblk);    /* send ioctl to scy */
```

SCY()

for the second board:

```
scys      0      ffff7000 0
scy       210    0        4
scy       214    0        4
scy       218    0        4
scy       21c    0        4
```

Build a new kernel.

Build a new kernel while in the configuration directory /usr/src/uts/m68k/cf.

```
make
mv /stand/sys/v68 /stand/osysV68  # save old binary
mv sysV68 /stand
```

Define the driver names to UNIX. (The following is done automatically by typing "make" or "make devs" in the /usr/cray/src/uts/streams directory.)

```
mknod /dev/cray/cy0 c smaj 0
mknod /dev/cray/cy0_00 c maj 0
mknod /dev/cray/cy0_01 c maj 1
mknod /dev/cray/cy0_02 c maj 2
mknod /dev/cray/cy0_03 c maj 3
mknod /dev/cray/cy0_04 c maj 4
mknod /dev/cray/cy0_05 c maj 5
      etc.
```

for the second board:

```
mknod /dev/cray/if/scy1 c smaj 1
mknod /dev/cray/cy1_00 c maj 64
mknod /dev/cray/cy1_01 c maj 65
mknod /dev/cray/cy1_02 c maj 66
mknod /dev/cray/cy1_03 c maj 67
mknod /dev/cray/cy1_04 c maj 68
mknod /dev/cray/cy1_05 c maj 69
```

Finally, boot new kernel.

ERRORS

Open

Opening *cymux* will fail if one of the following are true:

If the minor device number is out of bounds. [EINVAL]
If the device is already in use. [EBUSY]

SCY()

Opening *scy* will fail if one of the following are true:

If the open is a CLONEOPEN. [EBUSY]
If scy is already opened. [EBUSY]

Read

Read will fail if one of the following are true:

If the byte count is less than a header (cymux). [EINVAL]
If the byte count is greater than the maximum. [EINVAL]
If the user interrupts the read. [EINTR]

Write

Write will fail if one of the following are true:

If the byte count is less than a header (cymux). [EINVAL]
If the byte count is greater than the maximum. [EINVAL]
If the byte count is not a multiple of 8. [EINVAL]
If the address is not a fullword (32 bit) boundary. [EINVAL]
If cymux is not linked to to scy (cymux). [ENODEV]

Ioctl

Ioctl will fail if one of the following are true:

If the command is unknown. [EINVAL]
If the address to copy data to is invalid. [EFAULT]
If cymux is not linked to to scy (cymux). [ENODEV]

SEE ALSO

Cray VMEbus Interface Reference Manual, CRI publication HR-0322
Cray VMEbus Driver Technical Note, CRI publication SN-0244

scyadmin(8), scytest(8), cymux(4), ipcy(4)

BUGS

Flow control is not implemented.

Writes can fail silently for a variety of reasons.

B.2 MISCELLANEOUS COMMANDS

This subsection describes miscellaneous commands that may be helpful in the use of the OWS. The commands described are:

- **clear** Clears terminal screen
- **fortune** Prints a random adage
- **more** Controls scrolling of files
- **screen** Refreshes the display
- **syslog** Controls the system log
- **syslogd** Logs system messages
- **test** Initializes terminal

NAME

clear – Clears terminal screen

SYNOPSIS

clear

DESCRIPTION

The **clear** command clears your screen. It looks in the environment for the terminal type and then in **/usr/lib/terminfo** to determine how to clear the screen.

NOTES

This command is based on the Fourth Berkeley Software Distribution under license from The Regents of the University of California. Most Berkeley commands, including this one, are located in **/usr/ucb**. Hence, to use this command, either enter the full path (**/usr/ucb/clear**) or include **/usr/ucb** in your PATH environment variable.

FILES

/usr/lib/terminfo	Terminal capability data base

NAME

fortune – print a random, hopefully interesting, adage

SYNOPSIS

/usr/games/fortune [–] [–wslao]

DESCRIPTION

Fortune with no arguments prints out a random adage. The flags mean:

–w Waits before termination for an amount of time calculated from the number of characters in the message. This is useful if it is executed as part of the logout procedure to guarantee that the message can be read before the screen is cleared.

–s Short messages only.

–l Long messages only.

–o Choose from an alternate list of adages, often used for potentially offensive ones.

–a Choose from either list of adages.

FILES

/usr/games/lib/fortunes.dat

AUTHOR

Ken Arnold

NAME

more, page – Lets you control scrolling of files while perusing them

SYNOPSIS

more [**–c**] [**–d**] [**–f**] [**–l**] [**–s**] [**–u**] [*–n*] [*+command*] [*files*]

page [**–c**] [**–d**] [**–f**] [**–l**] [**–s**] [**–u**] [*–n*] [*+command*] [*files*]

DESCRIPTION

The **more** command is a filter that lets you examine a continuous text one screenful at a time. It normally pauses after each screenful, printing **--More--** at the bottom of the screen. If you then type a carriage return, one more line is displayed. If you hit the spacebar, another screenful is displayed. Other possibilities are enumerated later.

Options:

–c Draws each page by beginning at the top of the screen and erasing each line just before drawing on it. This avoids scrolling the screen, making it easier to read while **more** is writing. This option is ignored if the terminal does not have the ability to clear to the end of a line.

–d Prompts the user with the message "Hit space to continue, Rubout to abort" at the end of each screenful. This is useful if you are using **more** as a filter in some setting, such as a class, where many users may be unsophisticated.

–f Counts logical, rather than physical screen lines. That is, long lines are not folded.

–l Does not treat CONTROL-L (form feed) specially. If you do not specify this option, **more** pauses after any line that contains a CONTROL-L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen is cleared before the file is printed.

–s Squeezes multiple blank lines from the output, producing only one blank line. This option maximizes the useful information present on the screen.

–u Suppresses underline processing. Normally, **more** handles underlining such as produced by *nroff* in a manner appropriate to the particular terminal: if the terminal can perform underlining or has a stand-out mode, **more** outputs appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file.

–n Specifies an integer that is the size (in lines) of the window, which **more** uses instead of the default

+command

Where *+command* can be either *+linenumber* or *+/pattern*, as follows:

+linenumber	Starts up at *linenumber*
+/pattern	Starts up two lines before the line containing the regular expression *pattern*

If **more** is invoked as **page**, then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and $k - 1$ rather than $k - 2$ lines are printed in each screenful, where k is the number of lines the terminal can display.

The **more** command looks in the **/usr/lib/terminfo** directory to determine terminal characteristics and the default window size defined by TERM environment varible. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

The **more** command looks in the environment variable MORE to preset any desired flags. For example, if you prefer to view files using the **–c** mode of operation, the csh(1) command

"setenv MORE -c"

or the sh(1) command sequence

"MORE='-c' ; export MORE"

would cause all invocations of **more**, including invocations by programs such as **man**(1) and **msgs**(1), to use this mode. Normally, you place the command sequence that sets up the MORE environment variable in the **.cshrc** or **.profile** file.

If **more** is reading from a file rather than a pipe, a percentage is displayed along with the --More-- prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences that may be typed when **more** pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1) :

i<space>
: Displays *i* more lines, (or another screenful if no argument is given)

i CONTROL-d
: Displays 11 more lines (a "scroll"). If *i* is given, then the scroll size is set to *i*.

*i*d
: Acts the same as CONTROL-d

*i*z
: Acts the same as typing a space except that *i*, if present, becomes the new window size

*i*s
: Skips *i* lines and prints a screenful of lines

*i*f
: Skips *i* screenfuls and print a screenful of lines. Skips one line if *i* is not present.

q or Q
: Exits from **more**

=
: Displays the current line number

v
: Starts up the editor *vi* at the current line

h
: Help command; gives a description of all the **more** commands.

i/expr
: Searches for the *i*-th occurrence of the regular expression *expr*. If less than *i* occurrences of *expr* exist, and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. You can use the erase and kill characters to edit the regular expression. Erasing back past the first column cancels the search command.

*i*n
: Searches for the *i*-th occurrence of the last regular expression entered

'
: (Single quote) Goes to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!command
: Invokes a shell with *command*. The characters '%' and '!' in "command" are replaced with the current file name and the previous shell command respectively. If no current file name exists, '%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i:n
: Skips to the *i*-th next file given in the command line (skips to last file if n doesn't make sense)

i:p
: Skips to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, then **more** goes back to the beginning of the file. If *i* does not make sense, **more** skips back to the first file. If **more** is not reading from a file, the bell is rung and nothing else happens.

SCREEN (1M)

NAME

screen – Refreshes the display at a regular interval

SYNOPSIS

screen [**–r** *rate*] [*command arguments*]

DESCRIPTION

The **screen** command refreshes the display at a regular interval for any command. The valid command line options, arguments, and operands are as follows:

–r *rate*

Refresh rate in seconds; the default is 5.

command arguments

Name of a command and the desired options (*arguments*)

Once in **screen**, you may use the following commands:

+ Advances to next page

– Returns to previous page

> Increases refresh interval by 1 second

< Decreases refresh interval by 1 second

R Scrolls through all pages

r Ends scroll

q (CTRL-d)

Quits **screen**

CTRL-c

Ends current command and prompts for new command

:f Displays the current file name and line number.

:q or :Q Exits from **more** (same as q or Q).

. (Dot) repeats the previous command.

The commands take effect immediately; that is, it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may hit the line kill character to cancel the numerical argument being formed. In addition, the user may hit the erase character to redisplay the --More--(xx%) message.

At any time when output is being sent to the terminal, you can hit the quit key (normally Control-\). **More** stops sending output and displays the usual --More-- prompt. You may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, because any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

The terminal is set to *noecho* mode by this program so that the output can be continuous. Thus, what you type does show on your terminal, except for the / and ! commands.

If the standard output is not a teletype, then **more** acts just like **cat**(1), except that a header is printed before each file (if there is more than one).

A sample usage of **more** in previewing *nroff* output would be

```
nroff -ms +2 doc.n | more -s
```

NOTES

These commands are based on the Fourth Berkeley Software Distribution under license from The Regents of the University of California. Most Berkeley commands, including these, are located in /usr/ucb. Hence, to use this command, either enter the full path (/usr/ucb/more) or include /usr/ucb in your PATH environment variable.

FILES

/usr/lib/terminfo Terminal data base

SEE ALSO

csh(1), **script**(1), **sh**(1), **pg**(1)

NAME

syslog, openlog, closelog, setlogmask – Controls system log

SYNOPSIS

#include <syslog.h>

setloghost(*host***)**
char ***host**;

setlogport(*port***)**

openlog(*ident, logopt, facility***)**
char ***ident**;

syslog(*priority, message, parameters ...* **)**
char ***message**;

closelog( )

setlogmask(*maskpri***)**

DESCRIPTION

The **syslog** routine arranges to write *message* onto the system log maintained by **syslogd**(1M). The message is tagged with *priority*. The message looks like a **printf**(3S) string except that **%m** is replaced by the current error message (collected from **errno**). A trailing new-line character is added if needed. This message is read by **syslogd**(1M). It is then written to the system console or log files, or it is forwarded to **syslogd** on another host as appropriate.

Priorities are encoded as a *facility* and a *level*. The facility describes the part of the system generating the message. The level is selected from the following ordered list:

Facility	Level
LOG_EMERG	A panic condition, which is normally broadcast to all users
LOG_ALERT	A condition that should be corrected immediately, such as a corrupted system database.
LOG_CRIT	Critical conditions; for example, hard device errors.
LOG_ERR	Errors
LOG_WARNING	Warning messages
LOG_NOTICE	Conditions that are not error conditions, but possibly should be specially handled.
LOG_INFO	Informational messages
LOG_DEBUG	Messages that contain information normally useful only when debugging a program.

If **syslog** cannot pass the message to **syslogd**, it attempts to write the message on **/dev/console** if the LOG_CONS option is set (see below).

If the log messages are to be sent to another system on the network, call **setloghost(***host***)**. If the system log daemon, on either the local or remote host, is waiting on a port other than the one specified in **/etc/services**, call **setlogport(***port***)**. Both **setloghost(***host***)** and **setlogport(***port***)** must be called before you call either **openlog** or **syslog**.

If special processing is needed, **openlog** can be called to initialize the log file. Parameter *ident* is a string that is prepended to every message. Parameter *logopt* is a bit field indicating logging options. Current values for *logopt* are as follows:

Value	Description
LOG_PID	Logs the process ID with each message; this is useful for identifying instantiations of daemons.
LOG_CONS	If **openlog** cannot send the message to **syslog**, LOG_CONS forces messages to be written to the console. This option is safe to use in daemon processes that have no controlling terminal because **syslog** forks before opening the console.
LOG_NDELAY	Opens the connection to **syslogd** immediately. Normally the open is delayed until the first message is logged. This is useful for programs that need to manage the order in which file descriptors are allocated.
LOG_NOWAIT	Indicates not to wait for child processes forked to log messages on the console. This option should be used by processes that enable notification of child termination using **SIGCHLD**, because **syslog** can otherwise block waiting for a child process whose exit status has already been collected.

The *facility* parameter encodes a default facility to be assigned to all messages that do not have an explicit facility encoded, as follows:

Facility	Description
LOG_KERN	Messages generated by the kernel. These messages cannot be generated by any user processes.
LOG_USER	Messages generated by random user processes. If none is specified, this is the default facility identifier.
LOG_MAIL	Messages generated by the mail system.
LOG_DAEMON	Messages generated by system daemons, such as **ftpd**(1M) or **routed**(1M).
LOG_AUTH	Messages generated by the authorization system, such as **login**(1), **su**(1), or **getty**(1M).
LOG_LOCAL0	Reserved for local use. (Similarly for LOG_LOCAL1 through LOG_LOCAL7.)

The **closelog** routine can be used to close the log file.

The **setlogmask** routine sets the log priority mask to *maskpri* and returns the previous mask. Calls to **syslog** with a priority not set in *maskpri* are rejected. The mask for an individual priority *pri* is calculated by macro LOG_MASK(*pri*); the mask for all priorities up to and including *toppri* is given by macro LOG_UPTO(*toppri*). The default allows all priorities to be logged.

NOTES

To use these routines, option **–lnet** must be specified on the **cc**(1) or **ld**(1) command line.

EXAMPLES

```
syslog(LOG_ALERT, "who: internal error 23")

openlog("ftpd", LOG_PID, LOG_DAEMON)
setlogmask(LOG_UPTO(LOG_ERR))
syslog(LOG_INFO, "Connection from host %d", CallingHost)

setloghost("secure_log_system")
openlog("login",LOG_PID,LOG_AUTH)
syslog(LOG_WARNING, "%s logged in with a null password.",UserName)

syslog(LOG_INFO|LOG_LOCAL2, "foobar error: %m")
```

SYSLOG(3BSD)

SEE ALSO

logger(1)
syslogd(1M)
log(4D)

NAME

syslogd – Logs system messages

SYNOPSIS

/etc/syslogd [**–f***configfile*] [**–m***markinterval*] [**–p***path*] [**–P***port*] [**–d**]

DESCRIPTION

The **syslogd** command reads and logs messages into a set of files described by the configuration file **/etc/syslog.conf**. Each message is one line. A message can contain a priority code, marked by a number in angle braces at the beginning of the line. Priorities are defined in the include file **sys/syslog.h**. **syslogd** reads from the named pipe **/dev/log**, from an Internet domain socket specified in **/etc/services** (see **services**(4N)), and from the special device **/dev/klog** to read kernel messages; see **klog**(4D).

The shell script **newsys**(1M) starts **syslogd**. However, before it starts the daemon, **newsys** renames the log files so that they are of zero length. This is because if the daemon were simply started the files being renamed, they would eventually grow to fill the file systems on which they reside (**syslogd** never truncates the log files).

newsys also saves the ten most recent copies of the files and deletes the any older copies. Once this is done for all log files, **newsys** starts the **syslogd** daemon.

The following options are available for use with **syslogd**:

–f*configfile* Specifies an alternate configuration file

–m*markinterval*
Selects the number of minutes between mark messages

–p*path* Specifies a named pipe other than **/dev/log**

–P*port* Specifies an alternate port. This is a handy feature if more than one **syslogd** is to run at the same time.

–d Turns on debugging

syslogd configures when it starts and whenever it receives a hangup signal. Lines in the configuration file have a *selector* to determine the message priorities to which the line applies and an *action*. The *action* field is separated from the selector by one or more tabs.

Selectors are semicolon-separated lists of priority specifiers. Each priority has a *facility* describing the part of the system that generated the message, a dot, and a *level* indicating the severity of the message. Symbolic names may be used. An asterisk selects all facilities. All messages of the specified level or higher (greater severity) are selected. More than one facility may be selected using commas to separate them. Known facilities and levels recognized by **syslogd** are those listed in **syslog**(3C) without the prefix "LOG_". The additional facility "mark" has a message at priority LOG_INFO sent to it every 20 minutes (this may be changed with the **–m** option). The "mark" facility is not enabled by a facility field containing an asterisk. The level "none" may be used to disable a particular facility. The second part of each line describes where the message is to be logged if this line is selected. There are four forms, as follows:

A file name (beginning with a leading slash);
the file is opened in append mode.

A host name preceeded by an at sign (@);
selected messages are forwarded to the syslog daemon on the named host.

A comma-separated list of users;
selected messages are written to those users if they are logged in.

An asterisk;
selected messages are written to all logged-in users.

Blank lines and lines beginning with '#' are ignored.

syslogd creates the file **/etc/syslog.pid**, if possible, containing a single line with its process ID. This can be used to kill or reconfigure **syslogd**.

To bring **syslogd** down, send it a terminate signal (for example, **kill `cat /etc/syslog.pid`**).

EXAMPLES

The following line selects all facilities at the **emerg** level and the **mail** and **daemon** facilities at the **crit** level: ***.emerg;mail,daemon.crit**

The following line sends all messages except mail messages to the selected file: ***.debug;mail.none**

The following configuration file logs all kernel messages and 20 minute marks onto the system console, all notice (or higher) level messages and all mail system messages except debug messages into the file **/usr/spool/adm/syslog**, and all critical messages into **/usr/adm/critical**; kernel messages of error severity or higher are forwarded to ucbarpa; all users will be informed of any emergency messages, the users **eric** and **kridle** are informed of any alert messages, and the user **ralph** is informed of any alert message, or any warning message (or higher) from the authorization system:

kern,mark.debug	**/dev/console**
*.notice;mail.info	/usr/spool/adm/syslog
*.crit	/usr/adm/critical
kern.err	@ucbarpa
*.emerg	*
*.alert	eric,kridle
*.alert;auth.warning	ralph

FILES

/etc/syslog.conf	Configuration file
/etc/syslog.pid	Process ID
/dev/log	Name of the named pipe
/dev/klog	Kernel log device

SEE ALSO

newsys(1M)
logger(1)
syslog(3BSD)
klog(4D)

NAME

tset, reset – Terminal-dependent initialization

SYNOPSIS

tset [**–e***c*] [**–k***c*] [**–n**] [**–I**] [**–Q**] [**–m** [*ident*] [*test baudrate*]:*type*] [*type*] [**–**]

reset [**–e***c*] [**–k***c*] [**–n**] [**–I**] [**–Q**] [**–m** [*ident*] [*test baudrate*]:*type*] [*type*] [**–**]

DESCRIPTION

The **tset** command sets up your terminal when you first log in to a UNICOS system. It performs terminal dependent processing such as setting erase and kill characters, setting or resetting delays, and sending any sequences needed to properly initialize the terminal. It first determines the *type* of terminal involved, and then performs necessary initializations and mode settings. Type names for terminals can be found in the **terminfo**(4F) database. If a port is not wired permanently to a specific terminal (not hardwired), it is given an appropriate generic identifier such as *dialup*.

Options:

–e*c* Sets the erase character to the named character *c* on all terminals. The default is the backspace character on the terminal, usually CONTROL-H. The character *c* can either be typed directly, or entered using the hat notation used here.

–k*c* Sets the line kill character to the named character c on all terminals; *c* defaults to CONTROL-U. The kill character is left alone if **–k** is not specified. The hat notation can also be used for this option.

–n On systems with the Berkeley 4BSD tty driver, **–n** specifies that the new tty driver modes should be initialized for this terminal. For a CRT, the CRTERASE and CRTKILL modes are set only if the baud rate is 1200 or greater. See **tty**(4) for more details.

–I Suppresses transmitting terminal initialization strings.

–Q Suppresses printing the "Erase set to" and "Kill set to" messages.

– Specifies the name of the terminal finally decided upon to be output on the standard output. This is intended to be captured by the shell and placed in the environment variable TERM.

If you do not specify any arguments, **tset** simply reads the terminal type out of the environment variable TERM and reinitializes the terminal.

When used in a startup procedure (**.profile** for **sh**(1) users or **.login** for **csh**(1) users) it is desirable to provide information about the type of terminal you usually use on ports that are not hardwired. These ports are identified as *dialup*, *plugboard*, or *arpanet*, and so on. To specify what terminal type you usually use on these ports, the **–m** (map) option is followed by the appropriate port type identifier (*ident*), an optional baud rate specification (*baudrate*), and the terminal type (*type*). (The effect is to "map" from some conditions to a terminal type, that is, to tell **tset** "If I'm on this kind of port, assume that I'm on that kind of terminal".) If you specify more than one mapping, the first applicable mapping prevails. A missing port type identifier matches all identifiers. Any of the alternate generic names given in *terminfo* can be used for the identifier.

A *baudrate* is specified as with **stty**(1), and is compared with the speed of the diagnostic output (which should be the control terminal). The baud rate *test* may be any combination of: >, @, <, and !; @ means "at" and ! inverts the sense of the test. To avoid problems with metacharacters, it is best to place the entire argument to **–m** within "'" characters; **csh**(1) users must also put a "\" before any "!" used here.

Thus,

```
tset -m 'dialup>300:adm3a' -m dialup:dw2 -m 'plugboard:?adm3a'
```

causes the terminal type to be set to an *adm3a* if the port in use is a dialup at a speed greater than 300 baud; to a dw2 if the port is (otherwise) a dialup (that is, at 300 baud or less). If the *type* finally determined by **tset** begins with a question mark, you are asked if you really want that type. A null response

means you do want that type; otherwise, you can enter another type, which is then used instead. Thus, in the above case, you will be queried on a plugboard port as to whether you are actually using an **adm3a.**

If no mapping applies and you specify a final *type* option (not preceded by a **–m**) on the command line, that type is used; otherwise, the identifier found in the environment is taken to be the terminal type. This should always be the case for hardwired ports.

It is usually desirable to return the terminal type, as finally determined by **tset**, and information about the terminal's capabilities to a shell's environment. You can do this by using the **–** option. For example, using the Bourne shell, **sh**(1):

export TERM; TERM=`tset – *options...*`

or using the C shell, **csh**(1):

setenv TERM `tset – *options...*`

With **csh**, it is convenient to make an alias in your **.cshrc:**

alias tset 'setenv TERM `/usr/ucb/tset – \!*`'

This alias allows the command

tset 2621

to be invoked at any time from your login csh. **Note to Bourne Shell users:** It is **not** possible to get this aliasing effect with a shell procedure because shell procedures cannot set the environment of their parent.

These commands cause **tset** to place the name of your terminal in the variable TERM in the environment.

Once the terminal type is known, **tset** engages in terminal driver mode setting. This normally involves sending an initialization sequence to the terminal, setting the single character erase (and optionally the line-kill (full line erase)) characters, and setting special character delays. Tab and new-line expansion are turned off during transmission of the terminal initialization sequence.

On terminals that can backspace but not overstrike (such as a CRT), and when the erase character is the default erase character ('#' on standard systems), the erase character is changed to BACKSPACE (CONTROL-H).

If **tset** is invoked as **reset**, it sets cooked and echo modes, turns off cbreak and raw modes, turns on newline translation, and restores special characters to a sensible state before any terminal-dependent processing is done. Any special character found to be NULL or "–1" is reset to its default value.

This is most useful after a program dies leaving a terminal in an unusual state. You may have to type "<LF>reset<LF>" to get it to work since <CR> may not work in this state. Often none of this will echo.

NOTES

These commands are based on the Fourth Berkeley Software Distribution under license from The Regents of the University of California. Most Berkeley commands, including these, are located in **/usr/ucb**. Hence, to use this command, either enter the full path (**/usr/ucb/tset**) or include **/usr/ucb** in your **PATH** environment variable.

EXAMPLES

These examples all assume the Bourne shell and use the – option. If you use **csh**, use one of the variations described above. Note that a typical use of **tset** in a **.profile** or **.login** also use the **–e** and **–k** options, and often the **–n** or **–Q** options as well. (NOTE: some of the examples given here appear to take up more than one line, for text processing reasons. When you type in real **tset** commands, you must enter them entirely on one line.)

At the moment, you are on a 2621. This is suitable for typing by hand but not for a **.profile**, unless you are *always* on a 2621.

```
export TERM; TERM=`tset – 2621`
```

You have an h1,9 at home that you dial up on, but your office terminal is hardwired and known in **/etc/ttytype**.

```
export TERM; TERM=`tset – –m dialup:h19`
```

You have a switch that connects everything to everything, making it nearly impossible to indicate on which port you are coming. You use a VT100 in your office at 9600 baud, and dial up to switch ports at 1200 baud from home on a 2621. Sometimes you use someone elses terminal at work, so you want it to ask you to make sure what terminal type you have at high speeds, but at 1200 baud you are always on a 2621. Note the placement of the question mark, and the quotes to protect the greater than and question mark from interpretation by the shell.

```
export TERM; TERM=`tset – –m 'switch>1200:?vt100' –m 'switch<=1200:2621'
```

All of the above entries will fall back on the terminal type specified in the environment if none of the conditions hold. The following entry is appropriate if you always dial up, always at the same baud rate, on many different kinds of terminals. Your most common terminal is an adm3a. It always asks you what kind of terminal you are on, defaulting to adm3a.

```
export TERM; TERM=`tset – ?adm3a`
```

If the environment is not properly initialized and you want to key entirely on the baud rate, the following can be used:

```
export TERM; TERM=`tset – –m '>1200:vt100' 2621`
```

Here is a fancy example to illustrate the power of **tset** and to hopelessly confuse anyone who has made it this far. You dial up at 1200 baud or less on a concept100, sometimes over switch ports and sometimes over regular dialups. You use various terminals at speeds higher than 1200 over switch ports, most often the terminal in your office, which is a vt100. However, sometimes you log in from the university you used to go to, over the ARPANET; in this case you are on an ALTO emulating a dm2500. You also often log in on various hardwired ports, such as the console, all of which are properly entered in the environment. You want your erase character set to control H, your kill character set to control U, and don't want **tset** to print the "Erase set to Backspace, Kill set to Control U" message.

```
export TERM; TERM=`tset –e –k^U –Q – –m 'switch<=1200:concept100' –m 'switch:?vt100'
–m dialup:concept100 –m arpanet:dm2500`
```

FILES

/usr/lib/terminfo	Terminal capability database

SEE ALSO

csh(1), **sh**(1), **stty**(1)
terminfo(4F)
Defining and Compiling **terminfo** Definitions for the Operating System, publication SN-2067.

C. FEI-3 INSTALLATION AND TEST

This appendix provides information needed to install the FEI-3 interface on the Motorola workstation. It also provides the man pages for scytest and scyadmin commands used to test the driver.

C.1 FEI-3 INSTALLATION

CRAY FEI-3 Interface(8)

NAME

Cray FEI-3 Interface Installation Technical Note

DESCRIPTION

This paper provides a synopsis of the steps required to install a Cray FEI-3 (FEI-3M, FEI-3S, ...) interface and the supporting software. Please refer to the documents listed under "SEE ALSO" for more details.

INSTALLATION - CRAY X/MP IOS Kernel

Change the following parameters in $APTEXT if necessary:

symbol	default	description
I@COV0	6	First FEI-3 ordinal
I@NVME	1	Number of FEI-3 channels
I@NVID	1	Number of station IDs allowed to log on to COS/SCP over a single FEI-3 channel pair
MAXSCER	256	Maximum consecutive errors before the driver terminates
NSCBFC	32	Maximum MOS buffers for data
VMESFRST	0	First logical path defined for use
VMESLAST	15	Last logical path defined for use
VMESMXIN	4	Maximum input messages to be buffered
VMESRTO	0600	Physical read timeout in 0.1 seconds
VMESWTO	0100	Physical write timeout in 0.1 seconds

Change the following in deck AMAP (example shown is for two FEI3 interfaces; for one FEI3 delete the third and fourth CHANNEL macros):

```
*D AMAP.279,AMAP.510
        .
        .
        .
*
*       MIOP Channel Configuration
*
AACH    *
        .
        .
        .
       CHANNEL  (30),TY=VM,ORD=I@COV0    .FEI-3 (IN)
       CHANNEL  (31),TY=VM,ORD=I@COV0    .FEI-3 (OUT)
       CHANNEL  (32),TY=VM,ORD=I@COV0+1  .FEI-3 (IN)
       CHANNEL  (33),TY=VM,ORD=I@COV0+1  .FEI-3 (OUT)
        .
        .
        .
```

See VMEbus Driver Technical Note, CRI Publication SN-0244 for more information.

INSTALLATION - CRAY X/MP UNICOS Kernel

sys/param.h: increase NLCHAN if necessary; the default is 6.
Add a "hyinit" macro for the FEI-3 interface to the hyaddr table in uts/cf/conf.c

```
/*
 *      Set up HYPERchannel configuration.
 *
 *              hyinit( unit, chan ),
 *
 *          unit = NSC adaptor unit number (hex) if NSC
 *          unit = unique arbitrary number (hex) if FEI
 *          unit = unique number (hex) if FEI-3S or 3M; must match hyroute table entry
 *
 *          chan = IOP channel number (octal)
 *
 *        for UNICOS 4.0 and above:
 *
 *              hyinit( unit, chan, type ),
 *
 *          type = device type one of HYNSC, HYFEI, or HYVME
 */


struct hyad hyaddr[] = {
      hyinit( 0xC7, 036),    /* NSC,    unit 0xC6, IOP channel 036 */
      hyinit( 0xC3, 034),    /* NSC,    unit 0xC3, IOP channel 034 */
      hyinit( 0x00, 022),    /* FEI,    IOP channel 022 */
      hyinit( 0x01, 024),    /* FEI,    IOP channel 024 */
      hyinit( 0x02, 030),    /* FEI-3S, IOP channel 030 */ };

/*      to add a second FEI-3 add a line such as the following:      */

      hyinit( 0x03, 032),    /* FEI-3M, IOP channel 032 */
```

Note that the FEI-3S interface will be designated hy4 with the above table. The minor number for the FEI-3 entries will be the index in the above table times 32; hence, the entries start at 0 for hy0, 32 for hy1, 64 for hy2, 96 for hy3, 128 for hy4, 160 for hy5, 192 for hy6, and 224 for hy7. There is a maximum of NLCHAN (6) subchannels for the FEI-3 driver on the X/MP and a maximum of 8 interfaces (HYPERchannel, FEI, and FEI3.) The "unit" parameter must be the same as the first two hex digits of the address in the hyroute file (see below.)

- Determine the host names and Internet addresses you will be using. See *NAMES AND ADDRESSES* below.
- Generate a new kernel. For details of this procedure, refer to the UNICOS SIB.
- Boot the new kernel.

CRAY FEI-3 Interface(8)

Make the device entries for the FEI-3 interface:
As noted above, for FEI-3 "hy4" the minor numbers would start at 128.

```
/etc/mknod /dev/vme00 c 4 128
/etc/mknod /dev/vme01 c 4 129
/etc/mknod /dev/vme02 c 4 130
/etc/mknod /dev/vme03 c 4 131
/etc/mknod /dev/vme04 c 4 132
/etc/mknod /dev/vme05 c 4 133
```

You can check the parameters by typing "ls -l /dev/vme*".

• Enter "VME" (or "VME *ch*" where *ch* is the channel pair no.) at the IOS console. The default is the first FEI-3 channel.

```
example:    VME 30
            VME 26
```

• Create a routing file ("hycf.fei3") for hyroute and run hyroute.

The hycf.fei3 file should look like this:

```
#
direct  sn228-vme      0205    ff00    0;
direct  binkley-vme    0201    ff00    0;
```

The first two hex digits ("02" in this case) must match the first parameter in the hyinit table. The last two hex digits must match the last octet of the Internet addresses on the front end machine.

You can test before you modify rc.boot by entering the *hyroute* and *ifconfig* manually:

```
/etc/hyroute hy4 -s hycf.fei3
/etc/ifconfig hy4 'hostname'-vme -trailers up
```

For the second FEI-3 you would have an additional hyroute file, hycf.fei3m:

```
#
direct  sn228-fei3m    0305    ff00    0;
direct  motorola-fei3  0301    ff00    0;
```

and you would enter

```
/etc/hyroute hy5 -s hycf.fei3m
/etc/ifconfig hy5 'hostname'-fei3m -trailers up
```

OPERATIONAL NOTE - CRAY X/MP

If you do not have 2IJ module FCO # 589 installed and you must disconnect the channel cables to the front end machine containing the FEI-3 interface, or power off the front end machine, you *MUST* first enter "VMEND *ch*" or "VMEND" at the IOS console. Failing to do this will require an IOP-0 master clear before the channel is usable again. The symptom is that the CRAY can write to the front end machine but the front end machine cannot write to the CRAY.

INSTALLATION - CRAY 2 UNICOS Kernel

If you are adding an additional low speed channel for the FEI-3 interface, you MUST increase LSPMAX in sys/param.h and may need to change HYMAX. Note that these must be exactly right or the foreground processor will get very confused!

Next, make device entries for the interface (see lsp(4D) man page):

```
mknod name c major minor dtype index proto

dtype = 1 for 50 Mbit/sec;
index is the Foreground Processor index of the channel
proto = 2 for ELCP protocol

/etc/mknod /dev/vme00 c 12 0 1 0 2
/etc/mknod /dev/vme01 c 12 1 1 0 2
/etc/mknod /dev/vme02 c 12 2 1 0 2
/etc/mknod /dev/vme03 c 12 3 1 0 2
/etc/mknod /dev/vme04 c 12 4 1 0 2
/etc/mknod /dev/vme05 c 12 5 1 0 2
```

You can check the parameters by typing "file /dev/vme*".

• Create a routing file ("hycf.xxx") for hyroute and run hyroute.

The hycf.fei3 file should look like this:

```
#
direct  cray2-fei3     0205    ff00    0;
direct  sunxx-fei3     0201    ff00    0;
```

The first two hex digits ("02" in this case) are arbitrary.
The last two hex digits must match the last octet of the Internet addresses on the front end machine.

You can test before you modify rc.boot by entering the *hyroute* and *ifconfig* manually:
The interface name will be "vme" and an index equal to NHY plus the FEI-3 ordinal; for example, with two HYPERchannel connections and two FEI-3 connections, the names would be hy0, hy1, vme2, and vme3.

```
/etc/hyroute vme2 -s hycf.xxx
/etc/ifconfig vme2 'hostname'-vme -trailers up
```

CHECKOUT and TROUBLESHOOTING

• Connect the cables coming from the FEI-3 boards together with a loopback cable (part no. 0220-3405.) Now run cytest in loopback mode for 100 passes on the front end machine.

```
scytest -d 100
```

If scytest fails, check the following:

1) Are the switches set correctly on the FEI-3 interface boards? The address switches must match the "cy" synopsis line (Sun) or the line(s) in dfile (system V) you used to build the front end machine kernel.

CRAY FEI-3 Interface(8)

2) Did you remove the BG3 and IACK jumpers from the rear of the backplane for the slot containing the control board (but not the buffer board?) This is BG2 and IACK for Motorola machines.

3) Are the cables connected correctly (output on the top, input on the bottom; pin 1 end (brown-red-orange sequence) towards the bottom)?

• Remove the loopback cable and connect the cables to the CRAY channel. Type "VME" at the IOP console if the CRAY is an X/MP. If you are running COS on the Cray, you can run "iosping" on the Motorola to check the connection. Now run scytest between the front end machine and the CRAY if you are running UNICOS. Note that you must use the -X parameter on both ends if the CRAY is an X/MP.

sun: scytest -Xt 100 r
cray: scytest -Xt 100 w

and the reverse:

sun: scytest -Xt 100 w
cray: scytest -Xt 100 r

If scytest fails, check the following:

1) Are the switches set correctly on the FEI-3 interface boards?

2a) (X/MP) Are the CRAY channel boards set for 50 Mbit (6 Mbyte) mode?

2b) (CRAY 2) Are the /dev/vme*nn* entries made for 50 Mbit (6 Mbyte) ELSP mode?

3) Is the CRAY UNICOS kernel configured correctly? Are the /dev/vme*nn* entries correct? Type "ls -l /dev/vme*" on the X/MP or "file /dev/vme*" on the Cray 2 to check them.

3a) (X/MP only) Is the CRAY IOS kernel configured correctly? Type "CONFIG" at the MIOP console to check - your channel pair should say "VME".

3b) (X/MP only) Did you type "VME *ch*" at the IOP 0 console?

• Run *ifconfig* on the front end machine system and *hyroute* and *ifconfig* on the CRAY system and try to *ping* the CRAY system from the front end machine system.
• If the previous step was successful, try to rlogin to UNICOS from the front end machine system.

NAMES AND ADDRESSES

Each machine on the Cray FEI-3 channel will need a host name and an Internet address. You may assign a class A, B, or C Internet address to each machine on the Cray FEI-3 channel.

Cray side:

The program hyroute allows the driver to map between Internet addresses and logical paths across the FEI-3. The address in the hycf.xxx file used by hyroute consists of four hex digits. On the X/MP, the upper two digits (octet) must match the "unit" in the hyaddr table. The lower two digits are the logical path and must match the lower two hex digits (octet) of the front end machine Internet addresses.

front end machine side:

The front end machine software does not yet support hyroute, so the lower two hex digits of the Internet address must match the lower two hex digits of the address in the Cray hyroute file.

The lower five bits of the lower octet are the "logical path" that allows multiplexing of multiple protocols over the Cray FEI-3 channel. The upper bit of this octet is used as a flag by the Sun to allow local loopback for testing.

Here is an example, a fragment of /etc/hosts:

```
# Ethernet addresses
192.9.99.24     binkley binkley.cray.com
192.9.9.50      binkley-2
192.9.9.26      mu
192.9.14.50     binkley-3
192.9.14.22     fuji
# Cray FEI-3 channel addresses
88.0.0x02.1     binkley binkley-vme
88.0.0x02.0x81  binkley-vme-loop
88.0.0x02.2     sn228-uscp
88.0.0x02.5     sn228 sn228-vme
```

The corresponding entries in /etc/networks:

```
opusnet         192.9.9
sierranet       192.9.14
mh1440net       192.9.99
crayfei3        88
```

In the example above, we have five machines on three Ethernets. Two Sun-3 workstations, *mu* and *fuji*, are on Ethernets. A CRAY X-MP, *sn228*, is connected to a Cray FEI-3 board set in *binkley*. *binkley* has three Ethernet connections (binkley, binkley-2, and binkley-3) and acts as a gateway between these Ethernets and the Cray. Note that to run more than two Ethernets on a Sun you must have a Sun source license and rebuild your kernel using the source for the ethernet driver (the released binary is built for two interfaces at most.)

Note that the machines that are on both networks use aliases. This is useful for processes (such as the CRAY UNIX station) that know how to search the hosts file for machines on a given network.

Don't forget to modify /etc/rc.boot or its equivalent on the front end (gateway) machine to start the FEI-3 interface at boot time:

```
/etc/ifconfig cy0 'hostname'-vme up
```

For Motorolas, see the ipcy man page for information on tpid.conf.

ROUTING

In order for TCP/IP traffic to make use of the FEI-3 interface, routes must be added on the gateway machine (the one containing the FEI-3 boards,) as well as the Cray and other machines on the network.

On the CRAY (sn228) add a route to each (Ethernet in this case) network on the gateway machine:

```
/etc/route add opusnet binkley-vme 1
/etc/route add sierranet binkley-vme 1
```

On the gateway (binkley):

```
/etc/route add sn228 sn228-vme 1
```

On the other suns (mu, fuji) add a route from the FEI-3 network to the gateway:

```
on "mu"         /etc/route add sn228 binkley-2 1
on "fuji"       /etc/route add sn228 binkley-3 1
```

SUBNET SUPPORT - ON THE CRAY

UNICOS release 4.0 supports a limited subnet routing capability. Subnet masks must be 8 bits larger than the size of the network field; in other words you can use class A addresses with a 16 bit ("class B") mask, or class B addresses with a 24 bit ("class C") mask. You can't subnet class C addresses under this release.

To use subnets you must specify the -S option on hyroute. This indicates that this interface's network is using (limited) subnets. You may also specify the "local" option on the ifconfig command (the "-S" option on the hyroute command will have set this flag already.) The high bit of the subnet byte must be set; i.e., valid subnet bytes are 128 through 254 decimal.

See the network and hosts files below. The hycf.4 file looks like this:
direct sunrise 0201 ff00 0 4144;
direct crayxmp 0205 ff00 0 4144;

Now enter:
/etc/hyroute hy4 -S -s hycf.4
/etc/ifconfig hy4 crayxmp local

SUBNET SUPPORT - FRONT END

Systems on the same network must also support subnets. This support is in BSD 4.3 and systems derived from it (i.e. SunOS 3.5 etc.) For example, to use a class B address with subnets, the network mask would be 255.255.255. Note that the subnet number must be between 128 and 254.

oursubnet	128.162.128
sunrise	128.162.128.1
crayxmp	128.162.128.5

Note that since the if_cy driver on the Sun doesn't support hyroute as yet, the host numbers must match the numbers in the CRAY hycf.x file. Since the FEI-3 is the only thing on this subnet, this isn't a problem.

You must specify the subnet mask on the ifconfig command.

ifconfig cy0 sunrise -n 255.255.255 up
NOTE: SunOS 3.3 and above support subnets with the restriction that all interfaces with non-default subnet masks must be on the same IP network (but may be on different subnets.) In other words, you cannot have more than one subnetted network interfaced to any machine.
Motorola V.3.5 does not support subnets.

SEE ALSO

Sun - cy(4), scytest(4), scyadmin(4), if_cy(4)
Motorola - scy(4), scymux(4), ipcy(4), scytest(4), scyadmin(4)
X/MP - vme(4d), mknod(1M)
CRAY 2 - lsp(4d), mknod(1M)
ftp(1C), telnet(1C), ifconfig(8C), hyroute()
station(4) for a discussion of the Cray UNIX station
System Administration for the Sun Workstation, SMI publication 800-1150-01
CRAY VMEbus Interface Reference Manual, CRI Publication HR-3022
VMEbus Driver Technical Note, CRI Publication SN-0244

BUGS

ipcy.c on the Motorola does not support hyroute.

C.2 TEST MAN PAGES

This subsection provides hardcopies of the **scytest** and **scyadmin** commands.

NAME

scytest – VME bus to Cray Low Speed Channel Interface Tests

SYNOPSIS

scytest [-z driver][-T to][-F from] [-H HYparms] [-DX] -ldwrte [parms]

DESCRIPTION

The Low Speed Channel Interface provides a hardware connection between the VME bus and the Cray Low Speed Channel. The *cy* driver provides the software interface to that hardware. Scytest provides commands for the system administrator and maintenance personnel to test the *cy* driver and Cray Interface hardware, and get performance statistics. There are loopback tests and two machine tests.

Scytest can be used for more than testing just the *cy* driver, however. It can be used to test HYPERchannel drivers (see the -H option), Cray1-XMP/IOS channels in loopback, or Cray2 channels cross cabled.

Note only one command is allowed at a time.

OPTIONS

The different tests in *scytest* take different, but similar options, each is described below. Loopcount is the number of reads/writes to do. Writecount and readcount must be divisible by 8. Passcount is for the automatic tests, –d and –t, which generate semi-random values for loopcount, writecount and readcount.

–l [loopcount] [writecount] [readcount] [s] [w]
This option performs a loopback test of the Cray Interface. The Interface must of course be cabled in a loopback configuration. s is for sleep one second each pass, and w is for write first.

–d [startpass] [endpass] [governor]
This option performs an automatic loopback test, displaying the pass number, loopcount, writecount and readcount for each pass. The –d option can be started at any arbitrary passcount, or run to any arbitrary ending passcount. If *startpass* is not specified, the next parameter is taken to be *endpass*. The *governor* is a mask used to govern how large/small the readcount/writecount sizes will be. Example usage: scytest –d 0 99 0xfff8. Two notes: a hex mask must be preceded by 0x (octal by 0), and *startpass* and *endpass* must be specified before the *governor*. The default *governor* is 0xf8.

–w [loopcount] [writecount]
This option performs a one way write, used in conjunction with the –r test (read) either in a loopback test or a two machine test.

–r [loopcount] [readcount]
This option performs a one way read, used in conjunction with the –w test (write) either in a loopback test or a two machine test.

–t passcount rw
This option performs an automatic two machine test, with a cooperating process in the second machine. Both ends must specify the same passcount. One end must specify "r" for read, the other must specify "w" for write.

–e [loopcount] [readcount]
This option performs an echo test, with a cooperating –l scytest in a second machine. Both ends must specify the same passcount.

In addition to the options for the actual tests, there are several options which can be used to modify the behavior of scytest. These options must be specified before the final test parameters.

-z driver
This option allows the specification of an alternate driver name. This can be useful if the default name is already in use, or if a different interface must be specified.

-T to
This option allows the specification of an alternate TO address. This is useful if the destination scytest (on a -w test) is not the same logical channel as the source scytest. It can be specified in hex or octal by preceding the value with 0x or 0. This option should not be used with the -l option.

-F from
This option allows the specification of an alternate FROM address. The *cy* driver fills in the FROM address, so this option is not useful for *cy*, however, it may be useful with the -H option. It can be specified in hex or octal by preceding the value with 0x or 0.

-H HYparms
This option allows specification of HYPERchannel parameters in bytes 8 and 9 of the HYPERchannel message proper. It can be specified in hex or octal by preceding the value with 0x or 0.

-D
This option causes scytest to run in debug mode. Scytest will print out more information relevant to the particular test type. In loopback, it will compare received data with transmitted data, printing the first instance of any discrepancy.

-X
This option is used when testing with an Cray1-XMP. Because the Cray1-XMP system returns an error to a user process if the data count received is greater than requested, scytest provides an option to insure the automatic two machine test does not generate this condition. This option is only applicable with the -t test.

HARDWARE INSTALLATION

Installing the Cray-VME Interface hardware is covered in other documentation, see the list of publications in cy(4) for details. It is worth noting that to run the Interface loopback tests, the Interface must be configured as a 50 Mbit channel master. In binary, without explanation, the switch settings would be:

```
control board sw1    1110111x    /* 1 is open, off, active */
control board sw2    10xxx000    /* x is unused */
control board sw3    1110000x    /* base address - could vary */
```

Also, for loopback tests a special Cray channel loopback cable must be used to connect the Interface output to the Interface intput

SOFTWARE INSTALLATION

Scytest and scyadmin are installed in the *scyadmin* directory on the FEI3 system directly from the distribution tape. For the two machine tests, this directory must be copied to the Cray system somehow. Use an alternate media or the ft program in the demo directory. (Ft would still need to be installed on the Cray system first).

Scytest and scyadmin are built from the same makefile. Some parameters may need changing on a site dependent basis. There is a driver name, typically */dev/cy02*, though this can be overridden on the command line with the –z option. The TO and FROM fields, which can also be overridden on the command line, are used to declare the default values Comments in the makefile will explain what to do.

For two machine tests, such as –r, –w, and –t (and –e used with –l), the second machine can be a Cray2 or Cray-XMP. If so, the driver name will typically be */dev/vme02*. Also, regardless of the type of the second machine, TO addresses must be coordinated between both ends.

FILES

/usr/include/sys/cyioctl.h	ioctl header file
/dev/cyNN	the Cray Interface device driver

SEE ALSO

scyadmin(8), cy(4)

NAME

scyadmin – VME bus to Cray Low Speed Channel Interface Administrator Commands

SYNOPSIS

scyadmin [–z driver] [–tcphbdme]

DESCRIPTION

The Low Speed Channel Interface provides a hardware connection between the VME bus and the Cray Low Speed Channel. The *scy* driver provides the software interface to that hardware. Scyadmin provides commands for the system administrator and maintenance personnel to control the *scy* driver.

Note multiple commands may be specified at one time, for example, scyadmin –pt will print the hardware registers, then the trace buffer.

OPTIONS

–z driver This option allows the specification of an alternate driver name. This can be useful if the default name is already in use, or if a different interface must be specified.

–c causes *scyadmin* to print the *scy* driver control blocks in hex.

–t causes *scyadmin* to print the *scy* driver trace table in hex.

–p causes *scyadmin* to print the Cray Interface hardware registers in hex. The format of the output is described in cyioctl.h.

–h causes *scyadmin* to print the *scy* driver message header buffer pool in hex.

–b causes *scyadmin* to print the *scy* driver buffer pool in hex.

–d disables disconnect and master clear interrupts from the Cray Interface. This is useful when uncabling the Interface to prevent interrupt flood messages on the console.

–m Master clear the Cray Interface hardware. This also enables disconnect and master clear interrupts, which may have been disabled by the –d command.

–e *scyadmin* will sleep until an error is detected by the *scy* driver. This could be used in conjunction with another command, scyadmin –et would sleep till an error, then print the trace table.

FILES

cyh.h Driver header file which explains trace types and driver control blocks

utrace.sort
A handy reference for decoding the driver trace as dumped by the –u command.

/dev/cyNN
the Cray Interface device driver

INSTALLATION

Scyadmin and scytest are built from the same makefile. Some parameters may need changing on a site dependent basis. In particular, there is a driver name, typically */dev/cy02*, though this can be overridden on the command line with the –z option. Comments in the makefile will explain what to do.

SEE ALSO

scytest(8), cy(4)

BUGS

The *scy* driver does not issue wakeup for all errors it detects, so the –e option isn't very useful.

D. CRAY-2 USAGE

The Operator Workstation (OWS) communicates with CRAY-2 computer systems through TCP/IP protocol and the OWS Station Interface. When an OWS is connected to a CRAY-2 it has the same privileges and features as those associated with the UNIX station.

The **fsload** command is used to read an initial file system from tape and is described on the following page.

NAME

fsload - Read initial file system from tape

SYNOPSIS

fsload *pathname port tape*

DESCRIPTION

fsload reads the initial CRAY-2 UNICOS file system from the OWS 9-track tape drive and writes it to the interface for installation on the CRAY-2.

pathname	The logical path to the CRAY-2
port	The logical port that the CRAY-2 is reading.
tape	The path name for the magnetic tape.

SEE ALSO

UNICOS System Administrator's Guide for CRAY-2 Computer Systems, SG-2019

CRAY-2 Computer System Installation Bulletin

Motorola System Administrator's Reference Manual, MU43812SAR/D2

Motorola System Administrator's Guide (Appendix A), MU43813SAG/D2

The **cymux** man page described in subsection B.2 of this manual

E. IOS DEADSTART DIAGNOSTICS

There are tow IOS deadstart diagnostics, **dsdiag** and **cleario**. **dsdiag** performs a quick check of the IOS hardware and ensures basic IOS functionality. **cleario** clears all of the IOS, including Local Memory, Buffer Memory, and all the channels.

For more information on these diagnostics, refer to the CRAY Y-MP, CRAY X-MP, and CRAY-1 Computer Systems On-line Diagnostic Maintenance Manual, SMM-1012.†

The first step in deadstarting the mainframe is to have the OWS software up and running. For more information on deadstarting, refer to Appendix A, Booting The System. Next, the operators boot deadstart diagnostics into the IOS. In the past, operators were accustomed to pushing the Master Clear and Deadstart buttons on the IOS chassis to send the control signals that start the booting of the system. These chassis buttons are no longer used in this process. A similar set of control lines run from the MIOP directly into the OWS. Now the operators simply enter the **dboot** command from the OWS. **dboot** requires that the diagnostic filename be specified (for example, **dboot -k dsdiag**). When the workstation processes this command, it internally sends these deadstart control signals to the MIOP. See section 2 for a description of the **dboot** command syntax.

The control signals cause the low-speed interface to perform a read on the channel associated with the control lines. The control signals are accepted by the IOS if the switch for the channel is enabled. The **dboot** command causes the OWS to send a standalone program, the Loader, across the low-speed channel into MIOP Local Memory location zero. The first portion of this Loader file contains several parameters that the workstation sets for the IOS's use. These parameters are: the word size of the IOS diagnostic Kernel binary; the sector size of the IOS diagnostic overlay binary (all overlays concatenated into one file); the low-speed input channel number that connects the MIOP to the OWS; a flag indicating if IOS debug mode is to be entered; and the maximum transfer size, in sectors, that the OWS can handle. This last parameter is needed because the microcomputer system software currently is limited to transfers of less than 8 sectors. For transfer of files greater than the limit, the OWS and MIOP must break the file into multiple transfers of the limiting block size.

---

† This publication is Cray Proprietary. Dissemination of this documentation to non-CRI personnel requires approval of the appropriate vice preseident and that the recipient sign a nondisclosure agreement. Export of technical information in this category may require an export license.

The low-speed input channel number parameter is needed so that the channel connecting the MIOP to the OWS can change without having to rebuild the IOS.

The Loader will first read the diagnostic overlay file from the OWS by doing several inputs of the above noted maximum block size. As each block is read in, it is written to the overlay area of Buffer Memory. Next the diagnostic Kernel is read from the OWS into MIOP Local Memory, starting at location zero. The Loader then forces the MIOP to start executing at location zero. Up to this point, all transfers across the low-speed channel to and from the OWS have been in "raw" mode; that is, there has been no protocol involved, just portions of raw data going across the channel.

The diagnostic program will display messages as it tests the IOS hardware. Upon completion, the program will display a "DIAGNOSTICS COMPLETE" message. It then sends a V-packet to the IOS to signal that the diagnostics have completed, and the OWS prompt is returned.

READER'S COMMENT FORM

Operator Workstation (OWS) Guide SN-3030

Your reactions to this manual will help us provide you with better documentation. Please take a moment to check the spaces below, and use the blank space for additional comments.

1) Your experience with computers: ____ 0-1 year ____1-5 years ____5+ years
2) Your experience with Cray computer systems: ____0-1 year ____ 1-5 years ____5+ years
3) Your occupation: ____ computer programmer ____ non-computer professional
   ____ other (please specify): _____
4) How you used this manual: ____ in a class ____as a tutorial or introduction ____ as a reference guide
   ____ for troubleshooting

Using a scale from 1 (poor) to 10 (excellent), please rate this manual on the following criteria:

5) Accuracy _____
6) Completeness ____
7) Organization ____
8) Physical qualities (binding, printing) ____
9) Readability ____
10) Amount and quality of examples ____

Please use the space below, and an additional sheet if necessary, for your other comments about this manual. If you have discovered any inaccuracies or omissions, please give us the page number on which the problem occurred. We promise a quick reply to your comments and questions.

Name _____ Address _____
Title _____ City _____
Company State/Country
Telephone Zip Code
Today's Date _____

CUT ALONG THIS LINE

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO 6184 ST PAUL MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY RESEARCH, INC.

Attention: PUBLICATIONS
1345 Northland Drive
Mendota Heights, MN 55120

FOLD

STAPLE

READER'S COMMENT FORM

Operator Workstation (OWS) Guide SN-3030

Your reactions to this manual will help us provide you with better documentation. Please take a moment to check the spaces below, and use the blank space for additional comments.

1) Your experience with computers: ____ 0-1 year ____1-5 years ____5+ years
2) Your experience with Cray computer systems: ____0-1 year ____ 1-5 years ____5+ years
3) Your occupation: ____ computer programmer ____ non-computer professional
____ other (please specify): _____
4) How you used this manual: ____ in a class ____as a tutorial or introduction ____ as a reference guide
____ for troubleshooting

Using a scale from 1 (poor) to 10 (excellent), please rate this manual on the following criteria:

5) Accuracy _____
6) Completeness ____
7) Organization ____
8) Physical qualities (binding, printing) ____
9) Readability ____
10) Amount and quality of examples ____

Please use the space below, and an additional sheet if necessary, for your other comments about this manual. If you have discovered any inaccuracies or omissions, please give us the page number on which the problem occurred. We promise a quick reply to your comments and questions.

Name _____
Title _____
Company _____
Telephone _____
Today's Date _____

Address _____
City _____
State/ Country _____
Zip Code _____

CUT ALONG THIS LINE

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY RESEARCH, INC.

Attention: PUBLICATIONS
1345 Northland Drive
Mendota Heights, MN 55120

FOLD

STAPLE