Cray Research, Inc. Software Training

UNICOS OPERATOR TRAINING

TR-UOT

PUBLIC

September 1991
Revision A
UNICOS 6.1

UNICOS Operator Training

PUBLIC

Software Training Department

Cray Research, Inc.
655-A Lone Oak Drive
Eagan, MN 55121

September, 1991

Record of Revision

Revision	Description
A	1991 - Original printing

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

Order desk (612)681-5907
Fax number (612)681-5920

TABLE OF CONTENTS

The OWS-E, cont'd.

The Operator's UNICOS Interface

UNICOS Startup, Shutdown, and Crashes

The Batch Subsystem (NQS)

The Batch Subsystem (NQS), cont'd.

Connectivity

Connectivity, cont'd.

Filesystems

Filesystems, cont'd.

The Tape Subsystem

Performance Monitoring

Useful Manuals

Introduction

Course introduction

This course is designed to aid operators in acquiring the skills necessary to monitor and control the UNICOS Operating System from the OWS-E workstation. The workbook is not intended to be used as a reference manual, but as a source for examples of the commands and monitors operators need. There are other reference manuals that should be used as sources of information to resolve points not covered in the workbook.

Course prerequisites

To obtain the best results from this course you should:

- Understand UNIX or UNICOS files system structure and command format.
- Have at least 16 hours hands-on experience using UNIX or UNICOS commands.
- Have a general understanding of operator responsibilities.

If you fail to meet the prerequisites, do not expect optimum results from this course.

Course materials

Materials you will be using in this course include:

- *Y-MP Model E Series Operator Training Workbook* (TR-UOT)
- *OWS-E Operator Workstation Operator's Guide* (SG-3078)
- *OWS-E Operator Workstation Reference Manual* (SR-3077)
- Access to an OWS-E or other computer system running SunOS and a CRAY computer system running UNICOS

Optional materials:

- *OWS-E Operator Workstation Administrator's Guide* (SG-3079)
- *UNICOS User Commands Ready Reference* (SQ-2056)
- *UNICOS Shell and Variable Ready Reference* (SQ-2060)
- *UNICOS vi Text Editor Ready Reference Card* (SQ-2054)
- *TCP/IP Network User Guide* (SG-2009)

Conventions used in this workbook

Commands and their arguments within text are shown in **boldface** `proportional` type. For example: **`qstat -a`**

All on-line examples are in `proportional` type and user input is **`proportional boldfaced`**. In addition, system prompts contain the name of the machine where the command is being executed. For example:

```
unicos$ who
mary     ttyp000  Jul 10 10:15   (sun1)
ops      ttyp002  Jul 10 08:08   (opsvax)
pete     ttyp033  Jul 10 10:17   (uscp-VX)
sarah    ttyp005  Jul 10 10:02   (SI)
```

System prompts used in examples in this workbook include:

`owse%`	(operator work station - C shell)
`owse#`	(operator work station - superuser)
`unicos$`	(UNICOS - Bourne shell)
`unicos#`	(UNICOS superuser)

A **`^C`** is the UNIX interrupt signal from a user. It is generated by pressing the CTRL and C (lowercase) keys simultaneously. A **`^D`** generates the UNIX end-of-file signal and is generated by pressing the CTRL and D (lowercase) keys simultaneously.

How to take this course

This course can either be read and studied independently or under the direction of an instructor in a lecture-base format.

If you study it alone it is recommended that you work at this course in a systematic manner. Only by reading the workbook and reference materials carefully and trying the commands on your system will you get the most out of this course and retain what you have learned. Of course, you will want to concentrate mostly on those sections that directly deal with your needs. Your administrator can help you decide what parts to concentrate on.

If you have time, read the sections a second time. Read the entire section carefully and methodically, highlighting key points, and making notations as you go. Concentrate especially on the examples.

The best way to learn something is to try it! This workbook and reference manuals should be used at a terminal so you can experiment, verify, or contradict what is written, and explore the limits and variations of the system.

Skills

This course is designed from a list of skills that have been identified as necessary for operators who use the UNICOS System.

Upon completion of this course the learner will be able to:

1. Perform operator work station startup and shutdown.
2. Perform Cray system startup and shutdown.
3. Monitor system activity.
4. Respond to tape subsystem messages.
5. Respond to error messages and dump procedures.
6. Perform back-up procedures.
7. Perform general troubleshooting procedures.

Learning log description

Progress in your level of competency can be graphed on a learning log. It is based on your own evaluation of your progress. Below are the levels you are to use when completing your learning log. On the following page is an example of a course participant's learning log. On the page following that is a blank log for you to use to indicate your current competency level and your continuing progress. The back side of the learning log is for the instructor's use. When you have completed the course, turn your completed learning log in to your advisor.

Evaluation method

Evaluation of your progress in acquiring these skills is done by the learner assigning a competency level to each skill.

Level	Description
0	No knowledge and no experience.
1	Has some knowledge and limited experience with this skill, but not sufficient to contribute in a work environment.
2	Can perform some parts of this skill satisfactorily but requires instruction and supervision to perform the entire skill.
3	Can perform some parts of this skill satisfactorily but requires periodic supervision and/or assistance.
4	Can perform this skill satisfactorily without assistance or supervision.
5	Can perform this skill with proficiency, speed, and quality without assistance or supervision.
6	Can perform this skill with initiative and adaptability to special situations without assistance or supervision.
7	Can perform this skill and can lead others in performing it.

Successfully completing this course should give you a competency level of at least three for most skills. Experience on the job will continue to increase your competency level.

Student Learning Log Sample

CRAY Y-MP Model E Series
Operator Training

Date: _____

Participant's Name: _____

Region/Country: _____

Instructor's Name: _____ Sample

Competency Levels:

0 Have no background in and no knowledge of the topic
1 Have vague knowledge of the topic or underlying concept
2 Need instruction in most areas of the topic
3 Need instruction in some but not all areas of the topic
4 Topic is familiar, but need detail
5 Can grasp the topic in detail
6 Am completely comfortable with all details of the topic

YUOTE	Learning Log						
Skills: At the end of the course the learner is able to:							
Perform Operator Work Station startup and shutdown	9/9				9/10		
Perform Cray system startup and shutdown		9/9		9/10			
Monitor system activity	9/9			9/10			
Respond to tape subsystem messages		9/9		9/10			
Respond to error messages and dump procedures	9/9				9/10		
Perform back-up procedures			9/9		9/10		
Perform general system troubleshooting		9/9		9/10			
Competency Levels:	**0**	**1**	**2**	**3**	**4 ***	**5**	**6**

* Maximum level discernable by the instructor in an instructional environment

Sessions attended/held _____/_____

This learning log is intended as an aid to the learner in establishing goals and plotting progress. It is not intended as an indicator of future job performance and therefore should not be used in determining future job actions.

Instructor's Feedback

Sessions attended/held _____/_____
Exercises completed/assigned _____/_____
Labs attended/held _____/_____
Test: student score (%) _____
class average (%) _____

MET THE PREREQUISITES OF THE COURSE

Not at all — Yes — Was over qualified

Specifics:

SELF APPRAISAL

Too High — Is Correct — Too low

3 Levels — 2 Levels — 1 level — 1 level — 2 Levels — 3 Levels

Specifics:

WAS ACTIVE AND ATTENTIVE IN CLASS

Not at all — To a normal degree — Exceptionally so

Specifics:

MADE GOOD USE OF TERMINAL TIME

Not at all — To a normal degree — Exceptionally so

Specifics:

KEPT UP WITH THE REST OF THE CLASS

Fell behind the class — Yes — Was ahead of the class

Specifics:

SHOWS A POSITIVE ATTITUDE ABOUT WORKING AT CRAY

Not at all — To a normal degree — Exceptionally so

Specifics:

Comments:

Student Learning Log

CRAY Y-MP Model E Series
Operator Training

Date: _____

Participant's Name: _____

Region/Country: _____

Instructor's Name: _____

Competency Levels:

0 Have no background in and no knowledge of the topic
1 Have vague knowledge of the topic or underlying concept
2 Need instruction in most areas of the topic
3 Need instruction in some but not all areas of the topic
4 Topic is familiar, but need detail
5 Can grasp the topic in detail
6 Am completely comfortable with all details of the topic

YUOTE Learning Log							
Skills: At the end of the course the learner is able to:							
Perform Operator Work Station startup and shutdown							
Perform Cray system startup and shutdown							
Monitor system activity							
Respond to tape subsystem messages							
Respond to error messages and dump procedures							
Perform back-up procedures							
Perform general system troubleshooting							
Competency Levels:	0	1	2	3	4 *	5	6

Sessions attended/held _____/_____

* Maximum level discernable by the instructor in an instructional environment

This learning log is intended as an aid to the learner in establishing goals and plotting progress. It is not intended as an indicator of future job performance and therefore should not be used in determining future job actions.

Instructor's Feedback

Sessions attended/held _____/_____
Exercises completed/assigned _____/_____
Labs attended/held _____/_____
Test: student score (%) _____
class average (%) _____

MET THE PREREQUISITES OF THE COURSE

Not at all — Yes — Was over qualified

Specifics:

SELF APPRAISAL

Too High — Is Correct — Too low

3 Levels — 2 Levels — 1 level — 1 level — 2 Levels — 3 Levels

Specifics:

WAS ACTIVE AND ATTENTIVE IN CLASS

Not at all — To a normal degree — Exceptionally so

Specifics:

MADE GOOD USE OF TERMINAL TIME

Not at all — To a normal degree — Exceptionally so

Specifics:

KEPT UP WITH THE REST OF THE CLASS

Fell behind the class — Yes — Was ahead of the class

Specifics:

SHOWS A POSITIVE ATTITUDE ABOUT WORKING AT CRAY

Not at all — To a normal degree — Exceptionally so

Specifics:

Comments:

The Operator's Environment

Operators and administrators

The role of an operator on a UNICOS system is difficult to define, because it depends to a very large extent on your system configuration, the ability of your users, and the roles of other staff in your company. In general, however, you will find that on a regular basis your tasks will interact fairly closely with those of your UNICOS system administrator. You and your administrator need to work together to define which tasks are operational, and which are administrative. This training course deals with the tasks that are usually considered to be operational. If you need more information on the administrative aspects of running a UNICOS system, then you should attend a UNICOS system administration course as well, such as **UNICOS System Administration**.

This course assumes a basic user-level knowledge of simple shell programming and basic UNICOS commands, and familiarity with fundamental operational terms and concepts such as **boot, dump, disk,** etc., which are non-UNICOS specific. No knowledge of Cray hardware is assumed.

Cray system hardware overview

The next page shows some Cray Y-MP Model E configuration examples. The configuration of your system may be a little different, but will be based on the same major components and connections. Ask your administrator for a diagram of your hardware configuration.

Every system consists of a Cray mainframe computer (e.g., Cray Y-MP), and an Input/Output Subsystem, which is usually referred to as an IOS. Your system may also have a Solid State Storage Device (SSD).

The mainframe computer runs software called UNICOS, and does the "real work". It has a number of Central Processors (CPUs) and central memory.

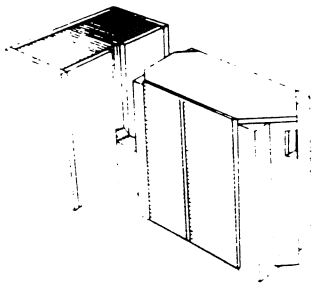
The IOS is a separate computer that handles all the input and output of data (I/O). It is connected to the mainframe. It is also connected to Cray disks, other computers and networks, and to tapes. More information on the IOS follows on the next page.

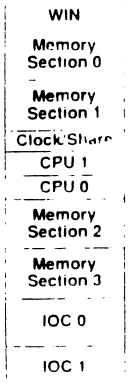
The SSD is best described as a very high speed disk device. It comes in a variety of sizes and is optional.

Your interface to the Cray system is through the Operator Workstation (OWS-E). It is a Unix-based system which provides you with a windowing system to control and monitor the software in the Cray mainframe. The next section discusses that environment. Hardware support personnel have a nearly identical interface to the Cray system called the Maintenance Workstation (MWS-E) from which diagnostics are run.

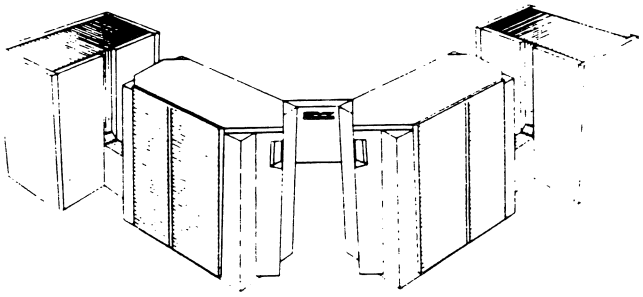
Find out from your administrator how many CPUs and how much memory your Cray mainframe has, what the IOS configuration is, and how large the SSD is if there is one. You can also learn about your system's mainframe hardware by using the UNICOS **target** command. See the man page for `target` for more information.

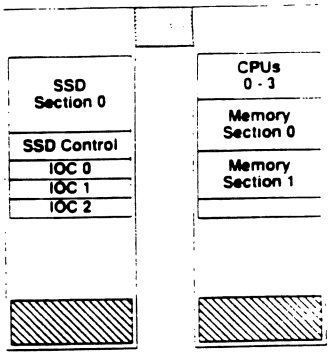
Y-MP2E Computer System



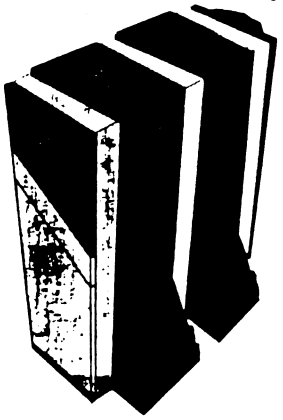


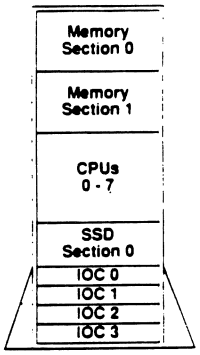
Y-MP4E System



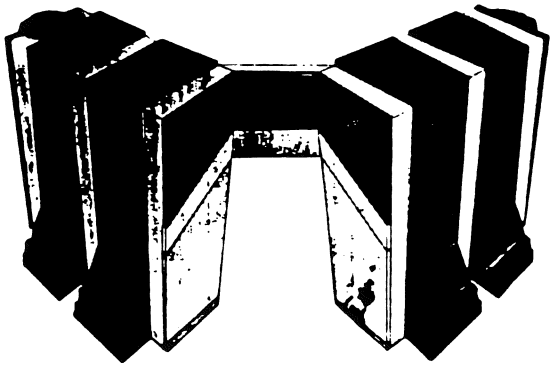


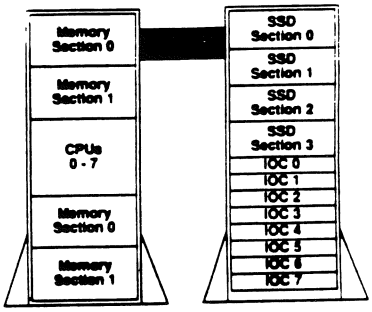
Y-MP8I System





Y-MP8E System





IOS Model E overview

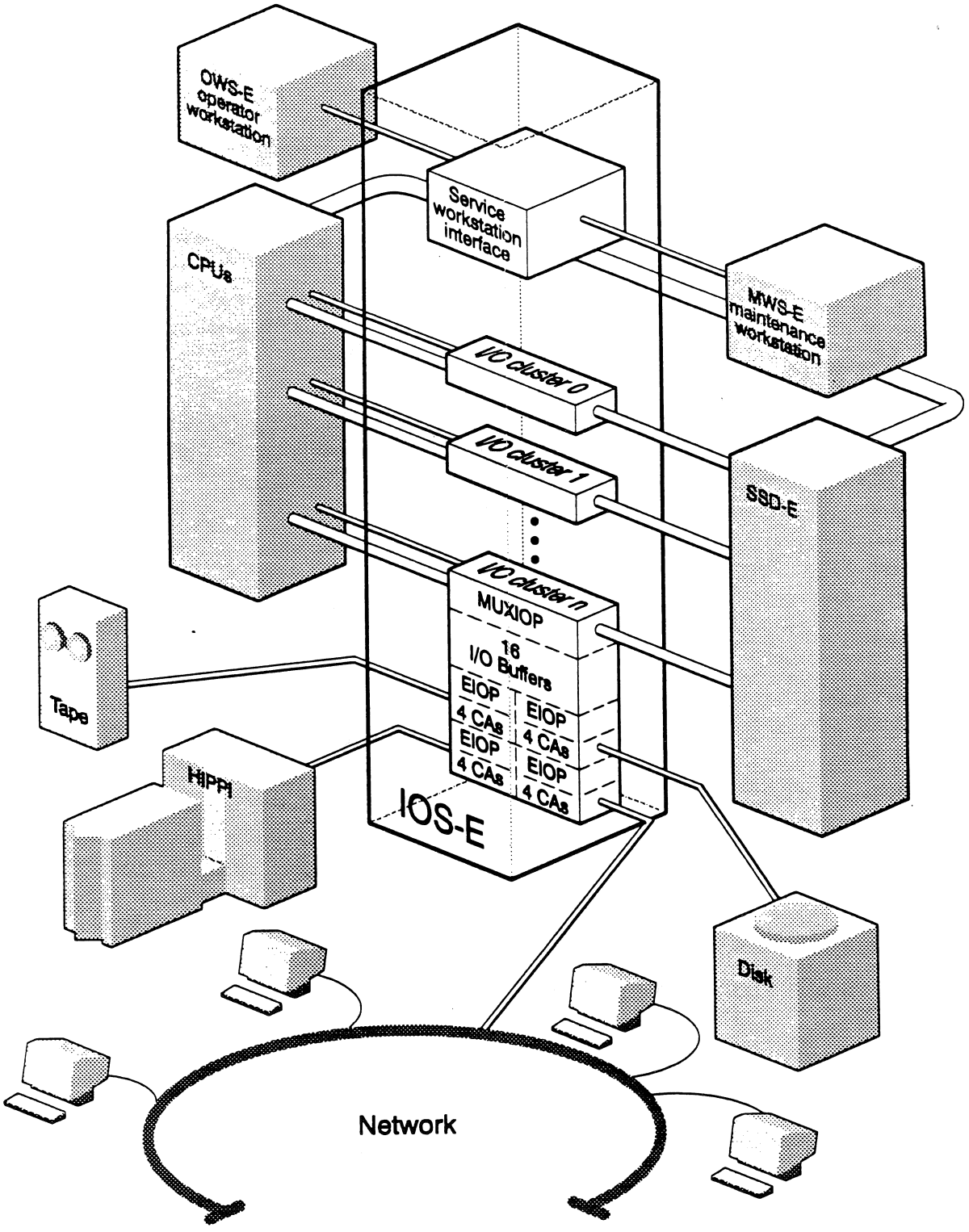
The IOS Model E actually consists of as many as 32 identical I/O Processors (IOPs) that all work independently, handling different aspects of the I/O. Those IOPs are grouped into clusters (IOCs). Each cluster is associated with one low-speed (LOSP) and one high-speed (HISP) channel. The LOSP is used for control information and the HISP transfers the data. Each IOC contains 4 IOPs plus one MUX (Multiplexor) IOP. The MUX IOP controls the IOPs, directing the data throughput between them and the mainframe.

Each IOP supports four channel adapters. However, an IOP can support only one kind of channel adapter. There are several kinds - for disk, tape, networking communications, and special high-speed communications. You will need to know what devices are configured on your system. Each channel adapter is associated with one 64 Kword circular I/O buffer where data is temporarily stored on its way to or from the Cray mainframe.

Emergency procedures

Part of your job as operator is to be very familiar with the procedures your installation follows in the event of hardware-related emergencies such as power fluctuations or outages, water leaks, earthquakes, etc. It may seem humorous to think of them now, but they do happen! Ask your system administrator what the various procedures are **before** a problem arises.

IOS Model E


OWS-E operator workstation
Service workstation interface
CPUs
MWS-E maintenance workstation
I/O cluster 0
I/O cluster 1
SSD-E
I/O cluster n
MUXIOP
16 I/O Buffers
EIOP 4 CAs
EIOP 4 CAs
EIOP 4 CAs
EIOP 4 CAs
Tape
HIPPI
IOS-E
Disk
Network

UNICOS overview

The Cray mainframe runs an operating system called UNICOS. As system operator, you need to know a bit about the system software as well as the user-level interface to it. The IOS runs separate software, but you do not need to know much about that software to be able to operate the system. The SSD does not run any software.

UNICOS is composed of a memory-resident kernel which is the system's interface to the Cray hardware. Kernel functions are accessed by utilities and applications. The central UNICOS utility which is accessed by all users of the system is the shell. There are three slightly different shells: the Bourne shell, the C shell, and the K shell. These shells act as a command interpreter and as a high-level programming language. When you log in to UNICOS, a shell controls your login session, interpretting all the commands you type in. This shell is referred to as your "login shell". The default shell used in UNICOS is the Bourne shell. The default shell used on the OWS-E is the C shell.

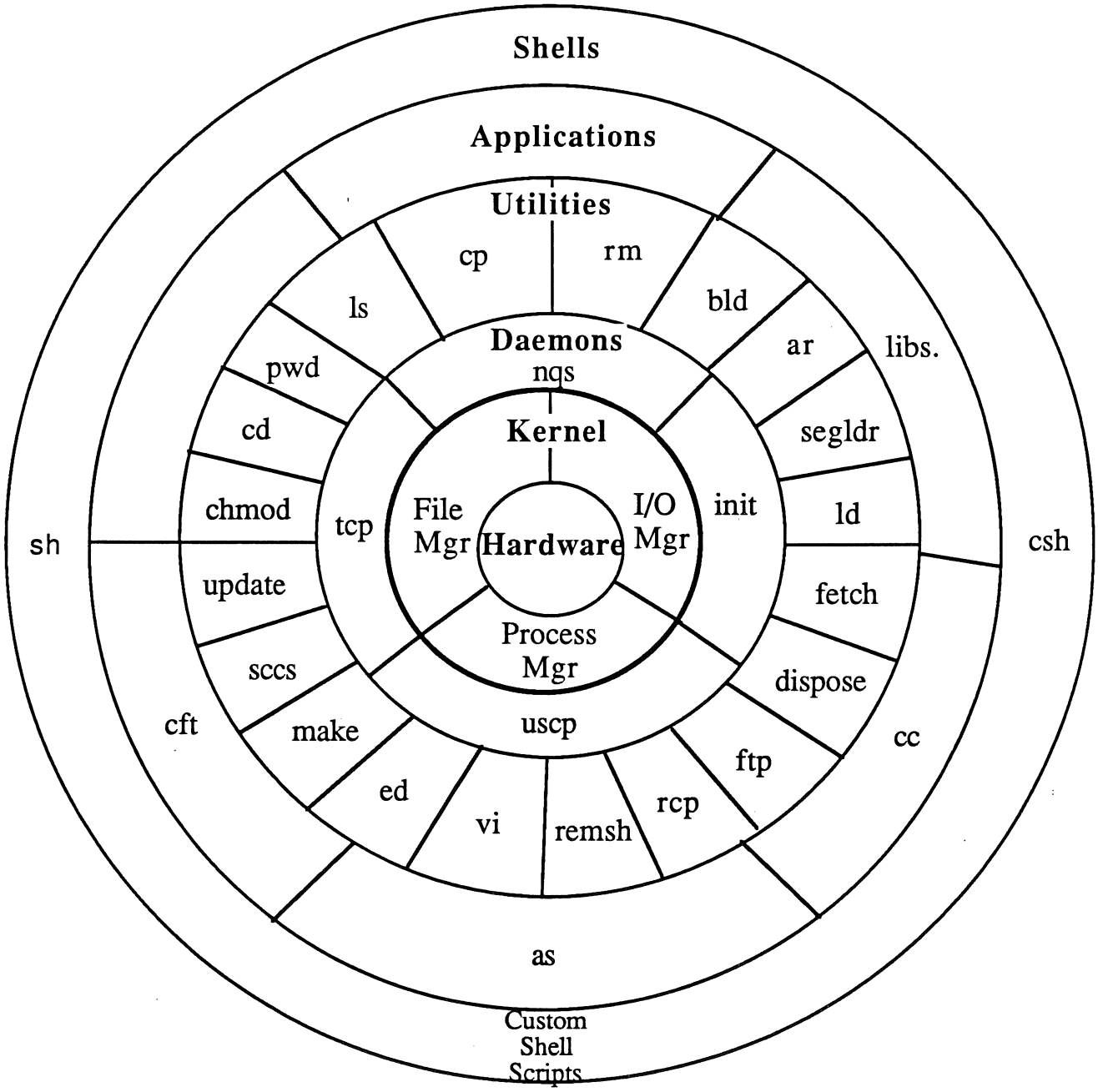
As system operator you will find it extremely useful to be able to program the shell, so that you can write your own operator commands. (It is not as difficult as it sounds!) It is recommended that you attend a Bourne shell programming course to learn how to do this, such as the UNICOS Command Language and Shell Programming course (UCL).

As system operator, you also need to be aware of some special system daemons. A system daemon is just a command that runs independently of any user or terminal. Each daemon handles a specific subsystem of UNICOS, and for this reason, you can think of them as being closer to the kernel than user-level shells and commands. For example there is a daemon for tape handling and a daemon to control the batch subsystem. The daemons are normally started when UNICOS is started.

OWS-E software overview

The OWS-E runs three main layers of software that operators need to be familiar with. One is the operating system, SunOS, a version of Unix. Another is the operator interface, OpenWindows, a version of the X Window System which provides many useful windowing tools to monitor Cray activity. The third layer of software is the group of commands which allow operators to control the IOS and mainframe software. Collectively these three layers of software are referred to as the OWS-E operating system. These layers are discussed more later.

Levels of UNICOS



The OWS-E

This section briefly describes the OWS-E hardware and provides an overview of the OWS-E software used to operate and monitor the Cray system.

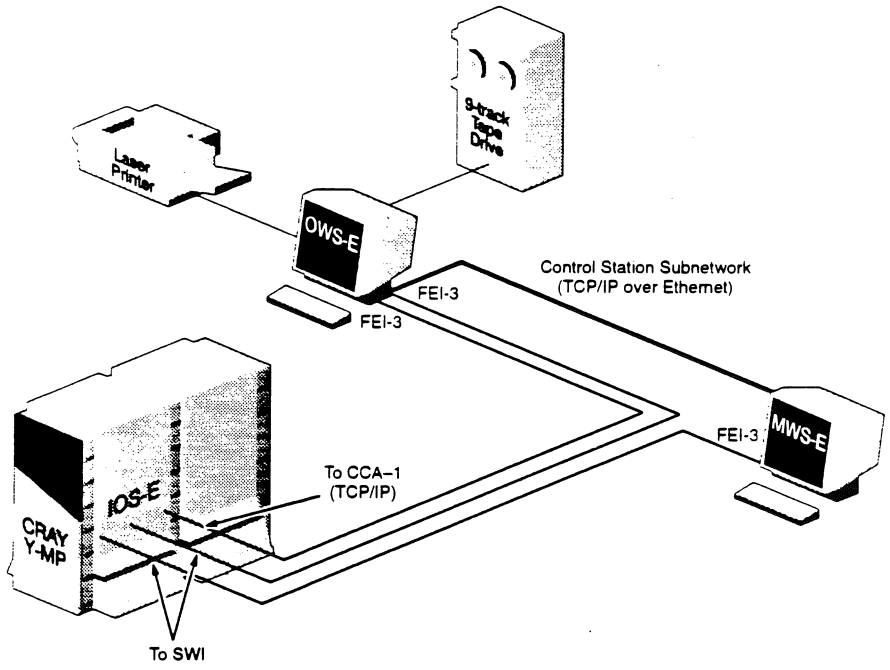
Workstation hardware description

The Operator Workstation (OWS-E) is an independent computer system which has a graphics screen, a compact-disk read-only memory (CD ROM) reader, a disk drive, a cartridge tape drive, a laser printer, and an optional 9-track tape drive. Currently UNICOS release tapes are available in both cartridge and 9-track format (round). It is used to start up, shut down and control the Cray system. The OWS-E runs the Unix operating system, so it is very similar to UNICOS. It is used only by the system operators and system administrators, not by users. The diagram on page the next page shows the major components of the OWS-E.

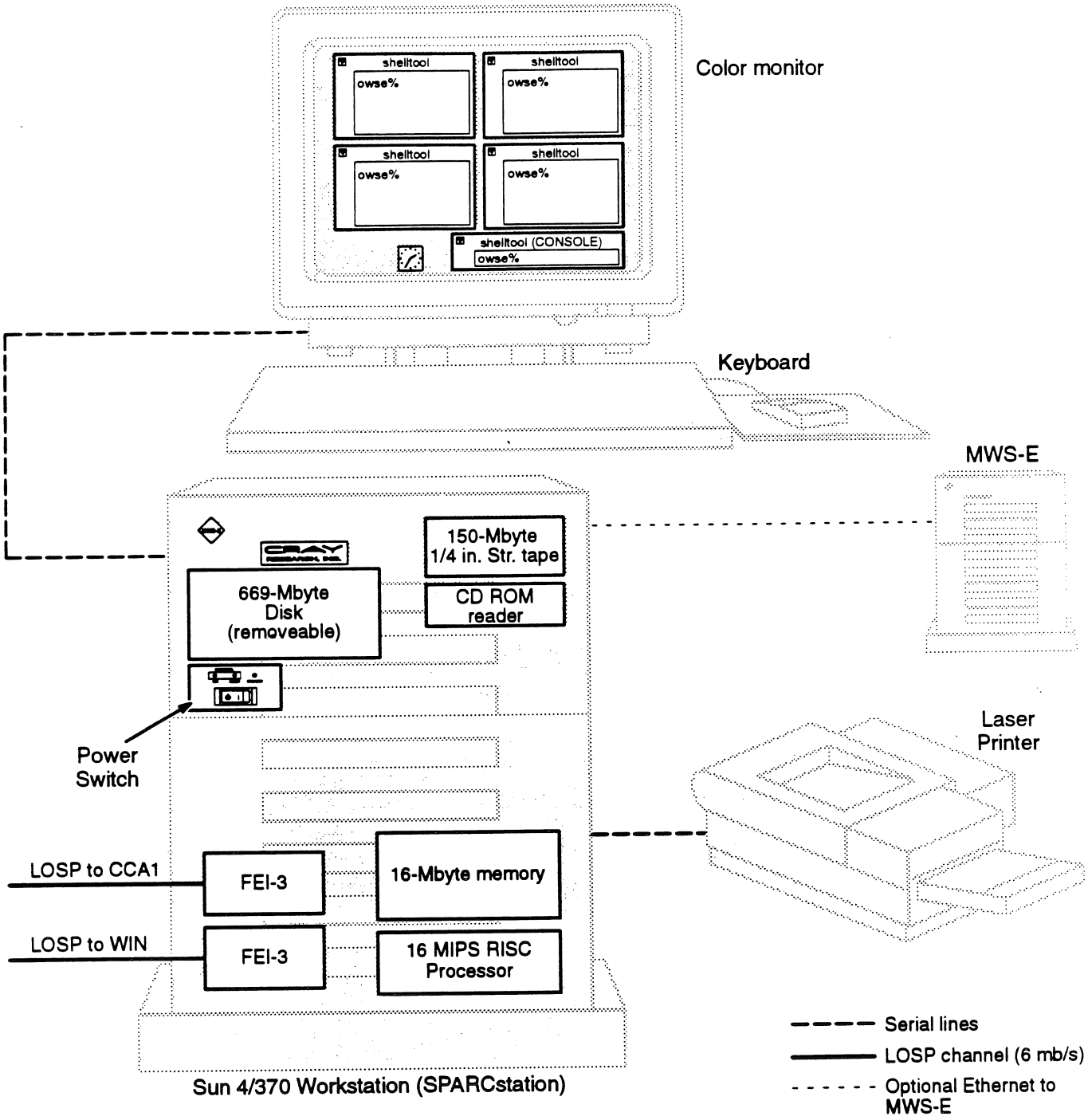
The OWS-E has two connections into the IOS-E, both using FEI-3 board sets. One is a connection to a standard communications channel (CCA-1) and the other is a special connection into a Service Workstation Interface (SWI) on the IOS-E.

A very similar workstation called the Maintenance Work Station (MWS-E) is also likely to be near the OWS-E. It has most of the same components as the OWS-E, but is used by hardware analysts (field engineers) to run diagnostics on the Cray system hardware. The MWS-E has one FEI-3 board set for a connection into the SWI similar to the OWS-E.

Note that the OWS-E and the MWS-E can communicate with each other via a dedicated Ethernet.



The Model E workstation



OWS-E software system overview

As described earlier, the OWS-E operating system consists of three main layers: SunOS (Unix), OpenWindows, and the command set to control and monitor the Cray system.

The OWS-E runs a graphics software package called OpenWindows. It is usual to display many windows on the workstation monitor that can provide you with information about the Cray system. After the OWS-E is booted you will normally login as **cri**. The home directory for login **cri** is /home/*localhost*/cri. A shortcut method of referencing this cri's home directory is ~cri.

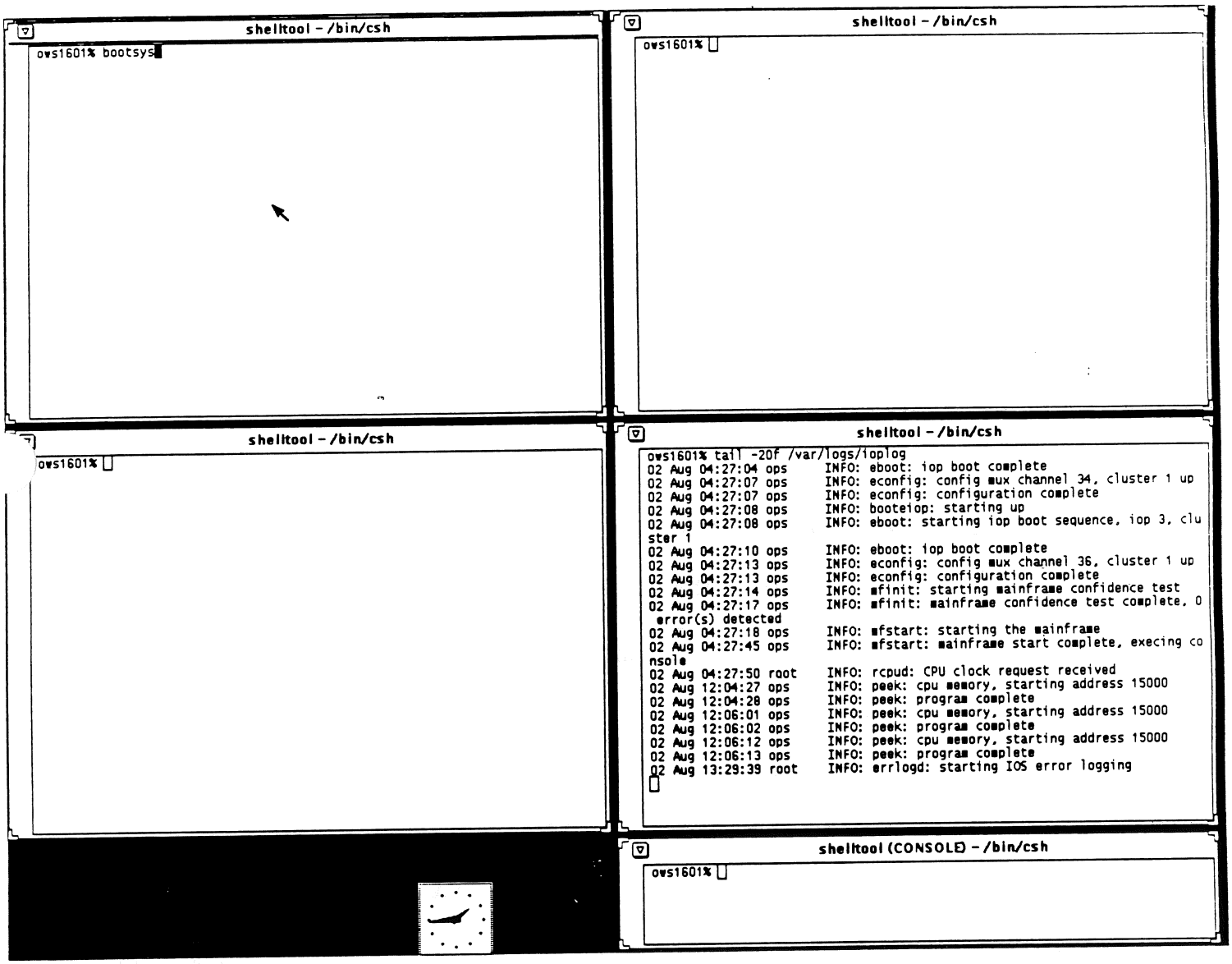
Five windows appear - four OWS-E windows and a small OWS-E system console window. The system console window is where the OWS-E sends important messages about how it is running. The other windows are provided as login windows to monitor and control the Cray system running UNICOS. A clock is also placed on the bottom of the screen. This startup environment is defined in an OpenWindow startup file called **.openwin-init**. This file is in the home directory of userid **cri** (or your operator login id) on the OWS-E.

At some point, you may want to change the layout of the screen a bit to suit your system. However, if you want to change it, you need to read some OpenWindow documentation first or talk to your system administrator. If the setup on your OWS-E is not default, then you need to talk to the person who set things up and make sure you understand how to use the graphics screen.

Another file called **.openwin-menu** is also referenced at login. This file contains the code to define the pull-down menus that can be accessed with the mouse buttons. These menus allow the operator to easily run a variety of commands. The menus can also be modified to include site-specific commands.

To learn how to manipulate these windows and menus, refer to the OWS-E Operator Workstation Operator's Guide (SG-3078) or the OpenWindow documentation that came with the workstation.

OWS-E workspace
(default OpenWindow configuration)



Shutting down the OWS-E

There may be times that you need to reboot the OWS-E, perhaps because it is behaving erratically. To shut down and immediately reboot the OWS-E, do the following:

1. Exit out of OpenWindows by placing the mouse pointer on the workspace background, holding down the right button, and sliding down the menu to `Exit`. Then choose the `Exit` box.

2. Logout using the `logout` command or `^D`.

3. Log back in as `root`. If you get the following prompt, answer `n`.

   ```
   Start OpenWindows now? [n]
   ```

4. Run the `who` command to make sure you are the only one logged onto the OWS-E.

5. Type `shutdown now` for the quickest shutdown.

6. Type `sync` to flush the disk buffers to disk.

7. Type `fastboot` to immediately reboot the OWS-E.

Powering down the OWS-E

You will need to power down the OWS-E if you are swapping disks or if hardware maintenance needs to be done. To power down the OWS-E, complete steps 1-6 listed above, but do the following instead of step 7 above:

7. Type `halt` to stop the OWS-E.

At the > prompt you can safely power-off the OWS-E.

NOTE: If the OWS-E will not respond to ANY commands and appears locked up, press the L1 and A keys simultaneously to get the > prompt. Type a `b` to reboot the OWS-E. If you decide not to boot the OWS-E, type `c` to continue where you were. Use the above method of shutting down the OWS-E instead of this method when possible. Note also that the OWS-E logs messages from UNICOS and cannot do so when down.

The following page shows a login example of the shutown process.

Output of the shutdown and fastboot process on the OWS-E

(exit OpenWindows)

```
owse% logout
login: root
Password:
Last login: .....
.
.
owse# who
root     console Jul 26 17:17

owse# shutdown now

    *** FINAL System shutdown message from root@owse ***

System going down IMMEDIATELY

System shutdown time has arrived

owse# sync
owse# fastboot
syncing file systems . . . done
rebooting . . .
Boot: sd(0,0,0)
.
.
.
login:
```

OWS-E backup basics

It is important to back up your OWS-E system so that you can restore files that have been accidently damaged or deleted. To simplify this, two commands are provided on the OWS-E that allow you to make regular backups to cartridge tape.

The `dumpdly` command creates incremental dumps for each filesystem on the workstation that has changed since the last `dumpdly` dump was done. To use it, simply type in the command and follow the prompts.

The `dumpwkly` command creates full dumps (every file) of each filesystem on the workstation. To use it, simply type in the command and follow the prompts.

Output from the dumpdly command

```
owse% dumpdly

This script will perform a daily backup of the operator
workstation.  There will be at least six tapes required, one for
each filesystem.

Label backup tape: Daily Backup - /dev/sd0a (/)
Insert the backup tape into the tape drive
Enter <carriage return> when ready:
  DUMP: Date of this level 9 dump: Wed Jun 26 13:04:39 1991
  DUMP: Date of last level 0 dump: Mon Jun 24 13:40:35 1991
  DUMP: Dumping /dev/rsd0a (/) to /dev/rst0
  DUMP: mapping (Pass I) [regular files]
  DUMP: mapping (Pass II) [directories]
  DUMP: mapping (Pass II) [directories]
  DUMP: estimated 1522 blocks (761KB) on 0.02 tape(s).
  DUMP: dumping (Pass III) [directories]
  DUMP: dumping (Pass IV) [regular files]
  DUMP: Tape rewinding
  DUMP: 1526 blocks (763KB) on 1 volume
  DUMP: DUMP IS DONE
Label next backup tape: Daily Backup - /dev/sd0d (/usr)
Insert the next backup tape into the tape drive
Enter <carriage return> when ready:
  DUMP: Date of this level 9 dump: Wed Jun 26 13:05:22 1991
  DUMP: Date of last level 0 dump: Mon Jun 24 13:43:05 1991
  DUMP: Dumping /dev/rsd0d (/usr) to /dev/rst0
  DUMP: mapping (Pass I) [regular files]
  DUMP: mapping (Pass II) [directories]
  DUMP: estimated 148 blocks (74KB) on 0.00 tape(s).
  DUMP: dumping (Pass III) [directories]
  DUMP: dumping (Pass IV) [regular files]
  DUMP: Tape rewinding
  DUMP: 510 blocks (255KB) on 1 volume
  DUMP: DUMP IS DONE
Label next backup tape: Daily Backup - /dev/sd0e (/export)
Insert the next backup tape into the tape drive
Enter <carriage return> when ready:
  DUMP: Date of this level 9 dump: Wed Jun 26 13:06:08 1991
  DUMP: Date of last level 0 dump: Mon Jun 24 14:11:51 1991
  DUMP: Dumping /dev/rsd0e (/export) to /dev/rst0
  DUMP: mapping (Pass I) [regular files]
  DUMP: mapping (Pass II) [directories]
  DUMP: estimated 134 blocks (67KB) on 0.00 tape(s).
  DUMP: dumping (Pass III) [directories]
  DUMP: dumping (Pass IV) [regular files]
  DUMP: Tape rewinding
  DUMP: 156 blocks (78KB) on 1 volume
  DUMP: DUMP IS DONE
  .
  .
  .
  .
  .
owse%
```

zip, ucon, and cztool

zip is an OWS-E command that provides a special login connection to the Cray mainframe. There are 20 of these connections; however most of them are reserved for use by other special monitoring programs. The connections available to operators are defined as `/dev/cz00`, `/dev/cz01`, `/dev/cz02`, `/dev/cz03`, and `/dev/cz04`. These connections act similarly to normal tty connections, but they provide a means of connecting to UNICOS even if the UNICOS network software is not functioning.

The most important one for operators is `/dev/cz00`, which is the connection to the UNICOS console. UNICOS sends important messages to the console. It is also the normal place for bringing up and shutting down UNICOS. The connection to the UNICOS console is automatically made whenever the `bootsys` command is run. However, there may be times when that connection has been terminated and needs to be restarted. To do that you need to run the `zip` command in one of the OWS-E windows.

There may also be a situation where another user has taken over that connection from another workstation and you need it. To regain control over that connection you can use the **ucon** command, which allows you to "usurp control of the console". This allows you to disable the current user from `/dev/cz00`. You can then run the `zip` command to regain control of the console. A warning message is entered into the `/var/logs/ioplog` whenever the mainframe console is usurped.

Before you take away control of the console from a user you may want to know who has it so you can notify them. `ucon` tells you what process has it and the `ps` command can show you what user has it. You can also use the **cztool** command to find out what process has it by using the `c` option at the `cztool` prompt.

If you do not specify which `cz` connection you want at the `zip` prompt, `zip` chooses the next available one, the search starting with `/dev/cz00`.

NOTE: If you ever `telnet` to the OWS-E and then use `zip` to connect to the mainframe, be sure to change the escape character in `zip` from `^]` to something else. If you don't and then escape from the Cray login, it will bring you back to the `telnet` prompt and you will not be able to quit `zip` and get back to the OWS-E. You will have to type `quit` at the `telnet` prompt which brings you back to your local computer system. See the man page on `zip` for more.

See the next page for some examples of the above commands.

Examples of zip, ucon, and cztool

```
owse% zip
zip> open /dev/cz00
Can't open tty channel /dev/cz00, errno 16
zip: Device busy
zip> q
owse% cztool

cztool> c
lpath   pid       sel   valid mnooc mtout mquota     rf       rl    sig

 0     4169 f81b87d8     1     6     0   0          0 f816ee1c     0
 7      112         0     8     0     0   0   f816edd8 f816edd8     0
 8      110         0     2     0     0   0          0        0     0
 9      111         0     0     0     0   0          0        0     0
10     4171         0     0     0     0   0          0 f816eea4     0


cztool> q
owse% ucon
WARNING: ucon: console in use by process 4169, usurp? [y/n] n
owse% ps -aux|grep 4169
ops       4169  0.0  0.0   88    0 p7 IW   15:42   0:00 zip
ops       4174  0.0  1.8   40  264 p3 S    15:43   0:00 grep 4169
owse% ucon
WARNING: ucon: console in use by process 4169, usurp? [y/n] y
owse% ucon
INFO: ucon: mainframe console not open
owse% zip
zip> open
Trying /dev/cz00...

Connected to mainframe via /dev/cz00
Routed via mux, cluster 0
Escape character is '^]'


Console Login:
```

The `estat` command

A simple way to find out if the IOPs are running is by typing in the `estat` command. There are options on the command to check the status of individual IOPs. Typing in the command with no options will give the status of all the IOPs and will also show that IOPs are not running in IO clusters that do not exist (up to the maximum possible in an IOSE).

The sample output of the `estat` command on the next page shows that configuration to have only one cluster of IOPs and the IOPs in that cluster are all running.

Output from the `estat` command

```
owse% estat
INFO: estat: iop 0, cluster 0 running
INFO: estat: iop 1, cluster 0 running
INFO: estat: iop 2, cluster 0 running
INFO: estat: iop 3, cluster 0 running
INFO: estat: iop 4, cluster 0 running
INFO: estat: no iop in cluster 1 running
INFO: estat: no iop in cluster 2 running
INFO: estat: no iop in cluster 3 running
INFO: estat: no iop in cluster 4 running
INFO: estat: no iop in cluster 5 running
INFO: estat: no iop in cluster 6 running
INFO: estat: no iop in cluster 7 running
owse%
```

The /etc/owsepermfile

In order to make it easier to allow different login accounts to run the main OWS-E commands, a special file has been created on the OWS-E which contains a list of those eligible to run any or all of these commands. That file is called `/etc/owsepermfile` in OWS-E 1.1 (it was `/etc/cwspermfile` in OWS-E 1.0).

Whenever a user attempts to run one of the OWS-E commands, the file is checked to see if they are eligible to run it. An example of the file is listed on the following page which includes comments on how it works and which commands check the file.

Example of the /etc/owsepermfile

```
owse% cat /etc/owsepermfile
#
#       (c) Copyright Cray Research, Inc.  Unpublished Proprietary Information. All Rights Reserved.
#
#       /etc/owsepermfile    File to control permissions to execute commands from CRAY RESEARCH INC.
#
#       The format of this file is:
#
#       Special characters and limitations:
#            - the '#' character signifies a comment which continues until a newline is encountered.
#            - A line may be no longer than 4096 characters.  No continuation lines are allowed.
#            - White space is made up of the characters <space> and <tab>
#            - Separators are the characters <colon> <comma> <semi-colon> <space> and <tab>
#            - The first line for a user is the only one used, later lines are ignored.
#            - This file must be owned by root or cri, and must not have
#              any of the following permission bits set: 07033
#            - The '*' character in the "names" section of a line will allow
#              a user to run any program using this file
#
#       Line format:
#            - Each line consists of a user name and the "names" of the programs that user may run.
#              Therefore, a simple line could consist of "name,program"
#            - The program "names" are the defined names for programs, not complete pathnames.
#              If a binary is renamed in the file system, the "name" will still be the same.
#              Legal names are:
#
#                                   autodump
#                                   cpuhalt
#                                   cztool
#                                   eboot
#                                   echopkt
#                                   econ
#                                   ecrash
#                                   edump
#                                   ehalt
#                                   emon
#                                   errlogd
#                                   estat
#                                   hbeat
#                                   hcon
#                                   mfdump
#                                   mfinit
#                                   mfstart
#                                   peek
#                                   poke
#                                   rcpud
#                                   resetscreen
#                                   zip
#
#           - The user name and the program name must be separated by at least one separator.
#
#       An example (if the comment characters were removed)
#user1     mfstart,edump,eboot          # allow mfstart edump eboot
#user2:mfstart,edump  eboot             # allow mfstart edump eboot
#user3    mfstart   edump    eboot      # allow mfstart edump eboot
#user4     *                        # do anything!
#       Set the permissions for the default accounts to allow them access to all programs.
cri     *
mws     *
ops     *
#       End of owsepermfile
```

The hbeat, rcpud, errlogd, and cpud daemons

When the Cray system (IOS-E and UNICOS) is booted, four daemons are started on the OWS-E which monitor and report how the Cray system is functioning. They are **hbeat**, **rcpud**, **errlogd**, and **cpud**.

`hbeat` continually polls each booted IOP, looking for either halt codes or clocks that have stopped (indicating a hung system). If either condition is detected, it runs a script which sends a message to the console (`/dev/cz00`) informing the operator and logs the condition in `/var/logs/ioplog`. `hbeat` also runs another script which dumps the IOS-E.

`rcpud` processes service requests such as starting an IOP, halting an IOP, upping and downing IOPs, notifying the operator that an IOP is alive or dead, getting the CPU date and time, and, when UNICOS panics, running a script that can automatically dump and restart the system.

`errlogd` monitors for hardware errors in the IOS-E and logs them in the `/var/logs/errlog` file.

`cpud` gathers and disperses CPU time statistics for the `graphs` command (`graph` with OWS-E 1.0).

The emon command

Operators will normally not need to deal with these daemons since they are started automatically at boot time. However, they can disappear from the system and you may notice that system messages are not being accumulated. You can determine if they are running by using the `ps` command (e.g., `ps -aux`). Daemons that are down can easily be restarted with the **emon** command.

`emon` restarts any or all of the four daemons if they are down. If you enter `emon` when the processes are already running, not damage will be done and you will not get an error message. You will get an error message only if `emon` is unable to start one of the daemons.

The .version file

If you need to find out what version of OWS-E software is installed on your workstation, simply `cat` the file `~cri/.version`. An example of that file's contents is shown below. This file did not exist until OWS-E release 1.1.

```
ows% cat ~cri/.version
OWSE 1.1        11 Jun 91       CRI
```

The Operator's UNICOS Interface

Setting up your UNICOS operating environment

A special account will have been set up on UNICOS for operational use. Typically it is login **operator**. You should use this account for all UNICOS operations because it has special privileges to allow you to perform tasks that most users cannot.

There are a number of things you will probably want to set up to tune your UNICOS environment. The most obvious one is your **PATH** variable, since the default path will not include all of the operator commands. In particular you should include the following in your path: **/etc**, **/usr/lib**, **/usr/lib/msg**, and **/usr/lib/monitor**.

Don't forget that any UNICOS commands can be grouped together into a file and executed by just typing the name of the file, as long as the file has execute permission. These are known as shell scripts. So, if you find that you frequently need to use one or more commands that involve too much typing, save yourself the effort!

If you define your own commands as suggested above, the best place to put them is in a directory called **bin** under your **HOME** directory. You may need to **mkdir** a **bin** directory if there is not one already. You will also need to add your new **bin** directory into your default search path (**$HOME/bin**).

Most of your operating tasks will be done from the OWS-E, especially UNICOS shutdown and restart. From the OWS-E you will likely **telnet** to the Cray from the various windows. But most operating tasks can be performed from any terminal. To log on to UNICOS from another computer in your network using `telnet`, all you need to know is the name used by your computer to refer to the Cray. A few messages will appear, followed by a UNICOS login prompt. In this example you are logged on to a computer called **mysun**, which refers to the Cray computer as **cray**.

```
mysun% telnet cray
Trying 85.0.17.5 ...
Connected to cray
Escape character is '^]'.

Cray UNICOS

login:
```

Most operating tasks do not require you to log into the system as superuser. Do not logon as superuser unless you really need to. Many installations will not allow operators to log in as superuser under any circumstances.

Finding out what is going on in the system

There are a number of commands you can use to find out what is going on in the system. The two most basic ones are **who** and **ps**. In addition, there are several commands for examining the various subsystems, such as tapes, batch and networks. Those commands are discussed in the following chapters.

The who command

The **who** command lists the users currently logged into UNICOS interactively. With no options, the command tells you the terminal name, login time and originating machine name of each userid logged in.

```
unicos$ who
ops    console   Jul 12 15:22
mary   ttyp000   Jul 12 14:19   (sun1)
john   ttyp001   Jul 12 13:54   (mysun)
mark   ttyp002   Jul 12 15:28   (iris)
```

who -u displays the amount of time the user's terminal has been inactive ("." means active), and the "process id" of the user's login shell. (See **ps** section below.)

```
unicos$ who -u
ops    console   Jul 12 15:22   0:01   1979
mary   ttyp000   Jul 12 14:19   0:06   7481   (sun1)
john   ttyp001   Jul 12 13:54    .      482   (mysun)
mark   ttyp002   Jul 12 15:28   0:17   2567   (iris)
```

There are several other useful options on the **who** command. See the **man** page, and experiment with some of them.

The **ps** command

Every command or program running on UNICOS is referred to as a process. Each process can be uniquely identified by a number called a process id, or PID. Each process also has a parent process associated with it. For example, when you log in to the system, your login shell is a process with a unique PID. If you run the `who` command, the `who` process gets a different PID and has your login shell as its parent process id (PPID).

The **ps** command reports the status of processes in the system. Without any option, only the commands being run at the current terminal are reported. As system operator you will normally want to see more processes than this (usually all of them or perhaps all processes belonging to a particular user).

The three most useful options are **e**, **f**, and **u**. **ps -e** will list all processes in the system, instead of just the ones running from your terminal. **ps -f** will report the name of the userid, instead of the userid number. **ps -u mary** will report all processes belonging to the userid mary. There are lots of other options on the `ps` command. Read the man page for details.

Another command worth investigating is the **jstat** command. It can provide more useful information than `ps`. `jstat` is discussed in the Batch Subsystem section.

The **kill** command

Sometimes it will be necessary for you to kill a process, because it is causing some sort of problem or because a user has asked you to. In order to kill a process, you need to use the PID. Use the `ps` command to find the appropriate PID number, and also check the owner of the process by using the **-f** option. If you are sure that you have the correct PID, and, if the kill is authorized (e.g., the user's own process), then issue the command **kill *PID***

```
unicos$ kill 39841
```

If the process has not disappeared when you re-run the **ps** command, then try the command **kill -9 *PID***

```
unicos$ kill -9 39841
```

You will need to be superuser (i.e., root) to kill a process which does not belong to you.

Output examples from the ps command

```
unicos$ ps
   PID TTY   TIME COMMAND
   371 p001  0:00 csh
   665 p001  0:00 ps


unicos$ ps -e
   PID TTY   TIME COMMAND
     0 ?     0:04 swapper
     1 00    0:00 init
     2 ?    21:53 idle
   118 00    0:00 avrproc
   262 00    0:00 csh
    87 00    0:00 shrdaemo
    90 00    0:00 errdemon
    93 null  0:00 cron
   109 ?     0:00 msgdaemo
   111 kmem  0:00 tpdaemon
   263 01    0:00 sh
   370 ?     0:00 telnetd
   252 ?     0:00 logdaemo
   137 net   0:00 inetd
   139 ?     0:00 talkd
   141 00    0:00 sendmail
   144 00    0:00 crayperf
   249 net   0:00 portmap
   247 00    0:00 syslogd
   253 ?     0:00 nqsdaemo
   256 net   0:00 netdaemo
   257 ?     0:06 uscpd
   267 kmem  0:01 uscpcmd
   615 01    0:01 ping
   353 ?     0:00 telnetd
   354 p000  0:00 csh
   393 ?     0:00 rlogind
   295 00    0:00 csh
   433 p002  0:00 sh
   637 00    0:00 more
   666 p001  0:00 ps
   641 p000  0:00 vi
   667 kmem  0:00 qstat
```

Output examples from the ps command, cont'd.

```
unicos$ ps -ef
    UID   PID  PPID    STIME TTY   TIME COMMAND
   root     0     0        - ?     0:04 swapper
   root     1     0        - 00    0:00 [ init ]
   root     2     0        - ?    21:59 [ idle ]
   root   118   111 18:07:44 00    0:00 [ avrproc ]
   jlg    262     1 18:08:49 00    0:00 [ csh ]
   root    87     1 18:07:39 00    0:00 [ shrdaemo ]
   root    90     1        - 00    0:00 [ errdemon ]
   root    93     1        - null  0:00 [ cron ]
   root   109     1 18:07:42 ?     0:00 [ msgdaemo ]
   root   111     1 18:07:43 kmem  0:00 [ tpdaemon ]
   root   370   137 18:13:28 ?     0:00 [ telnetd ]
   root   252     1        - ?     0:00 [ logdaemo ]
   root   137     1 18:08:40 net   0:00 [ inetd ]
   root   139     1        - ?     0:00 [ talkd ]
   root   141     1 18:08:41 00    0:00 [ sendmail ]
   root   144     1 18:08:41 00    0:00 [ crayperf ]
   root   249     1        - net   0:00 [ portmap ]
   root   247     1 18:08:45 00    0:00 [ syslogd ]
   root   253     1        - ?     0:00 [ nqsdaemo ]
   root   256   253        - net   0:00 [ netdaemo ]
   root   257     1 18:08:47 ?     0:06 [ uscpd ]
   root   267   257 18:09:02 kmem  0:01 [ uscpcmd ]
   root   353   137 18:12:45 ?     0:00 [ telnetd ]
    cnw   354     1 18:12:46 p000  0:00 [ csh ]
   root   393   137 18:14:13 ?     0:00 [ rlogind ]
   root   295   262 18:09:09 00    0:00 [ csh ]
    smm   371     1 18:13:28 p001  0:00 [ csh ]
   imcd   394     1 18:14:13 p002  0:00 [ csh ]
   root   433   394 18:15:03 p002  0:00 [ sh ]
   root   637   295 18:26:13 00    0:00 [ more ]
    smm   669   371 18:28:15 p001  0:00 [ ps ]
    cnw   641   354 18:26:27 p000  0:00 [ vi ]

unicos$ ps -fu smm
     UID   PID  PPID    STIME TTY   TIME COMMAND
     smm   371     1 18:13:28 p001  0:00 [ csh ]
     smm   680   371 18:28:48 p001  0:00 [ ps ]
```

Communicating with users

As system operator you will frequently need to communicate with your users. There are several methods of communication available for you to use. The method to use in any specific instance is generally determined by the urgency of your message.

1. **/usr/news** - Use for normal communication to all users.
2. **/etc/motd** - Use for special or critical messages to all users.
3. **/etc/issue** - Use for critical messages.
4. **/etc/wall** - Use for emergency messages only.
5. **write, mail** - Use for communication with specific users.
6. talk - comms with users

'finger' command available.

/usr/news

The **news** command is the preferred way to get non-critical messages to users. You will either need write access to the directory **/usr/news**, or logon as superuser. To issue a news item, just create a file with a meaningful name in the directory **/usr/news**. Make sure that the file is readable by all users. On most systems, a news item will be seen by 80% of users within one week.

/etc/motd

The message of the day (**motd**) is a file that is printed out each time a user logs into the system. It is used to keep users informed of the status of the system. You will either need write permission to **/etc/motd**, or logon as superuser. The sort of messages that you should put in the **motd** file are warnings about planned shutdowns. Users can view the file at any time (e.g. `cat /etc/motd`).

/etc/wall

Use **/etc/wall** to write emergency messages on the terminals of all users currently logged in. You need to use **wall** as superuser to override message suppression on any user's terminal (i.e., `mesg n`). Do not use **wall** indiscriminately. Normally, **wall** is used to warn users of an imminent shutdown.

/etc/issue

This file is printed out every time a user logs into the system, BEFORE the login prompt. Again you will need write permission to this file, or logon as superuser. This file is not often changed. You might want to put messages here asking users not to at the moment for some reason, or warning users that the password file has been restored from last week's backup tape.

write and mail

The **write** and **mail** commands can be used to send messages to specific users. **write** puts out a message on the terminal of a user who is currently logged in, and should be restricted to more urgent messages. You need to be superuser to override message suppression. **mail** is the best way to communicate a non-urgent message to specific users.

UNICOS Startup, Shutdown, and Crashes

Starting up and shutting down the Cray system

There are two stages in starting up the UNICOS system:

1. **boot/start** the IOS-E and UNICOS operating systems
2. enter **multi-user** mode on the UNICOS operating system.

There are also two stages in shutting down the UNICOS system:

1. enter **single-user** mode on the UNICOS operating system.
2. **sync** the data from the system buffers to the disks.

Booting the IOS-E and UNICOS operating systems

The easiest way to bring up both the IOS-E and Cray operating systems is with the **bootsys** command. The `bootsys` command is a program that boots all the IOCs, IOPs, and MUX IOPs in the IOS-E. It then runs a confidence test on the mainframe before starting UNICOS on it. You should be logged onto the OWS-E with a userid of **cri** (or your operator login) to run the `bootsys` command. The full path to `bootsys` is `/home/localhost/cri/bin/bootsys`. There are a number of options on the `bootsys` command, but they are normally not needed.

```
owse% bootsys
```

You may be prompted for information, such as verifying that the date and time are correct. UNICOS uses the OWS-E's clock as a reference, so make sure it is correct.

Your system administrator will customize the files that `bootsys` references.

The next page shows typical output of running the `bootsys` command.

This page not relevant to 3D models

Output from the `bootsys` command

```
ows1601% bootsys
Analyzing the parameter file '/home/ows1601/cri/os/uts/param'.
Verifing root device accessiblity.
Verifing swap device accessiblity.
bootsys: continue with the boot ? [y/N]: y
bootsys: do you want autodumps enabled ? [y/N]: y
Booting IOPs.
  Booting cluster 0 eiop 4 with '/home/ows1601/cri/os/ios/iopmux'
  Booting cluster 0 eiop 3 with '/home/ows1601/cri/os/ios/eiop.ipi'
  Booting cluster 0 eiop 2 with '/home/ows1601/cri/os/ios/eiop.dca1'
  Booting cluster 0 eiop 1 with '/home/ows1601/cri/os/ios/eiop.bmx'
  Booting cluster 0 eiop 0 with '/home/ows1601/cri/os/ios/eiop.hybrid'
Initializing IOP channels.
  Initializing hisp 0 for cluster 0.
  Initializing interiop channel for cluster 0, eiop 0
  Initializing interiop channel for cluster 0, eiop 1
  Initializing interiop channel for cluster 0, eiop 2
  Initializing interiop channel for cluster 0, eiop 3
Initializing mainframe sequence.
INFO: mfinit: mainframe confidence test complete, 0 error(s) detected

sn1601# Unicos/6.1.4bf: sn1601-dev.19 (CRAY Y-MP)

System gen time     = 07/27/91 18:59:50
UNICOS binary size = 1087488 words
Memory Configured  = 33554176 words
Memory Allowed     = 33554176 words
sysmem             = 991544 words
maxmem             = 31350784 words
Buffer pool size   = 1024000 words (2000 buffers)
User memory avail  = 30568448 words

CPUs configured     = 2, started = 2 (0, 1)

System-Call Timing On

uts/c1/os/sched.c-11: INFO      packtime calc'd  as 8 seconds.
uts/c1/os/sched.c-07: INFO      gfactor1_in set to 155 seconds.
uts/c1/os/sched.c-08: INFO      gfactor1_out calc'd  as 155 seconds.
uts/c1/os/sched.c-09: INFO      gfactor1_out limited to 0 seconds.
Wed Aug  7 13:06:23 CDT 1991

Trying /dev/cz00...

Connected to mainframe via /dev/cz00
Routed via mux, cluster 0
Escape character is '^]'


INIT: SINGLE USER MODE


         **** CRAY Y-MP2E SN1601 **** UNICOS 6.1 ****

                   6.0 nanosecond clock
              2 - cpus 32 megawords of real memory
              and a Kernel with 32 bit full memory address

SINGLE_USER#
```

Not relevant.

Going into multi-user mode

Once UNICOS has been started, you will automatically be logged in as superuser (**root**) and the system will be in single-user mode.

To bring UNICOS into multi-user mode, you need to type the following command on the UNICOS console (window).

```
unicos# init 2
```

A large number of messages will appear on the UNICOS console, some of which will require a reply.

You may be asked again to confirm that the date and time are correct, and to correct them if they are wrong.

You may be asked if you want to check the filesystems. Always answer **y** to this question.

If you took a system dump (because the system had crashed) then you will be asked to enter the reason why you took the dump. Just type in the reason, followed by a carriage return (<cr>).

There may be extra messages specific to your site which you will have to reply to.

When the system is completely up in multi-user mode, you will see a **Console login:** prompt on the UNICOS console, and you can now log on to the system. At this stage you may have other site-specific jobs to perform such as starting the appropriate NQS job queues, configure tape drives up, and ensuring that all parts of your network are up.

Shutting down the UNICOS system

The UNICOS system is shut down using the **shutdown** script. To execute this script you need to log on to UNICOS as superuser (`root`). You should do this on the UNICOS console (`/dev/cz00`). Once logged type:

```
unicos# cd /
unicos# /etc/shutdown nn
```

where ***nn*** is the number of seconds warning you wish to give users before actually shutting down the system. The default warning period is 90 seconds. If you really need to shut down the system in a big hurry, and cannot afford to wait the 90 seconds, then `shutdown 0` gives you the minimum warning to users, but this is not good practice and you will upset the users! You should always give a reasonable warning to allow interactive users to do some housecleaning and then logout. If you give any other numeric value other than 0, that value will be added to the default 90 seconds.

As soon as you have typed in the `shutdown` command, the `wall` command runs and prompts you for a warning message which will be sent to all users who are logged on. Your message should tell them that the system is going to be shut down. If you elect not to provide a message, a default one is provided for you. A final warning is issued after the delay period. Then the system will be brought to single-user mode. This may take a few minutes, because all the subsystems need to be stopped in an orderly manner. You will see messages telling you which subsystems are being shut down. Eventually you will see a message that you are at single-user mode. To prove that you are in single-user mode type:

```
unicos# who -r
   .       run-level S  Dec 19 08:12     S    0    2
```

The **r** option shows the run-time level. In this case it is **S** for single-user mode. It also is shown in the third field from the end (**S**). The last field shows the previous run-level (**2**) for multi-user level. That is, in multi-user mode, the output of the `who-r` command would normally look something like this:

```
   .       run-level 2  Dec 19 08:12     2    0    S
```

Again, you should not play around on the system when it is in single-user mode because it is easy to get things into a mess!

NOTE: Your site may have a modified version of the `shutdown` script causing it to behave differently than described above. Check with your administrator.

See the next page for the next shutdown step.

Flushing the data from system buffers to disk

UNICOS has some special buffers called **ldcache** in the SSD or central memory where certain data is stored before being written back to disk. This provides better performance. However, that data must be flushed to disk before a restart or it will be lost. To do so you should be in single-user mode on the UNICOS console.

There are two commands to flush out the data still in these special system buffers, depending on which buffers the data is currently in. You should type in each of these commands 2 or 3 times to be sure of flushing everything. The **sync** command should come before the **ldsync** command.

```
unicos# sync          # the full path to this command is /bin/sync
unicos# sync
unicos# sync
unicos# ldsync        # the full path to this command is /etc/ldsync
```

NOTE: There is more to know about ldcache and it is discussed in the Filesystems section.

shutdown and the cz00 connection

The proper and usual way to shutdown the Cray mainframe is from the UNICOS console (`/dev/cz00`) on the OWS-E because once you have brought the system to single-user mode the only connection allowed is the one through `cz00`. Doing a shutdown from any other connection (e.g., `telnet` or another `cz` connection) may not work properly because you get logged off at some point since you are not connected through `cz00`. If you then log off UNICOS (`exit` or `^D`) and quit `zip` (`^]`, then `quit` at the `zip` prompt) NO ONE would be logged onto UNICOS. This is not a problem because you could `zip` right back to `cz00`.

What is important is if you restart the whole Cray system with the `bootsys` command, the UNICOS console reattaches itself to the window that `cz00` is currently connected to. If the connection is not open, the window the `bootsys` command was run from will automatically be connected to the mainframe through `cz00`. Therefore, if you want the UNICOS console to be in a specific window after startup, be sure to quit `cz00` and run `bootsys` from your window of choice.

Handling System Crashes

If the UNICOS or IOS-E systems crash, you, as system operator, will be responsible for making sure that a system dump has been taken. In most cases, an error message will be written to the log file `/var/logs/ioplog` or on the UNICOS console, and the background of the OWS-E graphics screen will change to red. Occasionally this may not happen, but the system will appear to be dead. If autodumping has been invoked, a dump will be generated automatically .

If your site does not choose to have autodumping enabled, you will likely have a specially prepared command to run which dumps the mainframe and/or the IOS-E. The OWS-E commands used in a script like this are **cpudump** to dump the mainframe and **edump** to dump individual IOPs. Read the man pages for them on the OWS-E for more information. Be sure to ask your system administrator if you are not familiar with your site's procedure for dumping the Cray system.

When you bring UNICOS to multi-user mode after taking a system dump, you will be asked to type in the reason for the dump. Just type in a brief one-line comment followed by a carriage return, and the restart will continue (this information can usually be found on the UNICOS console or in the `/var/logs/ioplog` file). You may also find that there are extra questions to answer while checking the filesystems. These questions are explained in the **Filesystems** section of this workbook.

If you ever need to find out times that UNICOS was booted, look at the `/etc/boot.log` file on UNICOS. It contains one line for each UNICOS system boot. An easy way to see the ten most recent entries (since they are at the bottom of the file) is to type in the following:

```
unicos$ tail /etc/boot.log
```

Another way to find out when it was last booted is with the `-b` option on the `who` command.

UNICOS system messages are also recorded in two log files worth knowing about: `/usr/adm/syslog/daylog` and `/usr/adm/syslog/kern`.

The `crash` utility

The **`crash`** utility is used to analyze core dumps. It may be of use to an operator in completing an entry in the system log after a crash. The following information can be retrieved using the **`stat`** command within `crash`. First, locate the system dump. It is usually copied to the `/core` filesystem which is then unmounted. You may have to remount it. Core files are time-stamped to help you find the right one; however, since core files are very large, your site may keep only one or two anyhow.

Output from the `crash` utility

```
unicos# cd /core
unicos# ls -lt
total 118006
-r--r-----   1 root     craysrc    68051080    Mar  4    02:38    core.0304.0238
-r--r-----   1 root     craysrc    69685384    Mar  1    20:12    core.0301.2012
-r--r-----   1 root     craysrc    70574216    Feb  28   15:03    core.0228.1503
-r--r-----   1 root     craysrc    70344840    Feb  20   12:19    core.0220.1219
-r--r-----   1 root     craysrc    68911240    Feb  15   06:18    core.0215.0618
-r--r-----   1 root     craysrc    67793032    Feb  12   01:50    core.0212.0150
-r--r-----   1 root     craysrc    67965064    Feb  9    16:26    core.0209.1626

unicos# crash core.0304.0238
> stat
crash version: Linked 02/28/91 16:27:14
          Cray X-MP/EA (ema)
          UNICOS_LVL = '6.0'

Dump file:      /core/core.0304.0238

Namelist file: /unicos

System Ident:  sysname:  sn1101     nodename: sn1101
          release:  6.0        version:  mik.2
          machine:  CRAY Y-MP

Gentime:  02/28/91 17:11:06

System Boot:    03/04/91 02:31:46
          Up for 0 hours, 1 minutes, 10 seconds

Dump taken at: 03/04/91 02:38:18

Dump Reason:    PANIC:  c1iget.c: NULL fs pointer in c1iput

Panic buffer:   PANIC:  c1iget.c: NULL fs pointer in c1iput

Panic cpu:      cpu 1

     Cpu 227944467133366272 currently in the kernel

> quit
```

Starting and stopping the system daemons with `sdaemon`

UNICOS has some very important programs called daemons which continually run on the system to do specific tasks for the system and users. Some of the main ones include the NQS (Network Queuing System) daemon, the tape daemon, and the cron daemon. There are many others and some of them will be discussed further in this workbook.

These daemons are usually started when UNICOS is brought up. Occasionally, however, these daemons may need to be stopped and later restarted. Or they may have just disappeared from the system and need to be restarted. You can see which ones are currently running by using the **`ps -e`** command (see Chapter 1 for an example of output from this command).

Stopping and starting these daemons in pre-6.0 UNICOS is fairly tedious since you need to know the exact commands (and options) to issue to start and stop the particular daemon. With UNICOS 6.0 you can simply use the **`sdaemon`** command to start or stop these daemons. `sdaemon` is located in `/etc`.

The `sdaemon` command refers to a file called `/etc/config/daemons` which contains a list of all the daemons and what command(s) to run to start and stop them. Because of this, you need only remember the name of the daemon to start or stop it. This is known as the "tag" name of the daemon (the name you supply as an argument to the `sdaemon` command).

There are many options on the `sdaemon` command, but the ones you will use most often (**`-s`** and **`-k`**) are shown below.

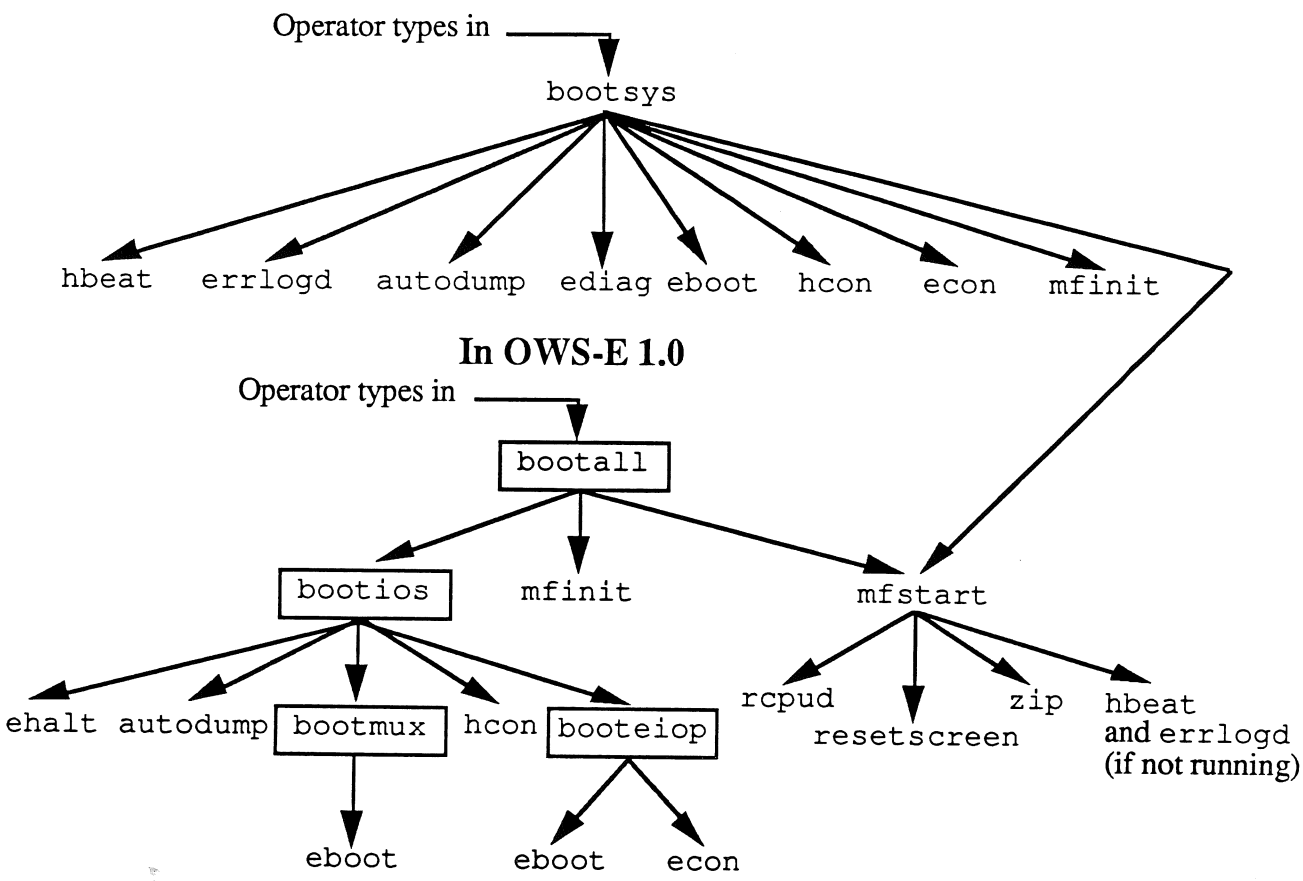
```
unicos$ sdaemon -s tpdaemon   # this starts the tape daemon

unicos$ sdaemon -k tpdaemon   # this kills (stops) the tape daemon
```

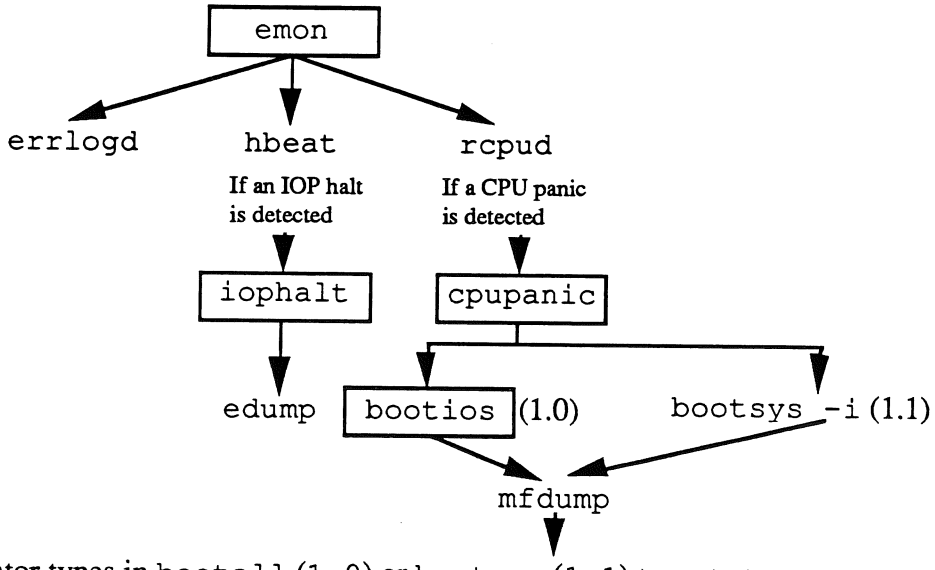
Example of the /etc/config/daemons file

```
# daemons v6.0
#
#   SN1033 - daemons - Edition 55  [Wed Feb 13 16:03:47 CST 1991]
#   Created by Configuration Generator Rev. 6.62
#
#
# Configuration file for TCP daemons (and other commands) started by
# /etc/netstart (through /etc/net/scripts/daemon).
#
# File format is:
#
# group tag              start   kill            pathname        arguments
#
SYS1    share            YES     *               /etc/shrdaemon
SYS1    share            YES     *               /etc/shradmin   -S300
SYS1    share            YES     *               /etc/shradmin   -t100 -F06 -K604
SYS1    cron             YES     *               /etc/cron
SYS1    msgdaemon        YES     /etc/msgdstop   /usr/lib/msg/msgdaemon
SYS1    dmdaemon         YES     /usr/lib/dm/dmdstop /usr/dm/dm.startup
SYS1    fsdaemon         NO      *               /etc/fsdaemon
SYS1    fsdaemon         NO      *               /etc/fsmon
TCP     gated            NO      /etc/gated.pid  /etc/gated      /usr/tmp/gated.g
TCP     named            YES     /etc/named.pid  /etc/named
TCP     inetd            YES     *               /etc/inetd
TCP     talkd            YES     *               /etc/talkd
TCP     sendmail         YES     *               /usr/lib/sendmail       -bd -q5m
TCP     printer          YES     -               /bin/rm -f /dev/printer /usr/spk
TCP     printer          YES     *               /usr/lib/lpd
TCP     snmpd            YES     *               /etc/snmpd
TCP     -                YES     -               /usr/bin/domainname     ""
TCP     portmap          YES     *               /etc/portmap
TCP     keyserv          NO      *               /etc/keyserv
TCP     ntpd             NO      *               /etc/ntpd
NFS     nfsd             YES     *               /etc/nfsd       4
NFS     cnfsd            YES     *               /etc/cnfsd      4
NFS     exportfs         YES     *               /etc/exportfs   -av
NFS     mountd           YES     *               /etc/mountd
NFS     biod             YES     *               /etc/biod       4
NFS     automount        YES     *               /etc/automount  -f /etc/auto/aum
NFS     pcnfsd           NO      *               /etc/pcnfsd
USCP    uscp             YES     /usr/lib/uscpterm /usr/lib/uscpd
SYS2    errdemon         YES     /etc/errstop    /etc/errdemon
SYS2    syslogd          YES     *               /etc/newsys     -s
SYS2    crayperfd        YES     *               /usr/lib/crayperfd
SYS2    tpdaemon         YES     /etc/tpdstop    /usr/lib/tp/tpdaemon    -rc
SYS2    FTA              NO      -               /bin/rm -f /usr/spool/fta/fqueu*
SYS2    FTA              NO      -               /usr/lib/fta    -recover
SYS2    NQS              YES     /usr/bin/qstop  /usr/bin/qstart
SYS2    cfs              YES     -               /usr/lbin/cfsinit
```

Overview of OWS-E script and command interrelationships



Operator types in emon if any of the three OWS-E daemons needs to be restarted



Operator types in bootall (1.0) or bootsys (1.1) to restart system

The Batch Subsystem (NQS)

Different methods of running in batch

Although designed as an essentially interactive operating system, the UNICOS system offers three different means by which users can schedule processes to be run in batch.

`at or batch`

The **`at`** or **`batch`** commands can be used by an operator or a user to run a process at a specified time. If the machine is not available at this time, the process will be run as soon as possible after the specified time. This type of batch processing requires no operational control.

`cron`

The **`cron`** command can be used by an operator or a user to run a process at specified dates and times. If the machine is not available at this time then the process will not be run at all. This type of batch processing requires no operational attention, but **`cron`** is a very useful tool which you might want to use for performing regularly scheduled tasks.

Network Queuing System (NQS)

The **NQS** subsystem is based on software developed by Sterling Software for NASA. **NQS** allows users to submit, terminate, monitor and control batch jobs. Operator intervention will be required to control and manage the **NQS** subsystem.

cron

Commands to be run by **cron** must be put into a **crontab** file. This file specifies which commands are to be run and when they are to be run. Each entry in the file consists of six fields separated by spaces or tabs.

minute	hour	day	month	weekday	command
(0-59)	(0-23)	(1-31)	(1-12)	(0-6) (Sunday=0)	

An asterisk (*) in any of the first five fields represents all legal values.

A comma-separated list of values in any of the first five fields represents multiple times.

An inclusive range of times can be represented by values separated by a - (e.g. **1-5**).

cron examples

```
10 * * * *  who > /tmp/whofile 2> /tmp/whoerrors
```

will report who is logged on at 10 minutes past every hour, every day of every month of every year, putting the output into **/tmp/whofile**, and any error output into **/tmp/whoerrors**.

```
0 18 * * 1-5  qmgr < /oper/nightime > /oper/night.out 2>&1
```

will run the **qmgr** command at 18:00 every Monday through Friday, reading input from file **/oper/nightime** and putting all output into file **/oper/night.out**.

To submit a crontab file to `cron` use the command **crontab myfile**, where **myfile** contains entries as described above. This will overwrite any existing **crontab** file which you may have. Therefore, as you come up with new commands to run using `cron` be sure to add them to your existing crontab file. This will ensure that the current cron commands will not be overwritten. Also, be sure to redirect input, output, and errors correctly. Any errors won't be known until after `cron` executes your commands (which could be when you are not available). Unredirected output will be mailed to you by **cron.**

NQS

NQS allows users to submit a job or **request** which can be terminated, monitored, and controlled by the user and by the operator. A request is just a shell script which has been submitted to NQS rather than being run at the user's terminal. NQS is designed to control the execution of these scripts in a batch environment since that is a more efficient use of the system's resources. For example, a job requiring a lot of memory and CPU time could be queued to run during non-peak hours.

Requests are submitted to **queues.**

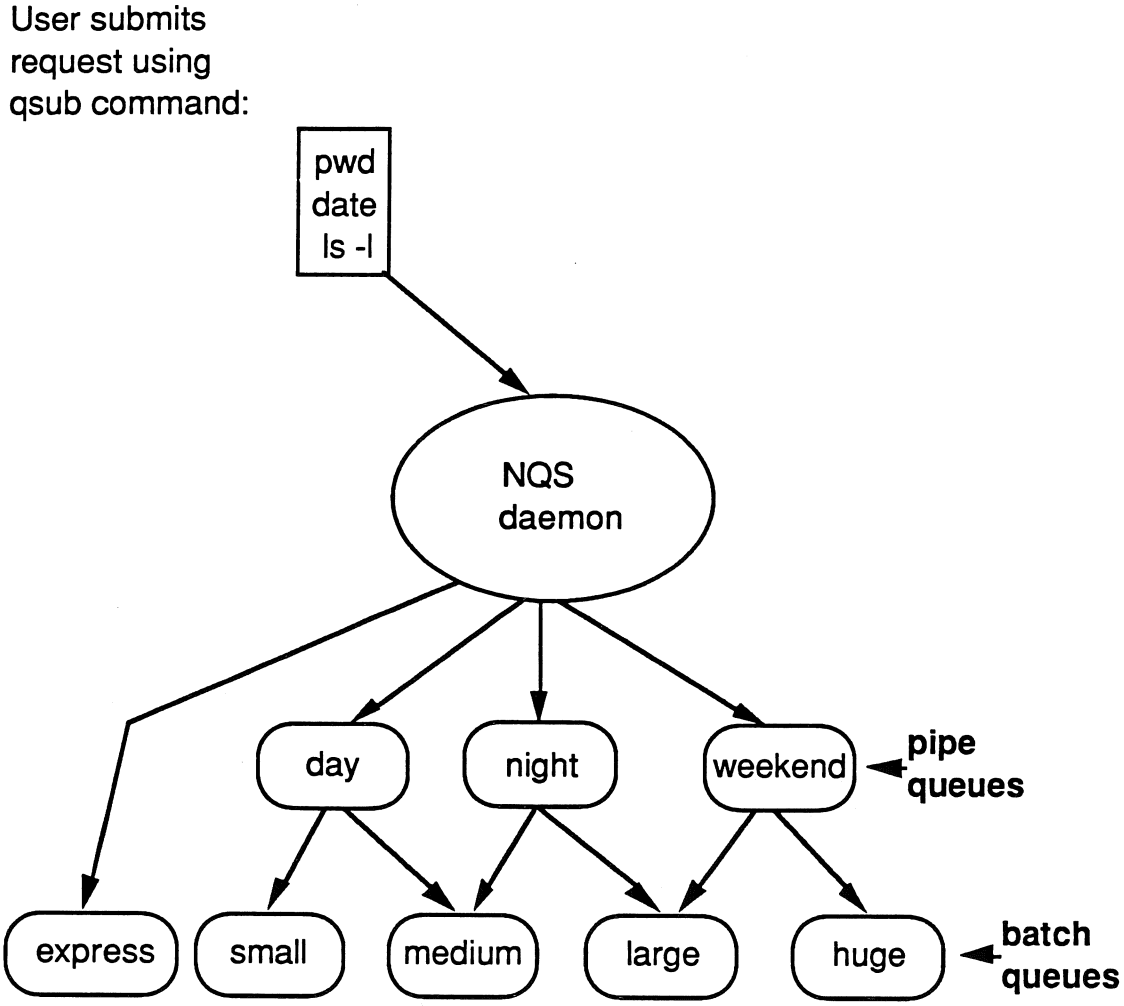
There are two types of queues - **pipe queues** and **batch queues. Pipe queues** just pass requests on to another queue. **Batch queues** actually execute the request for the user.

Some queues have higher priority than others, so requests in one queue may execute more quickly than those in other queues.

Resource limits such as time, memory, number of tapes, number of requests allowed to execute in the queue at once, etc., can be imposed on a queue and will affect all requests in that queue. Your system administrator will have set up your NQS subsystem with appropriate resource limits on the queues.

The next page shows a simple NQS queue setup with three pipe queues called day, night and weekend. There are five batch queues - express, small, medium, large, and huge. The NQS subsystem is controlled by the **NQS daemon** which will be started up when you boot the UNICOS system.

The user will use the **`qsub`** command to submit a request, and the **`qstat`** command to monitor the request. There is also a **`qdel`** command to terminate requests.


User submits
request using
qsub command:
pwd
date
ls -l
NQS
daemon
day
night
weekend
pipe
queues
express
small
medium
large
huge
batch
queues

More about NQS queues

Each queue has a **state**, which is composed of two parts.

The first part of the state of a queue indicates whether the queue is **enabled** or **disabled**. An **enabled** queue can accept (queue up) new requests. A **disabled** queue will reject (not queue up) incoming requests.

The second part of the state of the queue indicates whether or not the queue is actually processing the requests it receives. The queue can be **started**, **stopping, stopped** or **inactive.**

A batch queue can execute a request only after it has been started. Likewise, a pipe queue can pass a request on to another queue only if it has been started.

A stopped queue will just queue up all new requests until it is started. If a queue has been stopped but contains executing requests at the time, the requests will be allowed to finish. The queue is in a stopping state until they have all finished, at which time it becomes stopped.

An inactive queue is ready to process requests but is empty at the moment.

The system operator is responsible for stopping, starting, enabling and disabling queues to schedule the NQS requests in UNICOS. Queues can be stopped, started, enabled and disabled at any time when NQS is running.

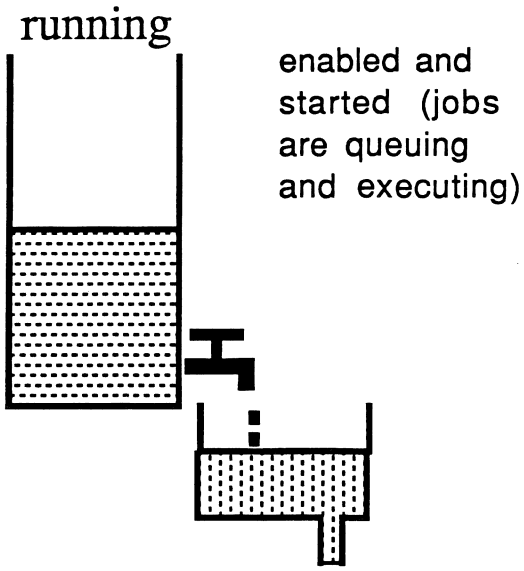
When the NQS subsystem is first started up all the queues are in a stopped condition.

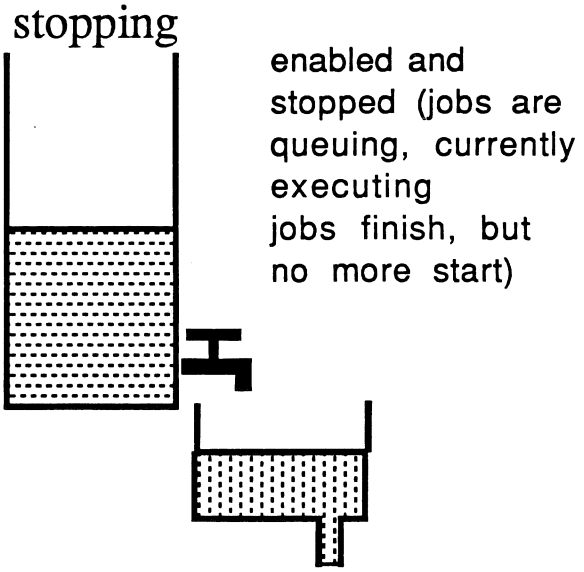
Once the system is up and running, you may need to start the appropriate queues. The particular queues which you will start will depend on how your administrator has set up NQS on your machine, and also on such things as the time of day, or perhaps the amount of work in the system. Often this job is automated for you.

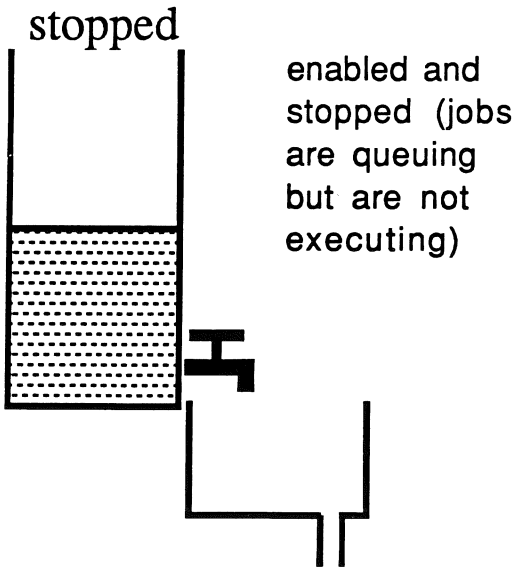
For the example on the previous page, you might need to start the day pipe queue and the small, medium and express batch queues, but leave the other queues stopped until off-peak hours.

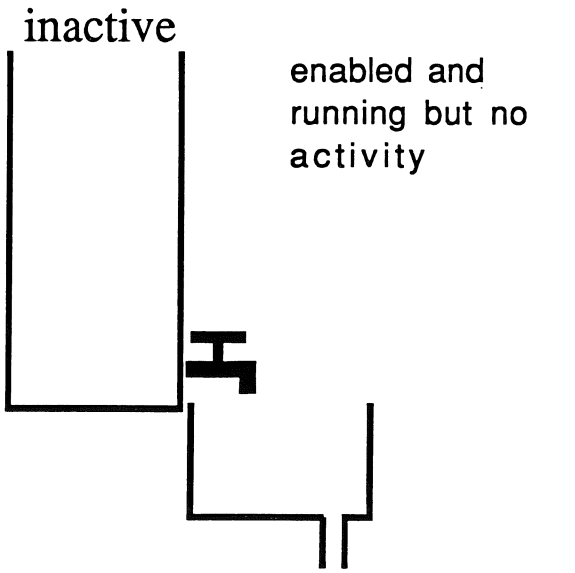
This is done through the **`qmgr`** utility. The `qmgr` utility and other NQS commands are discussed later in this section.

An NQS queue can be **running, stopping, stopped,** or **inactive.**









Using `qstat` to monitor NQS requests and queues

Users and operators can use the **`qstat`** command to monitor requests. Users will only see their own requests but the operator can see all requests.

There are several useful options on the `qstat` command:

`qstat` gives a summary for each queue (pipe and batch), telling you the state of the queue and how many requests in that queue are in each state (a **`-b`** option displays a summary of the batch queues only and a **`-p`** option displays a summary of the pipe queues only)

`qstat -a` tells you about all your requests (if regular user)
tells you about all requests (if root or operator)

`qstat -r` tells you about all your running requests (if regular user)
tells you about all running requests (if root or operator)

`qstat -f requestid` describes one request in detail

`qstat -f queuename` describes one queue in detail, including the resource limits that your system administrator has set up for that queue (more on this later)

`qstat -l` displays limits of all the queues (more on this later)

and many more. See the `man` page for `qstat` for details on options.

Examine the `qstat` example on the next page. Look at the states of each of the queues and also the resource limits imposed on those queues.

Output from the `qstat` command

```
-----------------------------------
NQS 6.0.0 BATCH QUEUE SUMMARY
-----------------------------------
QUEUE NAME       LIM TOT ENA STS QUE RUN  WAI HLD ARR EXI
---------------  --- --- --- --- --- ---  --- --- --- ---
express            1   0 yes  on   0   0    0   0   0   0
small              6   1 yes  on   0   1    0   0   0   0
medium             4   3 yes  on   1   2    0   0   0   0
large              1   1 yes off   1   0    0   0   0   0
huge               1   1 yes off   0   1    0   0   0   0
---------------  --- --- --- --- --- ---  --- --- --- ---
sn414             20   6           2   4    0   0   0   0
---------------  --- --- --- --- --- ---  --- --- --- ---
-----------------------------------
NQS 6.0.0 PIPE QUEUE SUMMARY
-----------------------------------
QUEUE NAME       LIM TOT ENA STS QUE ROU  WAI HLD ARR DEP  DESTINATIONS
---------------  --- --- --- --- --- ---  --- --- --- ---  ------------
day                10   0 yes  on   0   0    0   0   0   0  small@sn414
                                                            medium@sn414
night              10   3 yes off   3   0    0   0   0   0  medium@sn414
                                                            large@sn414
weekend            10   6 yes off   6   0    0   0   0   0  huge@sn414
---------------  --- --- --- --- --- ---  --- --- --- ---  -------------
sn414              20   9           9   0    0   0   0   0
---------------  --- --- --- --- --- ---  --- --- --- ---  -------------
```

LIM max. number of jobs that this queue is allowed to run at one time

TOT total requests currently in the queue

ENA enabled; yes indicates queue can accept requests

STS status; indicates if queue is started; `on` means that queued requests are permitted to run (or leave if a pipe queue); `off` means that requests are being queued but not allowed to run (leave)

QUE number of requests currently queued to run

RUN number of requests currently running or suspended

WAI number of requests waiting to run at a specific time

HLD number of requests that have been placed in the hold state by operator or NQS administrator

ARR number of requests currently arriving into the queue

EXI number of requests currently terminating in the batch queue

DEP number of requests currenlty departing the pipe queue

ROU number of jobs currently being routed to batch queues

More about NQS requests

Each request will be assigned a unique number called a request id as soon as it enters the NQS subsystem. The request will eventually end up in one of the batch queues where it will execute under the control of NQS. The particular batch queue the request ends up in will depend upon directives that the user may have included at the top of the file containing the request or on the qsub command used to submit the job.

The sort of directives that the user might use to influence which queue the job executes in describe the resources required by the request, such as time limit, memory limit, number of tape drives, etc. Your system administrator will have set up your NQS subsystem so that the queues have appropriate resource limits on them. NQS examines each incoming request and compares the resources required by the request with the limits on the NQS queues. The request is routed, possibly through pipe queues, into the most appropriate batch queue for execution.

Once the request has arrived in its final queue it may not begin to execute immediately if other limits have been reached. Each queue will have limits like total number of requests allowed to execute at once, total number of requests allowed to execute per user and possibly other restrictions. This means that the request may be "queued" in the queue until another request has finished. There are also global limits that apply to all queues and all requests. Global limits will be discussed later.

It is up to the user to figure out the level of resources his job requires. If too little memory is asked for, for example, the job will abort when it needs more than the limit specified (or given by default). Under normal circumstances, users do not directly choose the queue they want their jobs to run in; they are assigned based on the resource needs requested.

Simple NQS job example

```
unicos$ cat job
#QSUB -r bigjob    # request name is bigjob
#QSUB -lU 2        # request two tape drives
#QSUB -lM 50Mw     # request 50 MW of memory
#QSUB -lT 100      # request 100 cpu seconds for this job
cft77 crash.f      # compile a program
segldr crash.o     # load (link) the program
a.out              # execute the binary output of the load
```

Assuming the above lines were in a UNICOS file called **job**, the user would submit the job as follows:

```
unicos$ qsub job
Request 10763.sn1101 submitted to queue: batch.
```

Most NQS requests, however, are submitted from a computer networked to the Cray system. If the file `job` were located on a networked computer system it would be submitted with that computer system's special job submission syntax. For example, from a UNIX front-end station the command would be similar to the following (**cs** stands for cray station):

```
unix$ cs submit job
```

qstat -a

The output of **qstat -a** shows some very valuable information about user jobs. The following information refers to the example output on the next page.

IDENTIFIER Request ID# with appended name of host from which it was submitted; the ID# (without the host name) is used by many NQS commands to refer to specific requests
NAME Request name of the job take from #QSUB -r *name* ; if not specified, script name is used
USER Login name of the user who submitted the request
QUEUE Name of the queue in which the request currently resides
JID UNICOS job identification number; use this number with the `jstat` command
PRTY Priority of the request when in the queue; the nice value of the request if running
REQMEM The number of kilowords of memory the request is currently consuming
REQTIM The number of seconds of CPU time remaining for the request
ST The state of the request (see below)

At every stage in its life, a request will have an NQS **state**. You will sometimes need to examine the states of requests in the system and the queues in which they are running or queuing. The state of a request is composed of two parts, in a similar way to the state of a queue. The first part of the request state is represented by one uppercase letter, and the second part of the state is represented by two lowercase letters. Below that is a list of the more common request states that you are likely to see on your system. The man page for **qstat** shows a complete list of the request states.

First part of the request state (`ST` column, far right):
A = arriving in the NQS subsystem
D = departing from a pipe and on its way to another queue
E = exiting from the NQS subsystem
Q = queuing to be run or routed
R = running
S = suspended

Second part of the request state:
03 = the number of processes that are currently active
gr = reached global run limit
qr = reached queue run limit
qs = queue is stopped
qt = reached queue tape drive limit

Request state examples
Qqs = request is queued, because the queue it is in is **stop**ped
R02 = request is running and has 2 processes active
S04 = request is suspended and has 4 processes active
Qgr = request queued because the global run limit has been reached (i.e., global maximum number of running requests reached)
Hop = request has been held by operator

output from qstat -a

```
-----------------------------------
NQS 6.0.0 BATCH REQUEST SUMMARY
-----------------------------------
IDENTIFIER     NAME     USER     QUEUE                JID  PRTY REQMEM REQTIM ST
-------------- -------- -------- -------------------- ---- ---- ------ ------ ---
31099.sn414    testjob  tim      small                351   25     97     88 R02
31097.sn414    test     phil     medium               348   32    807    520 R04
31102.sn414    myjob    phil     medium               350   32    807    949 R02
31117.sn414    myjob2   phil     medium                     63    807   1000 Qqr
31098.sn414    test1    jan      large                      63   9216   9000 Qqs
31003.sn414    another  mary     huge                 108   63   5000  50000 S02
```

The JID# associated with a request can be used with the **jstat** command to display status of the entire job (all its current processes). The example below shows a jstat display for a request named nqstest on Cray system sn1101.

```
unicos$ qstat -a
-------------------------------
NQS 6.0.0 BATCH REQUEST SUMMARY
-------------------------------
IDENTIFIER     NAME     USER     QUEUE                JID  PRTY REQMEM REQTIM ST
-------------- -------- -------- -------------------- ---- ---- ------ ------ ---
9637.sn1101    nqstest  rjj      m_100_4@sn1101       4258   20    118    100 R02
no pipe queue entries
no device queue entries

unicos$ jstat -j 4258
  pid    status  state   utime   stime    size    addr   s-call   command
------  ------  ------  ------  ------  ------  ------  ------  ------
 95547  sleep   w-chan       0       0      66   17439  wait     sh
 95586  sleep   w-chan       0       0      53   18722  pause    sleep
```

jstat is just another way of looking at a job on a UNICOS system. Even interactive logins are considered jobs and will show up in jstat output. Notice that qstat indicates that the request is running and has two processes associated with it. jstat shows those two processes (even though they are sleeping, they are nonetheless processes taking up system resources). See the man page on jstat for more.

You could delete this user's request by doing the following:

```
unicos$ qdel -u rjj -k 9637
Request 9637.[104]@[0]
Request 9637 has been deleted.
```

Another method of deleting a request will be discussed later.

qstat -l

The `-l` option on `qstat` generates a detailed summary of the queue limits on all the queues. The limits are defined below.

RUN the maximum number of requests from the batch queue that can run at the same time. When a queue's run limit is reached, subsequent requests are still routed to the queue but remain queued until there is an available run slot.

MEMORY
The maximum amount of memory available to all requests running from the batch queue at the same time.

QUICKFL
The maximum number of Secondary Data Segments (SDS) that can be requested by all requests running in the batch queue at the same time.

USR The maximum number of requests that a particular user can run at the same time.

GRP The maximum number of requests that a particular group can run at the same time.

As shown in the example on the next page, the values under the `RUN`, `MEMORY`, and `QUICKFL` columns are in two fields, separated by a `/`. The first field indicates the maximum limit and the second field indicates the total currently allocated to that queue. A `--` in a field indicates the limit has not been specified. A `**` in a field indicates the limit is unlimited. The `MEMORY` and `QUICKFL` values are in thousands of Cray words (Kwords).

Output from `qstat -l`

```
----------------------------
NQS 6.0.0 BATCH QUEUE LIMITS
----------------------------
QUEUE NAME                RUN      MEMORY      QUICKFL    USR GRP
------------------------ --- --- ------ ------ ------ ------ --- ---
b_1000_16                 2/0      --/0         --/0       --  --
b_max_16                  2/1      --/12288     --/0       --  --
b_1000_32                 1/1      --/32768     --/0       --  --
b_max_32                  1/1      --/262144    --/0       --  --
b_max_64                  1/1      --/262144    --/0       --  --
b_max_96                  1/0      --/0         --/0       --  --
sjs_10                    1/0      --/0         --/0       --  --
sjs_100                   1/0      --/0         --/0       --  --
sjs_1000                  1/0      --/0         --/0       --  --
b_100_8                  10/0      --/0         --/0       --  --
b_1000_8                 10/1      --/879       --/0       --  --
b_max_8                  20/9      --/1590784   --/0       20  --
sjs_max                   1/0      --/0         --/0       --  --
system                    5/0      --/0         --/0       --  --
bench                    20/0      --/0         --/0       **  **
------------------------ --- --- ------ ------ ------ ------ --- ---
sn1033                   40/14     --/2161007 524288/0     10  20
------------------------ --- --- ------ ------ ------ ------ --- ---
```

qmgr

qmgr is an interactive tool for controlling the NQS queues.

To invoke qmgr, just type qmgr. qmgr will respond with a prompt. You can issue a large number of qmgr commands from within qmgr. Those commands that operators can use are noted in the man page for qmgr. To end a qmgr session, just type **exit**.

```
unicos$ qmgr
Qmgr 6.0.0: start queue small
Qmgr 6.0.0: exit
unicos$
```

You can get help at any stage while using qmgr by typing **help**.

Most of the qmgr commands can be abbreviated. The help command in qmgr will tell you the minimum abbreviation for each command. The examples in this workbook are all shown in their complete form.

You can create a file of qmgr commands and run qmgr with standard input redirected in from your file:

```
unicos$ qmgr < myfile
```

As a system operator, most of the commands which you will use within qmgr will be to control a particular queue or set of queues, or to control a single job.

Using qmgr to control queues

```
unicos$ qmgr
Qmgr 6.0.0: start queue night
Qmgr 6.0.0: start queue medium
Qmgr 6.0.0: start queue large
Qmgr 6.0.0: stop queue weekend
Qmgr 6.0.0: disable queue huge
Qmgr 6.0.0: exit
unicos$
```

The night pipe queue can now empty any requests into queues large and medium, but if any users submit a new request to the weekend pipe queue, the request will just stay in the pipe because weekend is **stop**ped. The batch queue huge will not accept any new requests but will continue to execute any existing requests.

If you decided that you wanted to **suspend** all of the huge requests now because it is the end of the weekend, this is how you would do it:

```
unicos$ qmgr
Qmgr 6.0.0: suspend queue huge
Qmgr 6.0.0: exit
unicos$
```

(You would probably also `stop` the huge queue to prevent any new requests from starting.)

`suspend`ing a queue has the affect of `suspend`ing the execution of all the requests in that queue. You cannot `suspend` a pipe queue, only a batch queue. Queue huge will remain `suspend`ed until you explicitly **resume** it.

```
unicos$ qmgr
Qmgr 6.0.0: resume queue huge
Qmgr 6.0.0: exit
unicos$
```

There are several other useful qmgr commands for controlling queues:

```
move queue queuename1 queuename2      # moves all requests in queuename1
                                      # to queuename2
purge queue queuename                 # knocks every request out of a queue
set queue run_limit = nn queuename    # sets the maximum number of
                                      # requests executing concurrently out
                                      # of this queue to nn
set queue user_limit = nn queuename   # sets the limit of how many jobs
                                      # any one user can have executing
                                      # out of this queue
start all_queues                      # starts all queues
stop all_queues                       # stops all queues
```

To examine the states of the queues in your system at any time use the qstat command with no options or use the **show queue** command within qmgr. There are many other useful qmgr commands. See the man page for qmgr.

As system operator you need to find out from your administrator how the queues are set up on your NQS subsystem. You need to discuss with your administrator which queues should be started, stopped, enabled and disabled at various times of day. You or your adminstrator might like to set up an operator crontab file to execute a set of qmgr commands at regular times of day or night. For example:

```
0 18 * * 1-5 qmgr < /oper/night > /oper/night.out 2>&1
0  8 * * 1-5 qmgr < /oper/day > /oper/day.out 2>&1
0 20 * * 5   qmgr < /oper/weekend > /oper/weekend.out 2>&1
```

The above example would run the qmgr commands in **/oper/night** at 18:00 every Monday through Friday, the qmgr commands in **/oper/day** at 08:00 every Monday through Friday, and the qmgr commands in **/oper/weekend** every Friday night at 20:00.

Don't forget that the queues would need to be set up in their correct states each time the machine was rebooted because when the **NQS daemon** first starts up, all the queues are stopped. Also, any queues which you had previously disabled, enabled, or suspended will still be in that state.

using qmgr to control requests

The previous examples have all been to control an NQS queue rather than an individual NQS request. There are several `qmgr` commands to control individual requests. Each request is allocated a unique requestid number which you will need to refer to when using `qmgr` commands to control that request. To find out the unique numbers use the `qstat` command.

```
unicos$ qmgr
Qmgr 6.0.0: suspend request 123          # freeze running request 123;
                                         # but do not free up resources
Qmgr 6.0.0: resume request 456           # unfreeze running request 456
Qmgr 6.0.0: hold request 789             # checkpoint, stop, and release
                                         # resources for request 789 if
                                         # running and restartable; if not
                                         # restartable, leave running; if
                                         # queued, make ineligible to run
Qmgr 6.0.0: release request 789          # reallocate resources to request
                                         # 798 and run it
Qmgr 6.0.0: abort request 234            # kill running request 234
Qmgr 6.0.0: delete request 345           # delete queued request 345
Qmgr 6.0.0: rerun request 789            # abort running request and requeue
Qmgr 6.0.0: move request = 988 huge      # move queued request 988 to
                                         # queue huge
Qmgr 6.0.0: exit
unicos$
```

Modifying a particular request

Ocassionally you may be asked to modify a particular parameter for a running request. This is done in `qmgr` and there are many parameters operators can modify. Several examples are shown below:

```
Qmgr 6.0.0: modify request 345 rtime_limit = 300
```

This has the effect of setting a new value to the request time limit. It does not add 300 seconds to the current value! If the request started with a limit of 200 seconds and `qstat -a` now shows 50 seconds left, the above command would effectively give the request an additional 150 seconds. 300 (new limit) - 150 (time already used) = 150 (new amount left).

```
Qmgr 6.0.0: modify request 345 rmemory_limit = 300kw
```

The above example will set a new ceiling for the amount of memory the request could ask for and get at any given time. In this case it is 300,000 Cray words (**kw**). You could also specify the amount of memory in many other formats including millions of words (**mw**). A number with no designator is assumed to be in bytes (**b**). See the Limits section of the man page on `qsub` for more.

Because the output of `qstat -a` shows the "current" amount of memory being used by the request and not the "maximum" at any one time, you should use `qstat -f` *`requestid`*. This will show the maximum available and the amount being used for both memory and time resources. Default values are surrounded by **<>**. If the values are changed, the **<>** will disappear. See the example of the `qstat -f` command on the next page. It includes a lot of valuable information about an NQS request.

Writing a message to a particular request

You can send a message to a particular nqs job using the **`qmsg`** command. You must be the owner of the nqs job or have nqs administrator privileges to use it. There are some options on the command, but the default is to take input from standard input and put it in the standard error file of the nqs job. For example:

```
unicos$ qmsg 35712
This is a message from the operator.  Your tape mount was
delayed due to an emergency on another machine.  Sorry for
any inconvenience.
^D
unicos$
```

Output from the `qstat -f` command

```
unicos$ qstat -f 10010
-----------------------------------------------
NQS 6.0.0 BATCH REQUEST: STDIN.sn1405.cray.com Status:          RUNNING
-----------------------------------------------                 2 Processes
                                                                Active
        NQS Identifier: 10010.sn1405            User:           rjj
                                                Group:          trng
        Job Identifier: 741                     Nice Value      20

        Created:        Tue Jan 29 1991         Queued:         Tue Jan 29 1991
                        09:17:43 CST                            09:17:43 CST
<QUEUE>
        Name:           b_30_1@sn1405           Priority:       30
<RESOURCES>
                                PROCESS LIMIT   REQUEST LIMIT   REQUEST USED
        CPU Time Limit               <30sec>         <30sec>            0
        Memory Size                   <1mw>           <1mw>         148kw
        Permanent File Space    <2147483647b>   unlimited
        Quick File Space                              <0>
        Type a Tape Drives                            <0>
        Type b Tape Drives                            <0>
        Type c Tape Drives                            <0>
        Type d Tape Drives                            <0>
        Type e Tape Drives                            <0>
        Type f Tape Drives                            <0>
        Type g Tape Drives                            <0>
        Type h Tape Drives                            <0>
        Execution Nice Value          <0>
<FILES>                 MODE            NAME
        Stdout:         spool
sn1405:/sn1405/mktg/techops/trng/rjj/ST0
        Stderr:         spool
sn1405:/sn1405/mktg/techops/trng/rjj/ST0
        Restart:                        <UNAVAILABLE>
<MAIL>
        Address         rjj@sn1405              When:
<MISC>
        Rerunnable      yes                     User Mask:      6077
        Restartable     yes                     Exported Vars:  basic
        Shell:          DEFAULT
        Orig. Owner:    1674@sn1405
```

Modifying priorities of a running request

There are two types of priorities you, as operator, can modify for a request. The first type is its ***intraqueue*** **priority**. That is the priority of a request while it is still queued (has not yet run). You can raise or lower this priority.

If you ever need to move a queued request to the head of a queue (so it will be the next one to run), simply **raise** its *intraqueue* priority. You will need to know what the request's priority currently is (`qstat -i` shows the priority of queued jobs - see the next page for an example).

Just modify the priority of the request to be higher than the default value for that queue (use `qstat -f` *`queuename`* to find the queue's default priority - see the previous page for an example). You only need to make the value higher than the queue default to put it at the head of the queue (assuming other requests in that queue have not had their priorities raised, too!).

```
Qmgr 6.0.0: modify request 345 priority = 40
```

Once a request is running, you may be asked to modify its **nice value**. The nice value is the value UNICOS uses as its priority scheme for running processes. You can make a request run at a lower priority (slower throughput) or a higher priority (faster throughput).

However, what's confusing is that to give a running request a ***higher*** priority, you give it a ***lower*** value (the lower the nice value, the better its priority). This is the reverse of setting intraqueue priorities.

The range of nice values for user processes is 0 (highest) to 39 (lowest) with 20 being the usual default nice value. The following example lowers a request's nice value (which raises its priority). If the default nice value is 20, it will now become 10 so will get a better priority.

```
Qmgr 6.0.0: modify request 345 nice = -10
```

To make a request run at a lower priority (slower throughput) do not use the - before the number as in the following example, which makes the nice value 30.

```
Qmgr 6.0.0: modify request 345 nice = 10
```

NOTE: It is very important that you understand that every time you modify the nice value you are changing it relative to the default value for that user (e.g., 20), not what the request is currently set to (e.g., 30)! Each user is assigned a default nice value in UNICOS. More on that later.

Output of the `qstat -i` command

```
------------------------------
NQS 6.0.0 BATCH REQUEST SUMMARY
------------------------------
IDENTIFIER     NAME     USER     QUEUE                JID  PRTY REQMEM REQTIM ST
-------------- -------- -------- -------------------- ---- ---- ------ ------ ---
11295.sn1033   dvr3d    criker   b_max_32@sn1033            31 262144   4800 Qqr
11261.sn1033   SJS10    teb      sjs_10@sn1033              31   8192     10 Qqs
11262.sn1033   SJS100   teb      sjs_100@sn1033             31   8192    100 Qqs
11263.sn1033   SJS1000  teb      sjs_1000@sn1033            31   8192   1000 Qqs
11264.sn1033   SJSMAX   teb      sjs_max@sn1033             31   8192  10000 Qqs
```

using `qmgr` to modify global limits

Your administrator will have set up resource limits on the individual NQS queues defined on your system. You, as an operator, will not be able to change these definitions. However, there are a number of **`global`** limits which you can change to control the requests in the system.

See the example of the `qmgr` **`show global`** command on the next page (you might try **`help show`** in `qmgr` to see many other NQS settings).

In the second part of the example on the following page the global limits are lowered for three categories. This could be because the system now needs to be "tuned" more to interactive use. Lowering the limit on tape-drive type a tapes could be because there is a problem with a couple of them.

These modifications will have the affect of setting a maximum of 10 requests allowed to execute at once, a total of eight type a tape drives to be allocated to any or all requests, and no single user allowed to have more than five requests executing at once.

Viewing and modifying qmgr global limits

```
unicos$ qmgr
Qmgr 6.0.0: show global

  Global batch run-limit:        40
  Global batch user run-limit:   10
  Global batch group run-limit:  20
  Global memory limit:           unspecified
  Global pipe limit:             20
  Global quick-file limit:       512mw
  Global tape-drive a limit:     10
  Global tape-drive b limit:     5
  Global tape-drive c limit:     5
  Global tape-drive d limit:     0
  Global tape-drive e limit:     0
  Global tape-drive f limit:     0
  Global tape-drive g limit:     0
  Global tape-drive h limit:     0

unicos$ qmgr
Qmgr 6.0.0: set global batch_limit = 10
Qmgr 6.0.0: set global tape_limit a = 8
Qmgr 6.0.0: set global user_limit = 5
Qmgr 6.0.0: exit
unicos$
```

NQS complex queues

NQS supports a third type of queue called a complex queue. A complex queue is not really a queue; it is a group of batch queues. Your administrator may decide not to set up any complex queues on your system, in which case you can ignore this section.

Defining a complex queue enables limits to be imposed on a group of queues rather than just one individual queue or all queues. It would allow an administrator to define a complex **big_ones** composed of queues **large** and **huge** and to set a limit on the number of executing requests in **big_ones**. It would be possible to set limits on each of the individual queues as well. For example, there could be a limit of three **large** requests and two **huge** requests, but a complex limit of four **big_ones** requests. Other resources such as number of tapes, etc., can also be imposed on a complex group of queues.

To examine complex queues on your system use the **qstat** command with a **-c** option (it will tell you if there are no complexes defined). All the columns in that output have been defined earlier in this section except for **MEM** which shows the number of member queues in this complex.

qstat -C followed by the name of a particular complex will give you details about the complex itself plus each of the batch queues that belongs to it. Sometimes you will find that requests are queuing because complex limits have been reached. These requests will have a request state that indicates this. For example, a request state of **Qcr** means that the request is queued because the complex run limit has been reached; i.e., the maximum number of executing requests in a complex has been reached and this request is queued up in a queue that is part of the complex.

To further examine user and group limits for complex queues, use **qstat -Lc** or **qstat -LC** ***complex-name***.

Examples of the output of some of these `qstat` options are shown on the next page.

Output from qstat -Lc

```
------------------------------------
NQS 6.0.0 QUEUE COMPLEX SUMMARY: sn1033
------------------------------------
```

QUEUE COMPLEX	MEM	LIM	TOT	QUE	RUN	WAI	HLD	ARR	EXI	RUN	MEMORY	QUICKFL	USR	GRP
small_memory	5	20	15	2	13	0	0	0	0	20/15	--/1603072	--/0	2	5
medium_memory	2	10	1	0	1	0	0	0	0	10/1	--/294912	--/0	2	5
large_memory	2	8	3	2	1	0	0	0	0	8/3	--/303104	--/40960	5	10

Output from qstat -C small_memory

```
------------------------------------
NQS 6.0.0 QUEUE COMPLEX: small_memory
------------------------------------
```

QUEUE NAME	LIM	TOT	ENA	STS	QUE	RUN	WAI	HLD	ARR	EXI
b_100_8	10	0	yes	on	0	0	0	0	0	0
b_1000_8	10	3	yes	on	0	3	0	0	0	0
b_max_8	20	8	yes	on	0	8	0	0	0	0
b_1000_16	2	0	yes	on	0	0	0	0	0	0
b_max_16	2	4	yes	on	2	2	0	0	0	0
small_memory	20	15			2	13	0	0	0	0

Output from qstat -C medium_memory

```
------------------------------------
NQS 6.0.0 QUEUE COMPLEX: medium_memory
------------------------------------
```

QUEUE NAME	LIM	TOT	ENA	STS	QUE	RUN	WAI	HLD	ARR	EXI
b_1000_32	1	0	yes	on	0	0	0	0	0	0
b_max_32	1	1	yes	on	0	1	0	0	0	0
medium_memory	10	1			0	1	0	0	0	0

Output from qstat -C large_memory

```
------------------------------------
NQS 6.0.0 QUEUE COMPLEX: large_memory
------------------------------------
```

QUEUE NAME	LIM	TOT	ENA	STS	QUE	RUN	WAI	HLD	ARR	EXI
b_max_64	1	3	yes	on	2	1	0	0	0	0
b_max_96	1	0	yes	on	0	0	0	0	0	0
large_memory	8	3			2	1	0	0	0	0

Shutting down and starting NQS

NQS will be shut down automatically when you shut down the system. All the currently executing requests will be **checkpointed** by NQS. This just means that NQS will save them away in a safe place so that when the NQS daemon restarts they can be **recovered** and continue to execute. Sometimes NQS cannot recover a request, so the request is **rerun** from the beginning instead.

If you want to shut down NQS at any time other than when you shut down the system there is a `qmgr` command to do it. The number after the shutdown command indicates the number of seconds to wait before shutting down NQS. It can be zero for an immediate shutdown (20 seconds is the default).

```
unicos$ qmgr
Qmgr 6.0.0: shutdown nn
Qmgr 6.0.0: exit
```

If you need to start NQS, then you need to restart the NQS daemon. **Before you do this, you should find out how the system normally starts up NQS.** It can be started a number of different ways. The original way is to use the same command as in the UNICOS startup scripts. If you use this method, the command you type in should be exactly the same as the system uses when booting.

In UNICOS 6.0 two new commands are available for starting and stopping NQS. They are **qstart** and **qstop**. If you are running UNICOS 6.0, the startup and shutdown scripts may use these commands instead of those discussed above.

As mentioned earlier in this workbook, with UNICOS 6.0 you can also use the `sdaemon` command to start or stop NQS (e.g., **sdaemon -s NQS**).

You will not normally need to stop and start the NQS daemon, but if you do, remember that you need to `start` some queues, because they will all be `stopped`. Also, don't forget that any queues or requests that you `suspended` will not have been magically `resumed` and that queues which you `enabled` or `disabled` will be in the same state as before you shut down NQS.

Many of the commands to monitor and control NQS queues and requests will not work when NQS is not running. You will get an error message telling you that the **NQS daemon is not present at the local host**.

NOTE: If you are using the UNICOS multilevel security (MLS) feature, you will have different procedures for restarting NQS. Your administrator will provide you with those procedures.

Help in qmgr

If you need help within qmgr, simply type **help** to get more information.

```
unicos$ qmgr
Qmgr 6.0.0: help

        The available commands are:

          ABort      ADd        Create     DElete     DIsable    ENable
          EXit       HElp       HOld       Initialize Lock       MODify
          MOVe       PReempt    PUrge      Quit       RELease    REMove
          RERun      RESUme     RESEt      RESTore    SEt        SHOw
          SHUTdown   SNap       STArt      STOp       SUspend    Terminate
          Unlock

        To obtain more information about a command, type:

          HElp <command-name>

        Each command keyword will always be shown as a sequence of upper
        and lowercase letters.  The uppercase prefix at the beginning of
        each keyword shows the smallest abbreviation acceptable for the
        command keyword.

        Lengthy commands can be broken across line boundaries by placing
        a backslash character: "\" at the end of a partial command line.
        For example:

Press <RETURN> to continue.
```

If you want a configuration list of all the parameters set for your NQS system, use the **SNAP** command within qmgr. Type **HELP SNAP** from within qmgr for more information on SNAP.

Other considerations for the UNICOS batch environment

As well as limits put on user jobs by NQS, there are many other resource limits which can affect a user's ability to utilize the system. These include the User Data Base (UDB), Disk Quotas, Fair Share Scheduler, and UNICOS Secure Mode.

An overview of the User Data Base follows, but you should consult your system administrator about how, or if, these limits are set up for your site. You may be asked by a user why his job did not run as expected and it may not be because of a limit set for NQS.

The UNICOS User Database

To better control users and their environments, UNICOS utilizes a **U**ser **D**ata **B**ase (UDB) to keep track of a wide variety of information and resource limits.

These limits can be the cause of a user's job not completing as expected since the limits set in the UDB override NQS limits, for example. As an operator, you need to be well aware of this and remember to consider these limits as well as those set in NQS.

The UDB scheme provides more security because general users' encrypted passwords are kept in a separate file called **/etc/udb** which is protected from group and other access. The password field in entries in **/etc/passwd** contains only an asterisk (*). The **/etc/udb** file also contains UNICOS secure mode information for users.

Another file, **/etc/udb.public** is a "shadow" file which contains only non-sensitive user information. The general user will be able to access only the public file. The user command **udbsee** allows users to view their own control parameters as well as the control parameters set for other users.

Output from the **udbsee** command

```
unicos$ udbsee -u operator
create  :operator:      uid     :9:         # logname and account id
        comment :Operator:                  # real name
        passwd  :/6jl2ZOoPvnzs:             # encrypted password
        gids    :9,     136:                # valid group ids
        acids   :9:                         # valid account ids
        dir     :/sn1405/ccn/operator:      # home directory
        shell   :/bin/sh:                   # login shell
        logline         :/dev/ttyp029:      # last line logged in on
        loghost         :ows1405.cray.com:      # last host logged in on
        logtime         :665423999:         # Fri Feb  1 09:59:59 1991
        resgrp          :5070:              # User name: CCN
        cpuquotaused    :926201.4:          # CPU time used and max (if set)
        jproclim[b]     :25:                # max processes per job (batch)
        jsdslim[b]      :1000000:           # sds limit per job (batch)
        jtapelim[b][0]  :99:                # batch tape type 0 limit
        jtapelim[b][1]  :99:                # batch tape type 1 limit
        jtapelim[b][2]  :99:                # batch tape type 2 limit
        psdslim[b]      :1000000:           # per process sds limits (batch)
        jproclim[i]     :25:                # per job max # processes (interactive)
        jsdslim[i]      :1000000:           # per job sds limit  (interactive)
        jtapelim[i][0]  :99:                # interactive tape type 0 limit
        jtapelim[i][1]  :99:                # interactive tape type 1 limit
        jtapelim[i][2]  :99:                # interactive tape type 2 limit
        psdslim[i]      :1000000:           # per process sds limit (interactive)
        shares          :100:               # user's allocated shares
        shusage         :0.000000:          # decaying accumulated costs
        shcharge        :628463259582791000.000000:   # user's accum. costs
        shextime        :665413415: # Fri Feb  1 07:03:35 1991
                                            # last time user's lnode was released
        maxcls          :16:                # max integrity class (security)
        lastlogtime     :665423999: # Fri Feb  1 09:59:59 1991
        comparts        :secadm, sysadm: # valid compartments (security)
        permits         :reclsfy, wrunlab, suidgid, lbypass:
                                            # valid permissions (security)
        valcat          :secadm:            # valid integrity categories (security)
```

The above example shows only some of the many possible fields of information which could be generated for a user, dependent upon how the user is utilizing the system and what the site specifically shows in its `udbsee` output. A complete list of all setting for a user can be seen with the **udbsee -v** command option.

For more information, see the man page for **udb** and read about **udbsee** either in its man page or in the SR-2011 manual.

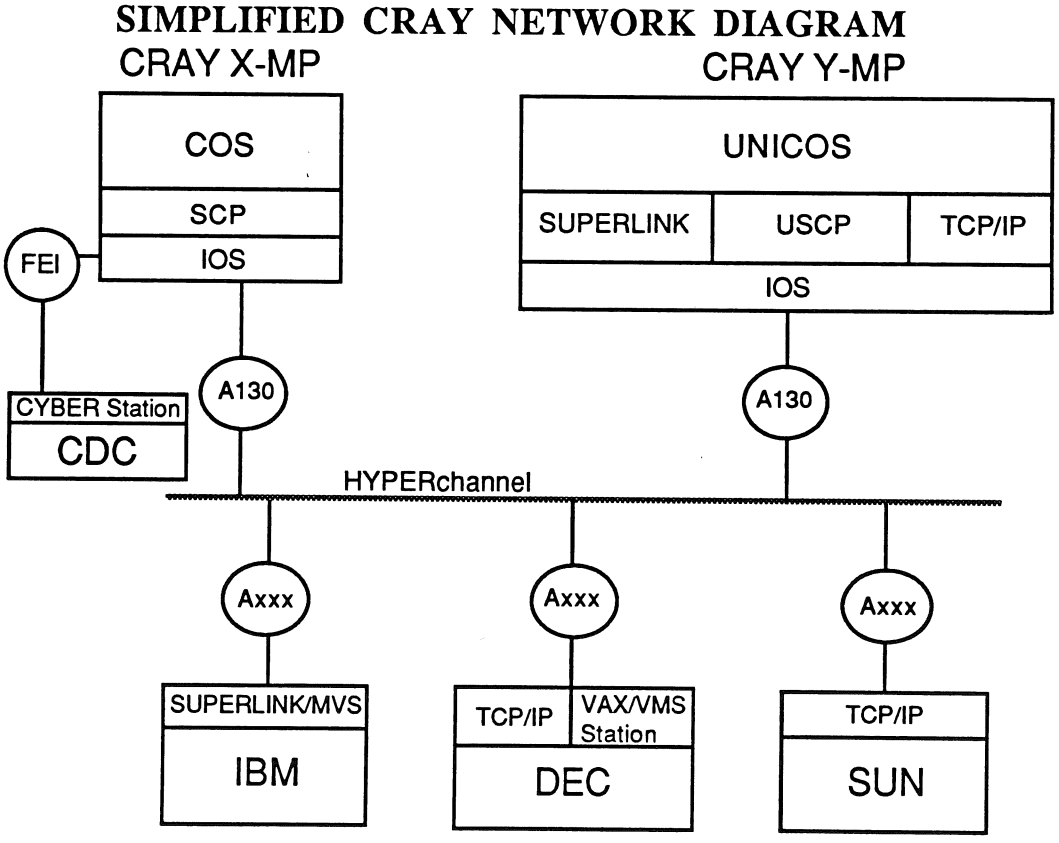
Connectivity

How users get to UNICOS

You and the system administrators are probably the only people who will log directly into the UNICOS system using the system console. All other users of the UNICOS system will log in or submit NQS requests to UNICOS from some other part of the network. These users may be connected to UNICOS via several pieces of computer and network hardware and several types of computer and network software. For this reason you need to have an accurate and up-to-date diagram of the network environment which you support. If a remote user somewhere in the network cannot connect to UNICOS, it could be any of the hardware or software at any point in the connection that is not working correctly.

There are a number of commands used to control and monitor the UNICOS part of your network. Other parts of the network will have similar commands, so get to know who the administrators are on all the systems you support.

The **hardware** connections that you need to control and monitor may include NSC, VME, FEI, and VAX-BI. The **software** that you need to control and monitor will include TCP/IP and maybe USCP and/or Superlink.



Network hardware

Network hardware such as NSC, FEI-3, FEI, and VAX-BI connect the IOS to other computers. Messages about any of these hardware connections may appear in the `/var/logs/ioplog` file (on the OWS-E) and on the UNICOS console.

These hardware connections will be configured up automatically when you start UNICOS. As system operator you may occasionally need to configure the hardware connection down and up, either to try to resolve a network problem or to disconnect the computer at the other end because it is going to be powered down.

The `nconf` command

To see which devices are attached to which channels you can use the UNICOS **`nconf`** command. This command also allows you to check the status of the channels and bring them up or down. `nconf` can be run in menu form (with refreshing screen) just by typing in the command name. It also accepts parameters to give you just the requested output in a static image (e.g., `nconf status`). The full pathname of `nconf` is `/etc/nconf`.

An example of downing and upping a channel would be:

```
unicos# nconf config 2 down
unicos# nconf config 3 up
```

The examples above configure ordinal connection 2 down and ordinal connection 3 up. Make sure you know what ordinal number is associated with what hardware (`nconf device` will show you).

NOTE: You may find that you need to configure down the network software before configuring down the network hardware because of errors that get generated on the UNICOS console window. That command is the `ifconfig` command and is discussed later in this section.

The following page shows examples of the `nconf` command.

The menu in `nconf`

```
   nconf  v. 61.0 (help)                  08/02/91 10:23:28 Frame  0

                        nconf v.61.0 COMMANDS
                        ---------------------

+                 -                 adapter           cmode             config
dbe               dfunc             dmode             dtype             device
end               exit              help              iopath            iosstat
lpath             quit              refresh           scroll            status
timeout           type

>
```

Ouput from the `device` option in `nconf`

```
 nconf  v. 61.0 (device)                  08/02/91 10:28:57 Frame  0

             device configuration parameters
ord state flags iopath cmode type  device mode func adap itmo otmo dbe
--- ----- ----- ------ ----- ----- ------ ---- ---- ---- ---- ---- ---
  0    UP    UP   0030   6MB  MP   FEI-3    0    0    0  100  100   0
  1    UP    UP   0032  12MB  PB   N130     0    0   70  100  100   0
  2    UP    UP   0034  12MB  PB   N130     0    0   c2  100  100   0
  3    UP  DOWN   0036   6MB  LCP  FEIVM    0    0    0  100  100   0
  4  DOWN  DOWN   0000   6MB  RAW  other    0    0    0    0    0   0
  5  DOWN  DOWN   0000   6MB  RAW  other    0    0    0    0    0   0
  6  DOWN  DOWN   0000   6MB  RAW  other    0    0    0    0    0   0
  7  DOWN  DOWN   0000   6MB  RAW  other    0    0    0    0    0   0
                          End of data

> device
```

Ouput from the `status` option in `nconf`

```
nconf  v. 61.0 (status)                  08/02/91 10:31:37 Frame  0

            device                              # of messages       # of errors
dev iopath cmode type device mode open     read      write      read   write
--- ------ ----- ---- ------ ---- ---- --------- ---------- ------ ------
  0   0030   6MB   MP  FEI-3    0    1         0          0      0      0
  1   0032  12MB   PB  N130     0    4     37545      38320      0      4
  2   0034  12MB   PB  N130     0    2    621660     220115      0      2
  3   0036   6MB  LCP  FEIVM    0    0         0          0      0      0
                          End of data

> status
```

Network software

Network software such as TCP/IP, USCP and Superlink connect UNICOS to other computer systems. These are the programs that control the data traveling over the network hardware just discussed. Messages from these pieces of the software will appear on the UNICOS console.

The network software is composed of daemons and software connections that are started when you bring UNICOS up in multiuser mode. As system operator you will probably need to monitor your network fairly regularly to check that all components are communicating properly. You may also occasionally need to stop and restart the daemons or configure parts of the network up and down to try to resolve a network problem.

Using the who command to monitor interactive network use

The **who** command can be used to tell you where in the network a user has logged into UNICOS from. It is useful for all types of networking software.

```
unicos$ who
mary     ttyp000  Jul 10 10:15   (sun1)
ops      ttyp002  Jul 10 08:08   (opsvax)
pete     ttyp033  Jul 10 10:17   (uscp-VX)
sarah    ttyp005  Jul 10 10:02   (SI)
```

Users mary, ops, pete and sarah have logged into UNICOS from connections called **sun1**, **opsvax**, **uscp-VX** and **SI**. These are hostnames of computers in the network. In this particular example, **sun1** and **opsvax** are TCP/IP hostnames, **uscp-VX** is a USCP connection, and **SI** is a Superlink connection. You should try to be familiar with the names and types of hostnames on your network.

The `who` command is useful if you intend to restart any part of the network hardware or software because interactive users will be disconnected from the Cray while the network is down.

If a user complains that he cannot connect to the Cray, the `who` command is also useful to see if other users are getting to the Cray from the system this user is trying to connect from. There could, however, be other reasons for connection failure.

Monitoring and controlling TCP/IP

TCP/IP can be used to communicate between UNICOS and a large number of other computer systems. You need to find out from your administrator and your local network diagrams which parts of the network are using TCP/IP, because anything you do to the TCP/IP software to try to resolve a problem on one part of the network may also affect the other parts of the TCP/IP network.

There are a number of commands that are available for monitoring and controlling your TCP/IP network.

ping	determines whether another computer is responding
netstat	gives information about network errors, routing, and current connections
ifconfig	is used to configure software interfaces up and down
netperf	is used to graphically show TCP/IP and NFS traffic (`netperf` will be discussed further in the Monitoring Performance section)

The `ping` command

`ping` is used to find out whether or not another computer in the network is responding. All you need to know is the name of the computer you want to **`ping`**, as referred to from the computer you are running the **`ping`** command on, i.e. its hostname. Messages should be issued roughly one per second, indicating that data has been sent and received. Use **`^c`** to terminate the **`ping`** command when you have seen enough. A summary is printed at the end.

For example, suppose that you are logged on to UNICOS and you want to know if Sun3 is communicating.

```
unicos$ ping Sun3
PING Sun3:56 data bytes
64 bytes from 85.0.17.5 icmp_seq=0.time=310. ms
64 bytes from 85.0.17.5 icmp_seq=1.time=8. ms
64 bytes from 85.0.17.5 icmp_seq=2.time=9. ms
...
^c
----Sun3 PING statistics----
6 packets transmitted,6 packets received,0% packet loss
round-trip (ms)  min/avg/max = 6/90/804
```

If you give an invalid hostname:

```
unicos$ ping junk
ping: unknown host junk
```

If there is no response, indicating a problem:

```
unicos$ ping Sun2
PING Sun2: 56 data bytes
^c
----Sun3 PING statistics----
9 packets transmitted,0 packets received,100% packet loss
```

The `netstat` command

The **`netstat`** command is useful for monitoring all TCP/IP connections to UNICOS. There are a large number of options on the **`netstat`** command so study the **`man`** page carefully and try some of them out. If you are investigating a network problem, run the **`netstat`** command on UNICOS, on the computer that will not communicate with UNICOS, and on all the computers in between.

A discussion of some of the more frequently used options follows.

`netstat -i`

The **`netstat -i`** display tells you about the traffic across each software interface on your system. The example on the next page has four interfaces called **hy2**, **hy3**, **hy5** and **lo0** configured up. An asterisk (*) by a name indicates the interface is down. Each interface has a network name (or number) and an address name (or number). These names will have been defined by your administrator.

The columns are defined as follows:

Name	UNICOS name for this interface
Mtu	Maximum transmission units - maximum amount of data (in bytes) allowed per packet
Network	Network name representing the network Internet address (`/etc/networks`)
Address	Host name representing the host Internet address (`/etc/hosts`)
Ipkts	Number of input packets received
Ierrs	Number of input packets errors
Opkts	Number of output packets sent
Oerrs	Number of output packet errors
Collisions	Number of network collisions on this interface

If you are investigating a problem, look to see if the number of input or output packets is changing, indicating that data is managing to get through the network. Also look to see if the number of input and output errors is increasing, indicating a problem on the network.

Output from the **netstat -i** command

```
unicos$ netstat -i
Name    Mtu   Network     Address        Ipkts   Ierrs Opkts   Oerrs Collis
hy0*    16432 none        none           0       0     0       0     0
hy1*    16432 none        none           0       0     0       0     0
hy2     16432 Sun_FEI3    sn904-sun      0       0     0       0     0
hy3     16432 Vax_bi      sn904-cuk820   0       0     0       0     0
hy4*    16432 none        none           0       0     0       0     0
hy5     16432 Hyperchan   sn904          26074   0     18589   0     0
lo0     32808 Loopback    localhost      30      0     30      0     0
```

Make sure you know exactly which pieces of network equipment each of the network and address names refers to. For example, you need to know exactly which Sun computer has the FEI-3 hardware that the Sun_FEI3 network refers to in the example above, so that you investigate the problem on the correct machine! Your system administrator should be able to provide you with this information. Request a network drawing or map from your administrator.

netstat -r

The **netstat -r** display tells you about the routing information defined in your system. This describes how UNICOS finds a computer on the network when it needs to communicate.

Each destination machine to which a route is defined will have an entry in this table. The Gateway is the name of the machine closest to the Cray through which the data is routed. In the example on the next page, data for Sun-FEI3 will be routed through the gateway called sn904-sun. Check that you understand which piece of equipment all the destination and gateway names on your machine are referring to.

Your administrator may have defined a destination called **default** on your system. All this means is that any data for a computer that is not specified in the routing tables should just be sent to the destination default. In the example above, such data would be routed via the gateway **uksun4-hy**. The interface names in the far right column are the same as in **netstat -i**.

You should not need to change the routing information in your network. However, it is useful to run **netstat -r** at each end of a broken connection and on various machines in the middle to try to isolate the problem down to two adjacent computers in the network The most normal cause for routing problems is that part of the network has a hardware or software problem (e.g., a remote routing computer is down) or an administrator somewhere on the network has fiddled with the routing table information!

Run **netstat -r** on your system so that you know what the output should look like when the network is working properly. See the **man** page on **netstat** for more details.

netstat -m

The **-m** option on `netstat` will show you statistics on the use of pre-allocated memory buffers (mbufs). These are important because if they are all being used, no one else will be able to connect to the Cray system. The most important lines in the output are the first and last. The first line shows the ratio of the number of mbufs being used to the number available. If they are equal, all are in use. If that is the case you may also see a non-zero value in the bottom line indicating how many requests were denied because no mbufs were available.

Output from the **netstat -r** command

```
unicos$ netstat -r
Routing tables
Destination      Gateway           Flags    Refcnt Use        Interface
localhost        localhost         UH       2      30         lo0
default          uksun4-hy         UG       14     19479      hy5
Vax_bi           sn904-cuk820      U        0      0          hy3
Sun_FEI3         sn904-sun         U        0      0          hy2
Hyperchan        sn904             U        0      122        hy5
```

Destination Name of the network or host

Gateway Name of the host through which the packet will be routed in order to reach its destination

Flags Several possible; U means the interface to the destination is up
H means the destination is a host-specific route
G means the destination host is acting as a gateway

Refcnt Number of current sessions

Use Number of packets (cumulative)

Interface UNICOS name for this interface

Output from the **netstat -m** command

```
unicos$ netstat -m
2216/4000 mbufs in use, allocated as 534 mbuf clusters:
        104 mbufs allocated to data (263 maximum)
        0 mbufs allocated to packet headers (80 maximum)
        143 mbufs allocated to socket structures (243 maximum)
        143 mbufs allocated to protocol control blocks (243 maximum)
        68 mbufs allocated to routing table entries (68 maximum)
        0 mbufs allocated to fragment reassembly queue headers (2 maximun
        1 mbufs allocated to socket names and addresses (7 maximum)
        0 mbufs allocated to socket options (1 maximum)
        3 mbufs allocated to interface addresses (3 maximum)
        0 mbufs allocated to NFS notify-when-free data areas (2 maximum)
        1623 mbufs allocated to NFS static data (1707 maximum)
        125 mbufs allocated to NFS dynamic data (125 maximum)
        6 mbufs allocated to system buffer headers (26 maximum)
0 requests for memory denied
```

If you need to know more about managing a TCP/IP network, talk to your UNICOS administrator and read the TCP/IP Network Administrator's Guide (SG-2021).

The ifconfig command

ifconfig is used to configure the software connections to the network up and down. You should use **ifconfig** to configure part of the network software down if you intend to configure that part of the network hardware down. For example, if you want to power down a Sun that is attached to the Cray via a VME hardware connection, you should down the network software using **ifconfig**, then down the hardware connection using **nconf**. You may also need to configure the network software down and up to try to resolve a network problem.

In order to use the **ifconfig** command you need to know the interface you are configuring up or down and address that the interface connects to. You can get this information from the **netstat -i** display or by asking your administrator. Make sure you know the correct parameters for the **ifconfig** command for all interfaces in your network before you need to use the command!

Using the example above, suppose you wanted to configure the connection to the Sun_FEI3 network down. Here is the command you should use:

```
unicos# ifconfig hy2 down
```

This will configure interface **hy2** down (which is the connection to the Sun_FEI3 network).

To configure it back up is very similar.

```
unicos# ifconfig hy2 up
```

To find out whether the interface **hy2** is currently up or down:

```
unicos$ ifconfig hy2
hy2: flags=61<UP,NOTRAILERS,RUNNING> iftype hy inet 223.255.12.4
netmask ffffff00
```

You need to be superuser (i.e., root) to configure interfaces up and down but not to just look at the current status.

There are a number of other parameters for the **ifconfig** command that you may need to use, especially if you are using the command on other machines in the network rather than just the Cray. Talk to the administrators of these systems. See the **ifconfig man** page for more details.

Monitoring and controlling Superlink

You may be using Superlink to communicate between UNICOS and one or more IBM systems. You need to find out from your administrator and your local network diagrams which parts of the network are using Superlink, because anything you do to the Superlink software to try to resolve a problem on one part of the network may also affect the other parts of the Superlink network.

There are a number of commands that are available for monitoring and controlling the Superlink network on UNICOS.

slstart	starts the Superlink daemons
slstop	stops the Superlink daemons
qmgr	starts and stops the NQS Superlink daemon
sltoffers	reports on Superlink status
sldevs	reports Superlink network connections
sltconns	reports on the low-level state of the network

Starting and stopping the Superlink network

The Superlink software is composed of several daemons, plus a superlink-NQS daemon which communicates between Superlink and NQS. All the daemons will be automatically started when you **init 2**; they will be stopped when you run **shutdown**. In addition, most of the Superlink daemons will automatically restart themselves if they crash for any reason. However, if you ever need to stop it and restart all of Superlink manually for some reason, you can do it using the commands **slstop** and **slstart**.

```
unicos# slstop
SLSTOP  SUPERLINK shutdown started.
SLSTOP  Waiting for shutdown to complete.
SLSTOP  Unmounted '/slfs/bdc'.
SLSTOP  SUPERLINK shutdown complete.
unicos$

unicos# slstart
SLSTART  SUPERLINK startup: Tue Dec  5 12:13:58 1989
SLSTART  Specification file:    /usr/spool/sl/slspec
SLSTART  Logfile directory:     /usr/spool/sl
SLSTART  Logfile name:      /usr/spool/sl/SLD0512135800
SLSTART  Error file name:   /usr/spool/sl/SED0512135800
SLSTART  Logfile opened successfully
SLSTART  Error file created successfully
SLSTART  Starting SUPERLINK/UNICOS version 2.2.0.
SLSTART  Processing specification file...
SLSTART  Specification file validated.
SLSTART  INFO: there are 4 corefiles in /usr/spool/sl.
SLSTART  Please remove any which are no longer required
SLSTART  Started Management Process, process id = 20611
SLSTART  SUPERLINK startup completed.
unicos#
```

To use **slstop** and **slstart** you need to be logged on to UNICOS as superuser.

If you are running UNICOS 6.0 you might be able to restart these daemons using the `sdaemon` command. Ask your administrator for the proper procedure.

NOTE: If you are using the UNICOS multilevel security (MLS) feature, you will have different procedures for restarting Superlink. Your administrator will provide you with those procedures.

Using qmgr with Superlink

sltop and **slstart** are used to stop and start all parts of Superlink running on UNICOS except the Superlink-NQS daemon. This part has to be stopped and started separately using **qmgr**.

If you have restarted Superlink you may also need to restart the NQS Superlink daemon. To do this you need to use the **qmgr** command **initialize superlink**.

```
unicos$ qmgr
Qmgr 6.0.0: init super
Qmgr 6.0.0: quit
unicos$
```

If you need to stop the NQS Superlink daemon for some reason, then you would use the command **terminate superlink**.

```
unicos$ qmgr
Qmgr 6.0.0: term super
Qmgr 6.0.0: quit
unicos$
```

It does no harm to initialize the NQS Superlink daemon when it is already running, you will just get a **qmgr** error message:

```
NQS manager[TCML_ALREADEXI]: Already exists at local host.
```

The **sltoffers** command

The **sltoffers** command can be used to check that all the components of Superlink are running on UNICOS. The normal output of the command will look similar to this:

```
unicos$ sltoffers
Max queue   Curr queue   TSAP-ID size   Avail TSAP-ID
        1             0              8     Yes   SLVTSLIA
        2             0              5     Yes   SLMIR
        5             0              5     Yes   AACJS
        5             0              3     Yes   NQS
```

The four lines of output may not always be in the same order, but should always be present. If there is no line for NQS this means that the Superlink daemon which talks to NQS is not running. You need to restart it using **qmgr** as described earlier. If any of the other three are not present, the simplest solution is to **slstop** and then **slstart** all of Superlink. Rerun **sltoffers** after you have done this to verify that all components of Superlink are now present.

If Superlink is not running at all on UNICOS, you will see the following output:

```
unicos$ sltoffers
SLTOFFERS  SUPERLINK is not active.
```

The sldevs command

The **sldevs** command reports the status of Superlink network hardware.

```
unicos$ sldevs
    NAME      TYPE MODE STA  SIZE  I/O   IN    OUT   RDS WRS QSZ FLGS  COMMENT
omm/n36/lp02 NSC   FDX RDY 64000    2 0.000 0.000   1   1   0 B   OPERATIONAL
```

The most relevant information is the name and type of connection, which indicates which physical connection is being described; the status (**STA**); the flags (**FLGS**) and the **COMMENT**. In the example above there is just one hardware device in the Superlink network. The type of the device is **NSC**, and the name indicates that it is on channel number **36** of the MIOP (remember that all network devices are always connected to the MIOP). This connection has a status of **RDY** (ready) and is **OPERATIONAL**. The flag **B** indicates that the connection is busy, and the **I/O** field shows **2** I/O operations are active - a read (**RDS**) and a write (**WRS**).

If there were some sort of problem with this Superlink interface, you might see other values in the flags and comment fields.

Possible flags include:

B (busy)
T (timeout)
C (closed, indicating a device error)

Possible comments include:

OPERATIONAL	(active and available for use)
WRITES FAILING	
NOT READING	
WAIT-SYNC	(remote computer is not reading/writing to device)
CLOSING DOWN	(a catastrophic device error has occurred, device is shut down until Superlink is restarted)

Make sure you know what **NAMES** and **TYPES** of Superlink connections are on your system, so that you know which device is being referred to by each line of the output from the `sldevs` command.

See the **man** page for **sldevs** for a detailed description of all fields.

The sltconns command

The **sltconns** command is useful for finding out if there is any activity at all on the Superlink network. It will report information about the lowest level of the networking software, and for this reason is rather cryptic. However, you might want to use it to check that there is no activity before shutting down the Superlink network for some reason.

This is what the output looks like on a completely idle Superlink network:

```
unicos$ sltconns
LTSAP-ID RTSAP-ID  TPDU IREF TREF MY-TU YR-TU MCD YCD   DT-S   DT-R R AD

                There are no active transport connections.
```

On a Superlink network where there is some activity you would see at least one entry in the output. Each line represents one connection into UNICOS, such as an interactive login connection, an NQS batch job submitted via Superlink, or a file transfer.

```
unicos$ sltconns
LTSAP-ID RTSAP-ID  TPDU IREF TREF MY-TU YR-TU MCD YCD   DT-S   DT-R R AD
_LTSAP54           4096 0406 5406    77    20   1   4    77     20    SI
```

The right hand column (**AD**) displays the Superlink hostname of the computer at the other end of this connection. This is the same as the Superlink hostname that appears in the output of the **who** command if the user is logged in via the Superlink network.

```
unicos$ who | grep SI
pete     ttyp003  Jul 10 10:17    (SI)
```

Operating Superlink on other computers in the network

As well as operating the UNICOS side of the Superlink network, you will probably be involved with operating Superlink on other computers in the network. Superlink runs on an IBM MVS system which is directly attached to the Cray via a network hardware device. Superlink may also run on remote IBM MVS systems connected to the attached IBM via VTAM. There are three MVS commands you will need to use to operate Superlink: **START**, **STOP** and **CANCEL**. These commands are documented in section 2 "Starting, Stopping and Cancelling SUPERLINK under MVS" of the Superlink Operators Guide (Cray publication SI-0192). There are also a number of Superlink MVS commands which are used to monitor and control Superlink on MVS. These are documented in section 3 "SUPERLINK MVS commands" of the same manual.

Make sure you are familiar with these two sections of the Superlink Operators Guide before you begin to operate the MVS side of Superlink.

Monitoring and controlling USCP

USCP is the **U**NICOS **S**tation **C**all **P**rocessor. It is made up of programs that run on the Cray which communicate with other computer systems through a point-to-point connection (front-end interface) or a network connection (e.g. HYPERchannel). The software that runs on the front-end is specific to that machine and is called "station", such as VMS station.

USCP consists of two daemons which are automatically started when you go to multiuser mode (**`init 2`**); they are stopped when you **`shutdown`** UNICOS.

Starting and stopping USCP

If you ever need to stop and restart USCP, use the following commands:

```
unicos# uscpterm      (stops USCP)
unicos# uscpd         (starts USCP)
```

You need to logon to UNICOS as superuser to stop or start USCP.

If you are running UNICOS 6.0 you might be able to restart the USCP daemon using the `sdaemon` command. Ask your administrator for the proper procedure.

All the USCP operator commands discussed in this section are in the directory **`/usr/lib`**. If you are going to be operating USCP, you will want to put **`/usr/lib`** into your **`PATH`** so that you do not have to keep typing **`/usr/lib/`*`command`***.

NOTE: If you are using the UNICOS multilevel security (MLS) feature, you will have different procedures for restarting USCP. Your administrator will provide you with those procedures.

The `uscplink` command

Many types of computer systems can connect to a USCP network by running a piece of software called a **station**. The station "logs on" to USCP so that users can access the connection to UNICOS. Each station has a unique mainframe id which is similar to a hostname.

The **`uscplink`** command shows you which of the possible stations are logged on to USCP.

```
unicos$ uscplink
mf state lmn oqc cis cos mis mos mas nssg  sssz fd ord
-- ----- --- --- --- --- --- --- --- ---- ----- -- ---

M2  on   145   0   0   0   4   4   5    1   512 15   2
AX  off
M6  on    44   0   0   0   4   4   5    1   512 16   6
PG  off
VX  on    60   0   0   0   2   2   4    1   512 16  13
```

The two columns of interest are the mainframe id (**`mf`**) and the **`state`**.

In the above example there are five stations in the USCP network, but only **M2**, **M6**, and **VX** are currently logged on to USCP. This may indicate a problem in the network that needs investigating, or perhaps the machines running stations **AX** and **PG** are down for maintenance.

The mainframe id is the same as the name that appears in the hostname field in the output from the **`who`** command, except that it is prefixed by **`uscp-`**.

```
unicos$ who | grep uscp
pete     ttyp033  Jul 10 10:17    (uscp-VX)
sarah    ttyp039  Jul 10 10:02    (uscp-M2)
```

For each computer in your USCP network you need to be familiar with the mainframe id, (in the example above you would need to know which VAX runs the station called **`VX`**). You may also need to know how to operate the station on each system. There are operational guides for all Cray stations, so talk to the administrator of the systems in question. As a minimum you need to know how to logon, logoff and examine the status of each station.

The uscpping command

You can also use the **uscpping** command (UNICOS 6.0) to test a connection to USCP. If there is no message, the station connection is up; otherwise, you will get an error message. Note that it does not function exactly like the `ping` command. `uscpping` is located in the `/etc` directory.

```
unicos$ uscpping -m M2
unicos$ uscpping -m PG
uscp-11 uscpping: WARNING
  Station 'PG' is not logged on to USCP.
```

The uscpstat command

The **uscpstat** command gives a general status of everything that USCP is doing. `uscpstat` will display **ALL** jobs submitted to NQS (via batch station, interactive station, and TCP/IP). The columns of output are defined below. You will notice that some of them correspond directly to the output of the `qstat -a` and `qstat -r` commands.

Jobname	the name of the NQS job
jsq	the USCP job sequence number (the number will be 0 for TCP/IP jobs)
Nqsid	the NQS request id number
Pid	the job's current process id, 0 if not running (queued)
Q	the queued status; several are possible (`E` = execute, `O` = output)
DC	the disposition code of the job; e.g., `IN` means it has been input to the Cray, `ST` = staged to front-end, `PR` = disposed to a printer; others are possible
Status	Current status of job in Cray; `QUEUED`, `RUN`, and others possible
Class	the NQS queue the job is in
Pri	the job's priority
FL	the job's current field length (amount of memory being used) in 512-word clicks
CPU	number of accumulated CPU seconds; i.e., time used so far (`qstat` shows time left)
LIMIT	the maximum time allowed for this request
MF	mainframe identifier; if `??`, the job (request) was not submitted to NQS through USCP
TID	terminal ID (the user's id on the remote computer system)

Output from the `uscpstat` command

```
unicos$ uscpstat

Jobname J s q Nqsid P i d Q DC Status   Class   Pri   FL   CPU  Limit MF TID
------- ----- ----- ----- - -- ------- ------- --- ----- ----- ----- -- --------
cfsroot  6145 21071     0 E IN waiting  orphan   0     0     0     0 M4    U0002
cfs____ 13321 13321     0 E IN QUEUED   USCP     0     0     0     0 M4    U0002
sjs.job     0 25297     0 E IN QUEUED   sjs_max 20     0     0  1500 ??      djp
imsl.st     0 25448     0 E IN QUEUED   b_max_1 20     0     0  3500 ?? cft77grp
imsl.sf     0 25447     0 E IN QUEUED   b_max_1 20     0     0  3500 ?? cft77grp
imsl.ma     0 25446     0 E IN QUEUED   b_max_1 20     0     0  3500 ?? cft77grp
runv.ad     0 25445     0 E IN QUEUED   b_max_1 20     0     0  1500 ?? cft77grp
runv.na     0 25444     0 E IN QUEUED   b_max_1 20     0     0  5500 ?? cft77grp
co3.run     0 25443     0 E IN QUEUED   b_max_1 20     0     0  1500 ?? cft77grp
 VERIFY     0 17303     0 E IN QUEUED   b_max_1 20     0     0  4784 ??      tam
 VERIFY     0 17302     0 E IN QUEUED   b_max_1 20     0     0  3655 ??     pags
rt8.jcl     0 25440     0 E IN QUEUED   b_max_1 20     0     0  9000 ??    c1394
  YORTP     0 25439 77262 E IN RUN      b_max_1 20 25213   484 10999 ??  taghavi
brian12     0 25438 13606 E IN RUN      b_max_1 20  8537  4439 25000 ??     sasi
 GCAL70     0 17306 86778 E IN RUN      b_1000_ 20   367    10   750 ??   cftgrp
   ad10  6166 10684 18746 E IN RUN      b_600_1 20   198     2   100 YL AAAAAPwA
ub.ad10  6195     0 20555 O PR    hold ------- 0     0     0     0 YL AAAAAPwA
```

The uscpstrs command

uscpstrs displays the file transfer status in progress in the USCP network.

```
unicos$ uscpstrs
mf p i d ojsq f i l e n a m e j s q dc fm stn mbits/s current   total duration
-- ----- ---- -------------- ----- -- -- --- ------- ------- ------- --------
M2      0 1942 fort.out         2080 MT TR   1  0.0000       0    9208 00:05:44
VX   3256 2075 testout          2090 ST UD   1  0.0430      60     106 00:00:45
AX   8432 5654 infile           5655 ST TR   1  2.5688     221       0 00:00:02

                                                ------- ------- -------
                                 t o t a l s  2.6118     281    9314
```

mf is the mainframe or name of the station which this transfer is to/from. The filename is the name of the file on UNICOS which is being sent/received and this is the most useful information for identifying a particular transfer when talking to users.

The **dc** field is the **d**isposition **c**ode, which tells you what sort of media the station is accessing: **MT**=tape-drive, **ST**=data is being staged to the front-end disk, **PR**=printer.

mbits/s tells you the data transfer rate in megabits per second. **current** is the number of blocks transferred so far.

total is the number of blocks in the file (zero for the file that USCP is receiving in this example) since USCP does not know how big it is until it has arrived. Duration is the number of hours, minutes, and seconds since the transfer was first started.

In the example above there are three transfers active - to stations **M2**, **VX**, and **AX**, for files called **fort.out**, **testout**, and **infile**. The top transfer has not begun yet (current is 0), and has been stuck for over five minutes. In this particular case, the transfer is stuck because it is waiting for a tape mount (the **dc** is **MT**) on the computer running the **M2** station.. The other two transfers are progressing.

The `uscpops` command

The only other command you may need to operate USCP is the **`uscpops`** command. This command has options which allow you to see exactly the same output as the **`uscplink`**, **`uscpstat`**, and **`uscpstrs`** displays, but has other uses as well:

`kill`	kills an interactive session, batch job, or file transfer; any output is discarded by USCP
`drop`	drops an interactive session, batch job, or file transfer; any accumulated output is returned to the front-end (same as kill except that the user gets his "output so far")
`addmf`	Adds (ups) a station ID to USCP's current configuration so it can logon
`delmf`	Deletes (downs) a station ID from USCP's current configuration so that it cannot logon
`resub`	Forces USCP to resubmit any batch jobs that USCP queued internally due to NQS being down (UNICOS 6.0)

There are other options as well. See the `man` page for the **`uscpops`** command for further information.

If you type any of the USCP display commands you will get a message **`USCP not running`** if the USCP daemons are not there. In this case you would need to restart the USCP daemons and report the problem to your UNICOS administrator.

The System Network Monitor (xnetmon)

xnetmon is a new network monitoring command available with UNICOS 6.0. From a main window you will be able to generate other windows to view the output of most of the TCP/IP and USCP commands discussed in this section. It also includes many other built-in commands designed more for network administrators.

The `xnetmon` command (run via a remote shell) can be added to the list of commands in your pull-down menu on the OWS for ease of execution. The full name of the command is `/usr/lib/monitor/xnetmon`.

`xnetmon` will be discussed further in the Monitoring Performance section.

See the man page for `xnetmon` or the UNICOS Administrator Commands Reference Manual (SR-2022).

Network problem resolution in general

Networking problems can often be complicated to resolve, due to the fact that there is a wide variety of software and hardware involved. Here are some very general guidelines to follow:

Don't power down machines without configuring directly attached hardware connections to them down also

Don't reset network hardware without configuring the hardware connections down at each end

Don't configure a hardware connection down without first configuring the software down on that section of the network

Always have an up-to-date diagram of your network, including names of each piece of hardware and software and all hostnames

Examine network problems from all computers that could possibly be involved by using the tools provided

Try to make sure that all administrators on your systems tell you when they make changes to the network!

Filesystems

UNICOS file system organization

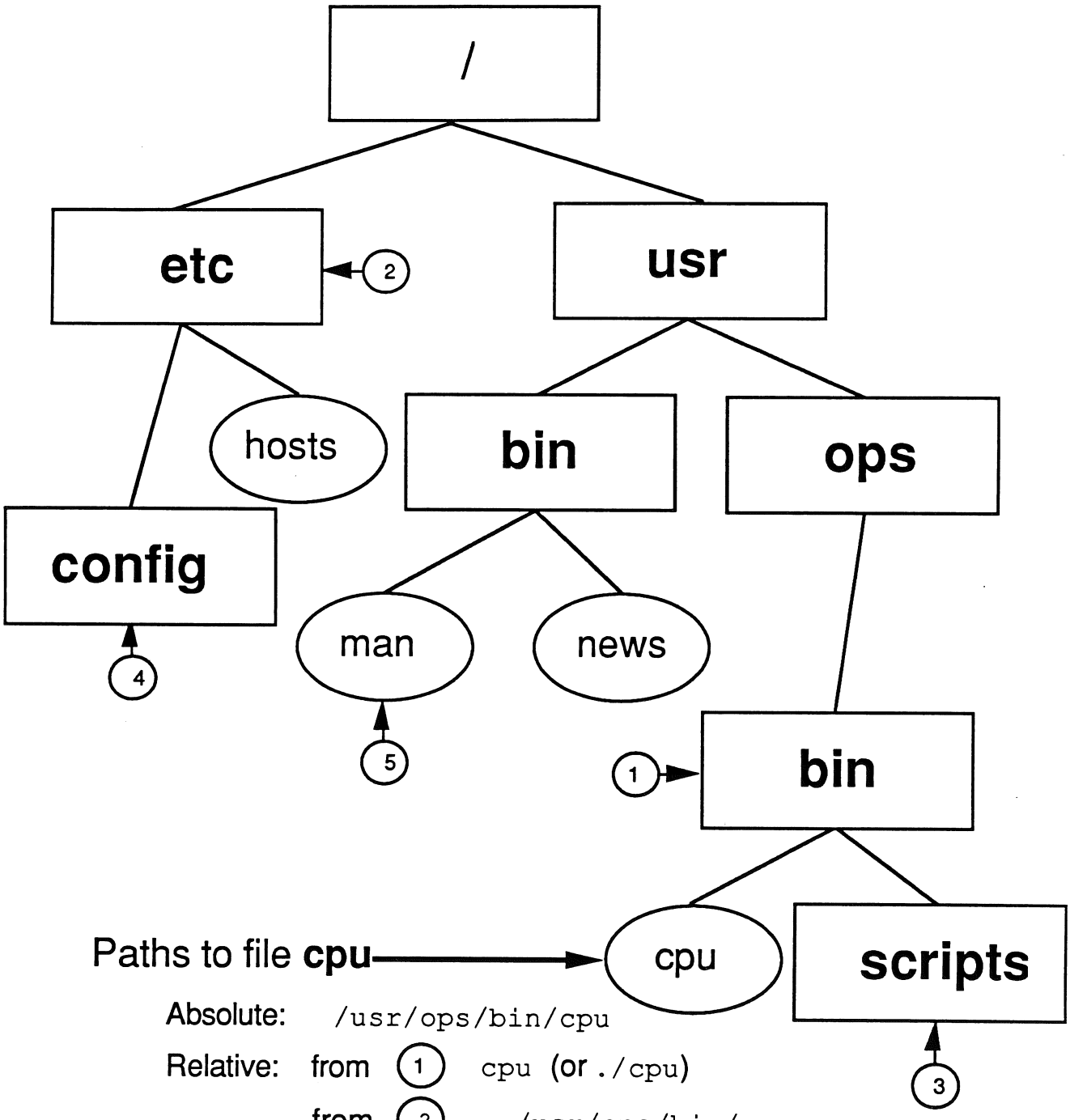
The UNICOS file system is organized hierarchically, like and organization chart. This organization is a natural way of organizing information, similar to the way files are organized in an office environment.

The UNICOS file system can be thought of as an inverted tree, with a root at the top, directory files as branches, and ordinary files as leaves. With this inverted tree analogy, branches may split into other branches so directories can contain other directories. Every directory has a parent directory, except of course the root directory. To locate a file, you can trace the path from the root through the branches to the leaf (file).

Files and directories

Any file or directory in the inverted tree-like structure can be referenced by its **PATH** through the tree structure. The example shows a number of different ways of referring to the file **cpu**.

PATHNAMES



Paths to file **cpu**

Absolute: `/usr/ops/bin/cpu`

Relative: from (1) `cpu` (or `./cpu`)

from (2) `../usr/ops/bin/cpu`

from (3) `../cpu`

from (4) `../../usr/ops/bin/cpu`

from (5) `../../ops/bin/cpu`

Filesystems

Once the system is fully running it looks as if there is just one big tree structure containing all files and all directories, but actually the big tree is composed of several subtrees. When you first start up the UNICOS system to single-user level only the top of the tree is there and none of the subtrees can be accessed at all.

During the process of going to multiuser mode at startup time the system will check that each subtree is properly structured and join it onto the big tree at the appropriate branch to build the whole big tree.

Each subtree is called a **filesystem.**

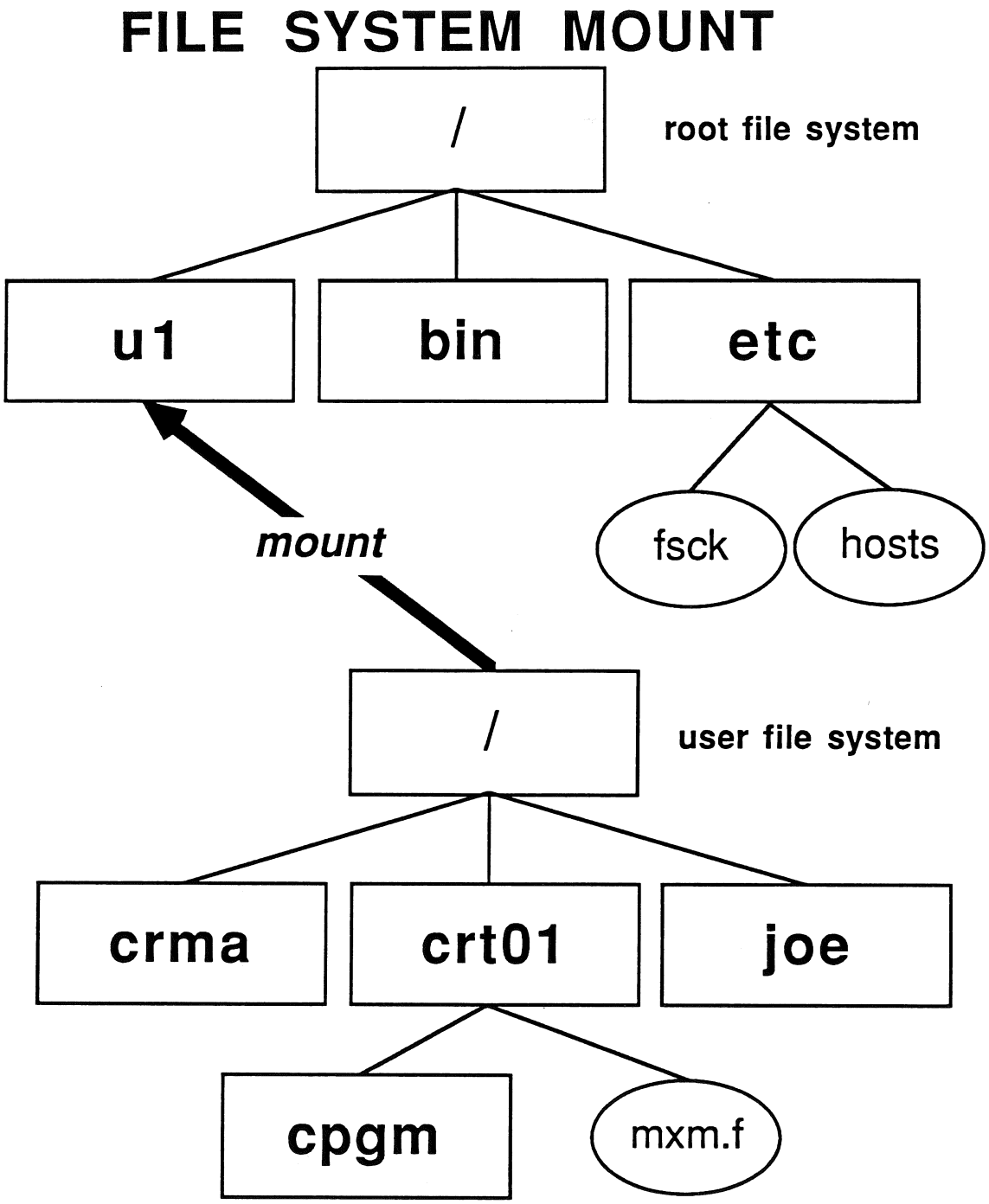
The top of the tree is called the **root filesystem.**

The command used to check the structure of each filesystem is called **`fsck`.**

The command used to attach each filesystem onto the big tree is called **`mount`.**

When you start UNICOS to single-user mode the **root filesystem** is mounted but no other filesystems are mounted. When you change to multiuser mode the the rest of the filesystems are fsck'd and mounted. During the shutdown process all filesystems except the **root filesystem** are unmounted again using the **`umount`** command (note that the command is **`umount`**, NOT **`unmount`**!).

You can see what filesystems are currently mounted by simply typing in the `mount` command with no options.


FILE SYSTEM MOUNT
/
root file system
u1
bin
etc
fsck
hosts
mount
/
user file system
crma
crt01
joe
cpgm
mxm.f

The `fsck` command

During system startup all filesystems will be checked using the **`fsck`** command. Various checks are made on the structure of the filesystem tree such as making sure that every file and subdirectory has a parent directory. fsck also checks a special part of each file and subdirectory called an **inode**. An inode contains information such as owner, group, permissions, size, access and modification times, and location of the data on the disk drive.

If UNICOS has been shut down in an orderly manner `fsck` should just check your filesystems silently and the system startup procedure will continue. However, if the system was not shut down properly or if you did not sync the data from buffers to disk or if the system crashed then you might get messages from `fsck` telling you that there is something wrong with one of the filesystems. UNICOS won't let you mount corrupted filesystems.

The normal format of an `fsck` message is to tell you about the problem it has discovered, followed by a question asking whether or not the problem should be fixed. You need to reply with either **`y`** or **`n`** in order to continue. It is extremely unwise to reply with **`n`** unless you really understand the internal workings of a filesystem or have been asked to do so by your system administrator. You need to discuss filesystem checking with your administrator and decide upon a set of operational guidelines appropriate for your system. You should make sure which questions you can just reply **`y`** to and which questions your administrator wants to investigate before replying.

The most usual question that you will be asked is **`Check for orphans?`** because there is a **`link count discrepancy`**. All this means is that there may be files in the tree structure that do not appear to belong in any parent directory. If you reply **`y`** to the question, `fsck` will scan the whole filesystem, looking for such orphan files. If any are found you will be asked **`Salvage?`**. If you reply **`y`** then the file will be put into a special directory called **`lost+found`** in the top of this filesystem subtree so that the owner of the file can go and find it. If you reply **`n`**, then the orphan file will be removed. Find out from your administrator what you should reply to this question.

Occasionally, `fsck` may find a problem with an individual file or directory. The things that could be wrong are **`bad mode`** (i.e., invalid file permissions), **`bad data blocks`** and so on. You will be asked if you want to **`Remove?`** the file. You must agree with your administrator whether you should go ahead and reply **`y`** to remove the file or get some help. If you ever remove files, note down as much information as possible, so the owner can be informed of the loss.

Filesystems and the **df** command

As system operator you will need to know what filesystems are on your UNICOS system. You should be able to determine how much free space is left in a filesystem so that you can issue warning messages to users who are using one that is almost full.

The command to use to find out the general status of all filesystems currently mounted on your system is **df** (**d**isk **f**ree). `df` with no options will tell you the name of each mounted filesystem (how the user refers to it), where in the root filesystem it has been mounted (how the system refers to it), an indication of the amount of free space left (%), and how many i-nodes are available (the number of files the filesystem can keep track of). An asterisk after the percentage free space left means that there are no empty tracks remaining on the file system (the available space is fragmented).

```
unicos$ df
/            (/dev/dsk/root    ):  44232  4K blocks ( 37.0%)*   26239 I-nodes
/admin       (/dev/dsk/admin   ):   5120  4K blocks ( 10.3%)*   34700 I-nodes
/core        (/dev/dsk/core    ):  51232  4K blocks ( 14.9%)*   32595 I-nodes
/tmp         (/dev/dsk/tmp60   ): 887312  4K blocks ( 92.7%)*  129970 I-nodes
/usr         (/dev/dsk/usr.a   ):  69056  4K blocks (   .8%)*   23649 I-nodes
```

Network file systems (**NFS**) can be mounted onto the UNICOS tree as well. NFS will be discussed later.

There are a number of options on the `df` command to display additional filesystem information. See the `man` page for details.

File System Monitor

UNICOS provides a filesystem monitor which can provide warning messages when filesystems are nearing certain threshholds. At that point you may be asked to send a message to users regarding the filesystem or users may be notified directly. In the example below, the **fsmon** command shows that the monitor is enabled for all the listed filesystems. `/usr` reached its critical (C) threshhold and a warning script has executed (X). Your administrator can tell you more about the file system monitor.

```
unicos$ fsmon
 n  File System     Status      Use    Warn   Crit  Time encountered
 1 /               E-------   63.0%  97.0%  99.0%
 2 /admin          E-------   89.7%  93.0%  97.0%
 3 /core           E-------   85.1%  88.0%  90.0%
 4 /tmp            E-------    7.3%  93.0%  97.0%
 5 /usr            EC-X----   99.2%  97.0%  99.0% 14:14:43 11/04
```

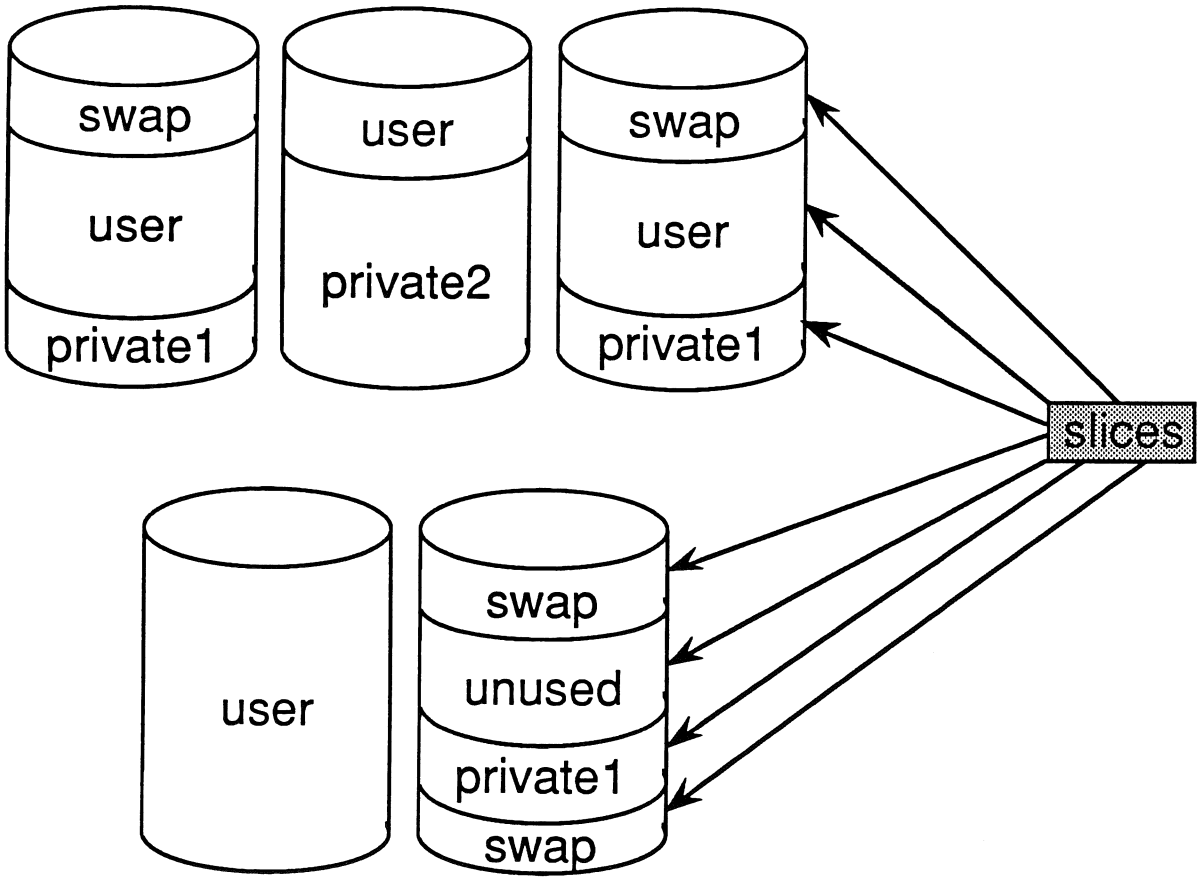
Filesystems and disks

Unlike most UNIX systems which require each filesystem to be defined within the boundaries of one disk drive, UNICOS filesystems are composed of one or more sections of one or more physical disk drives. These sections are called **slices**. This means that if there is a hardware problem with one of the disks, you will need to know which slices of which filesystems are on that disk in order to work out which filesystems are effected.

UNICOS File System

slice: all or part of a physical device

file system: one or more slices



file systems shown: ***user***
swap
private1
private2

The pddstat and dmap commands

The command to find out what physical disk drives are connected to your UNICOS system is **pddstat**. The command to examine the layout of filesystem slices on a particular disk is **dmap**. Both commands are in the `/etc` directory and you need to be superuser to run them.

The `pddstat` example on the next page shows all the physical drives attached to a UNICOS system. Each disk is uniquely identified by its iopath and unit number. An iopath number for a DD60 drive of 0230 means that the drive is attached to IO cluster 0, IOP 2, and channel 30. Its unit number (in column 3) relates to the drive's physical location in the box.

The iopath and unit numbers can be used in the `dmap` command to see what filesystem slices are on that drive. The `dmap -p` (physical) example of the disk called 1236.3 (iopath 1236, unit number 3) shows that there are slices belonging to the filesystems called `qte1`, `qtest1`, `tmpsrc`, `qtest2`, `vers`, `comp`, `free09`, and `tmp60`.

This means that if there were a hardware error causing this disk to be unusable, all of these filesystems would be inaccessible to UNICOS. Recovery from this type of situation is normally performed by a system administrator. It involves changing the UNICOS startup parameter file which contains the definition of all the physical disk devices on your system and the location of the filesystem slices on those disks.

The `dmap -l` (logical) output shows what drives filesystem `tmp60` is spread across.

As operator you should have available to you a copy of the output of the `dmap` command showing the physical disk devices on your system so you are able to properly assess the extent of filesystem damage in the event of disk problems. If user filesystems are the only ones damaged, you may be able to fix the problem yourself by reloading the filesystem from a backup tape, but some filesystems are vital to the UNICOS system. It is important that you discuss such recovery procedures with your administrator since it is partly an administrative problem and partly an operational problem. If you, as system operator, need to perform this type of task without assistance from your administrator you should attend a UNICOS System Administration class first.

Another command like `dmap` is **/etc/stor**. To see what filesystems are on all drives you could use **stor /dev/pdd/***. See the man page for more.

Output from the pddstat and dmap commands

```
unicos# pddstat
          device                                  # of blocks          errors
 iopath type unit state req  func     reads      writes    rd-ur wt-ur total
 ------ ---- ---- ----- --- ------ ---------- ---------- ----- ----- -----

  0230  DD60    4     up  0  wrtbhd      26854      52125     0     0     0
  0232  DD60    5     up  0  wrtbhd      12563       5347     0     0     0
  0234  DD60    6     up  0  wrtbhd       4164       3507     0     0     0
  0236  DD60    7     up  0  read       111369      19745     0     0     0
  0330  DD49    0     up  0  read          453        385     0     0     0
  0332  DD50    0     up  0  read          288         23     0     0     0
  0334  DD50    0     up  0  wrtbhd       8345       1768     0     0     0
  1032  DD60    1     up  0  read        1544       1248     0     0     0
  1034  DD60    2     up  0  wrtbhd      55586     108232     0     0     0
  1034  DD60    3     up  0  wrtbhd       2422        743     0     0     0
  1036  DD60    4     up  0  wrtbhd       5638       1723     0     0     0
  1036  DD60    5     up  0  wrtbhd      21427      23168     0     0     0
  1036  DD60    6     up  0  read        1631          0     0     0     0
  1036  DD60    7     up  0  read         717          4     0     0     0
  1130  DD61    0     up  0  wrtbhd     326927      49539     0     0     0
  1132  DD61    2     up  0  wrtbhd      28594       9096     0     0     0
  1132  DD61    3     up  0  read       73282       1181     0     0     0
  1134  DD61    4     up  0  wrtbhd     697093       1862     0     0     0
  1134  DD61    5     up  0  read        15554         19     0     0     0
  1136  DD61    6     up  0  read        17619        323     0     0     0
  1230  DD60    0     up  0  wrtbhd      51122       7734     0     0     0
  1232  DD60    1     up  0  read         1376          4     0     0     0
  1234  DD60    2     up  0  read       145896        727     0     0     0
  1236  DD60    3     up  0  wrtbhd       6990       4964     0     0     0
sn1601# dmap -p 1236.3
Physical device: 1236.3

Minor   Slice     Cyls/block    Blocks      Name
-----   -----    ------------  ------   -----------
  33        0        0 -  406    74888   /dev/dsk/qte1
  85        0      407 -  445     7176   /dev/dsk/qtest1
  74        0      446 -  581    25024   /dev/dsk/tmpsrc
  53        0      582 -  644    11592   /dev/dsk/qtest2
  13        0      645 -  807    29992   /dev/dsk/vers
  15        0      808 - 1079    50048   /dev/dsk/comp
  76        0     1080 - 1300    40664   /dev/dsk/free09
  57        3     1301 - 2601   239384   /dev/dsk/tmp60
sn1601# dmap -l /dev/dsk/tmp60
Logical device: minor 57   /dev/dsk/tmp60

Slice    Cyls/block    Blocks      Physical device
-----   ------------  ------     ---------------
    0   1301 - 2601   239384     dd60 - 0230.4
    1   1301 - 2601   239384     dd60 - 0232.5
    2   1301 - 2601   239384     dd60 - 0234.6
    3   1301 - 2601   239384     dd60 - 1236.3
```

Filesystem backups

As system operator you will be responsible for taking routine backups of all the filesystems on your UNICOS system. You may also be responsible for reloading an individual file from backup tapes because a user has accidentally deleted it.

You may regularly reload anything up to a whole filesystem from backup tapes if there has been some sort of disk problem causing loss of data. In order that you are able to do this you need to agree with your administrator on a list of all filesystems which need to be backed up.

You need to work out some method of recording which tapes are used for which filesystems, just as you would when taking backups of any other system.

The dump command

The command used to take a backup of a filesystem is **dump**. You need to be logged on to UNICOS as superuser (i.e., root) to use the dump command because you need to have read access to the disk area that you are backing up. When you dump a filesystem you need to select a **dump level**, from **0** to **9**.

A level of 0 will dump **all** files and directories in the filesystem. A dump at any other level will only dump files and directories that have been modified or created since the last dump at a lower level. For example, you might decide to take a level 0 dump once a month which would do a full backup of the filesystem. Then you might take a level 1 dump once a week which would back up all files and directories that had been created or changed since the level 0 backup (i.e., since the monthly full backup). Finally you might take a level 2 dump every evening which would dump all files and directories that had been created or modified since the level 1 dump (i.e., since the weekly incremental backup).

You need to choose levels and dump frequencies appropriate to your system. Your choice will depend on the level of guarantee expected by your users and the number of files and directories that are created and changed each day. For example, there is little point in taking weekly full backups and daily incrementals if the incrementals use as many tapes as the full backups (indicating that most files are new or have been changed). You may as well just take full backups all the time. The reverse is also true - if by the end of the week your incrementals are hardly using any tapes at all (indicating that very few files and directories have changed), maybe you should only take a full backup every month.

dump stores the information about which filesystems were dumped at which levels and on what date in a file called **/etc/dumpdates**. You can look at this file or you can use options on the dump command to find the information.

dump has several options, so read the man page before you start to use it. Most of the options are self-explanatory and many can be left to default. There are some examples at the end of the man page. One thing you will need to know is the name of the filesystem you are dumping. This information can be found by looking at the second column of the output from the df command. For the df example earlier in this section, the four filesystems are called **/dev/dsk/u**, **/dev/dsk/usr**, **/dev/dsk/root**, and **/dev/dsk/tmp**.

See the dump example in this section.

The `restore` command

`restore` reads backup tapes which were created with the `dump` command. There are options on restore to list the files on the backup (**`-t`**), retrieve specific files from the backup (**`-x`**), and reload the whole backup (**`-r`**). Another option which is very useful for restoring a few files or directories (rather than everything) is **`-i`**. This puts you into an "interactive restore". You can examine the contents of the backup tape using **`ls`**, **`pwd`**, and **`cd`**, and you can **`add`** files and directories to the list that you want to restore. When you have finished creating your list, **`extract`** will cause the restore command to go ahead and restore the files you selected.

There are many other options on the restore command, most of which are self-explanatory and many of them are defaults. Read the whole man page before you begin.

WARNING: Be extremely careful when reloading files because restore will create them in your current directory by default, NOT into the appropriate place in your hierarchy of filesystems. Therefore you could possibly overwrite like-name files. So make absolutely sure that you are in the correct directory before you start! You may want to restore them to `/tmp` (or better yet, to the user's $HOME/tmp), then have the users move them to their appropriate directories.

NOTE: You need to be superuser to run the `restore` command and if you are using the UNICOS multilevel security (MLS) feature, you will have different procedures for restoring files. Your administrator will provide you with those procedures.

Example of the dump command

```
unicos# dump -t0 -u -c     # this dumps all files to cartridge tape
                           # (-c) and writes a record of the dump to
                           # /etc/dumpdates (-u)
```

Example of the restore command

```
unicos# cd /tmp   # or user's $HOME/tmp directory
unicos# restore -i -l sl -V a1:a2 -c
                           # this is an interactive restore (-i) to
                           # /tmp from standard label cartridge tape
                           # (-l -sl and -c) using volumes a1 and a2
                           # (-V a1:a2)
```

ldcache

As was discussed in the Startup and Shutdown section, ldcache is an important performace feature in UNICOS and operators need to understand its function and how to deal with it.

Basically, ldcache is a means of copying parts or all of frequently accessed filesystems to SSD memory or a predefined space in main (central) memory. Ldcache works together with the system buffers (in central memory).

For example, if a job needed the `ls` command and `/bin` (where `ls` is located) had been copied to the SSD, the code for `ls` would be located very quickly, assuming the code for `ls` was not located in the system buffers.

Also, if a filesystem being written to is ldcached, the data is written there before it gets flushed to disk (at a later, more convenient, time for the system). Therefore, it stands to reason that at a system shutdown the data which has not been written back to disk would be lost if that data were not flushed out to a non-volatile storage area (disk).

It is possible to change this preallocated area of SSD or main memory while the system is running, using the **ldcache** command. You must be logged on as superuser (root) to modify ldcache, but not to see how it is currently set up.

Logical device caching is typically set up when the system is brought up into multi-user mode, through the **/etc/rc** startup script. However, an administrator may dynamically change the allocation of SSD or main memory (MEM) through an `ldcache` command entered on the command line. If logical device caching is being taken away from one filesystem and then given to another filesystem, the caching needs to be turned off of the old file system. This is accomplished by setting the units to 0 for the file system.

```
unicos# ldcache -l device  -n 0
```

ldcache -l *device* **-n** *units* [**-s** *size*] [**-t** *type*]

-l *device*	Full pathname or minor device number of logical device.
-n *units*	Number of cache units to assign. If 0, the logical device caching is released.
-s *size*	Size of each cache unit (in 4K byte blocks).
-t *type*	Type of memory for cache (**MEM** or **SSD**).

Setting up ldcache

```
unicos# cat /etc/rc
     .
     .
     echo "Do you want logical device cache? (y or n)\c"
     read reply
     if [ "$reply" = y ]
     then
          /etc/ldcache -l /dev/dsk/root -n 300 -s 42 -t SSD
          /etc/ldcache -l /dev/dsk/swap -n 200 -s 252 -t SSD
          /etc/ldcache -l /dev/dsk/usr -n 300 -s 42 -t SSD
          /etc/ldcache -l /dev/dsk/tmp -n 800 -s 84 -t SSD
          /etc/ldcache -l /dev/dsk/usr_src -n 100 -s 48 -t SSD
          /etc/ldcache -l /dev/dsk/usr_tmp -n 100 -s 96 -t SSD
          /etc/ldcache -l /dev/dsk/user_a -n 100 -s 42 -t SSD
          /etc/ldcache -l /dev/dsk/user_b -n 100 -s 42 -t SSD
          /etc/ldcache -l /dev/dsk/user_c -n 100 -s 42 -t SSD
          /etc/ldcache -l /dev/dsk/user_d -n 100 -s 42 -t SSD
          /etc/ldcache -l /dev/dsk/user_e -n 100 -s 42 -t SSD
     fi
```

Redistribution of ldcache

```
unicos# /etc/ldcache -l /dev/dsk/user_e -n 0
unicos# /etc/ldcache -l /dev/dsk/user_a -n 200 -s 42 -t SSD
```

ldsync

ldsync flushes data from all logical device caches to disk. Only data that has been written to logical device cache, but not to disk, is affected. It does not alter data in the system buffer cache. During normal operation, UNICOS periodically flushes the data to disk. However, the operator should execute the `ldsync` command at system shutdown, following a `sync` command, because data that is flushed from the system buffers goes to ldcache on its way back to disk if the filesystem the data belongs to were defined in ldcache. Also, the `ldsync` command should be placed in the `/etc/shutdown` script.

Example:

```
unicos# sync
unicos# sync
unicos# sync
unicos# ldsync
```

ldcache performance statistics

The output of the `ldcache` command with no options shows valuable information about how useful ldcache is in its current configuration. The columns of information are listed below (refer to the example on the next page).

T Type of memory being used for ldcache
B = BMR (Buffer Memory in the IOS)
M = Main (central) memory
S = SSD memory

unit Number of cache units assigned

size Size of each cache unit in 4096-byte blocks

reads Number of reads from the defined filesystem

writes Number of writes to the defined filesystem

hits Number of times the requested data block was located in ldcache

misses Number of times the requested data block was not located in ldcache

rate Percentage of time the requested block from the filesystem was found in ldcache

name Name of ldcached filesystem

Displaying the ldcache hit rate

```
unicos$ ldcache
T unit size    reads     writes       hits     misses    rate   name
- ---- ---- --------- ---------- ---------- ------- -----  --------

S  500   48    9325389    2402116    3365509  585379  85.18 /dev/dsk/ptmp
S   50   24    1588638    1343809    2103761  495535  80.94 /dev/dsk/spool
S  500   42   34284950   23013994   11804941  370960  96.95 /dev/dsk/tmp_50
S   50   42      25789      11424       9536     590  94.17 /dev/dsk/root_k
S  200   42   46172424    3429081    6148539  118956  98.10 /dev/dsk/root_o
S   50   42      37380       2094       3759     691  84.47 /dev/dsk/usr_k
S  200   42    7604357     190306     716006   26119  96.48 /dev/dsk/usr_o
S   50   48      49735      23339      21679     686  96.93 /dev/dsk/src_k
S  200   42     625200     174566     541194   63447  89.51 /dev/dsk/src_o
S  250   24      96981       6236      91246   10144  90.00 /dev/dsk/mktg
S  300   24    6319864     367026     795050  135653  85.42 /dev/dsk/soft
S  300   24     358064      12792      99751   14304  87.46 /dev/dsk/os
S  300   24     132089      33458      98593   17810  84.70 /dev/dsk/qa
```

```
unicos$ ldcache -l /dev/dsk/root_o

 /dev/dsk/root_o                    Thu Feb 28 08:42:54 1991


                                Read data          Write data
                                ---------          ----------

 Blocks transferred:            17862              848
 Cache to user:                 17862 blks         847 blks
 Cache to disk:                 336 blks           715 blks
 Cache/disk ratio:              53.160714          1.184615
 Avg request length:            26 blks            1 blks
 Lst transfer rate:             8.141254 Mbs       0.289739 Mbs
 Max transfer rate:             8.713365 Mbs       0.399379 Mbs
 Cache hits:                    791                768
 Cache misses:                  8                  3
 Cache hit rate:                98.998748          99.610895
```

NFS

A Network File System (**NFS**) is a filesystem that physically resides on a disk that is attached to another computer system but is mounted onto UNICOS, just as if it was a UNICOS filesystem. If your administrator has not configured your system to support NFS filesystems you can ignore this section.

Once an NFS filesystem is mounted it looks just like any other filesystem on UNICOS, even though the files and directories in that filesystem actually reside on another computer's disks. However, there are some operational differences between local filesystems and NFS filesystems.

When you start up UNICOS it should be able to **mount** all the local filesystems after **fsck**'ing them and the disks containing them will be available all the time. However, there is no guarantee that the NFS filesystems can be mounted. There could be a disk problem with one of the NFS filesystems or the computer that the NFS filesystem belongs to may be down. If UNICOS cannot mount one of the NFS filesystems at startup time, the request will be put into the background until it can complete.

When startup is complete, you should check to see if any NFS filesystems were not mounted. Use the **df** command to do this. If one has not been mounted, then you need to check that the network is working properly and check to see that the other computer is running. You may also need to do this at other times if you have network problems or a problem with a computer that owns an NFS filesystem. You can manually type in the command to mount an NFS filesystem, but you need to be logged onto UNICOS as superuser (i.e., root) to do this.

```
unicos# mount -t NFS -o bg mysun:/usr/test /test
```

This example mounts the NFS filesystem called **/usr/test** located on a computer called **mysun** onto the UNICOS system under the directory called **/test**. The **-o bg** option just puts the request into the background to keep trying if it did not work immediately. There are many other options on the `mount` command so you should discuss with your system administrator to determine the exact command to use on your system and look at the man page for `mount`.

NOTE: NFS uses TCP/IP memory buffers and therefore could use up the allocated space. Use the **netstat -m** command to check on mbuf allocation.

The umount command

In addition to mounting an NFS filesystem you may sometimes want to unmount one using the **umount** command. You need to logon to UNICOS as superuser (i.e., root) to do this. Unmount an NFS filesystem if you know that the computer that owns the filesystem is going to be shut down or the network is going down or any other condition that would mean that the NFS filesystem is not going to be available on UNICOS. For the example above you would unmount the NFS filesystem like this.

```
unicos# umount /test
```

It is also possible to mount the UNICOS filesystems as NFS filesystems onto another computer system in your network. Talk to the administrator of the other computer to do this.

NOTE: You cannot unmount a busy filesystem (i.e., users are accessing it). To find out if anyone is accessing a particular filesystem use the **fuser** command, as in the following example:

```
unicos$ fuser -u /dev/dsk/tmp
```

There are many options on the `fuser` command. See the man page on it for more.

The tape subsystem

Tape startup and **tpstat**

When the Cray system is booted, UNICOS defines the tape configuration for the tape controllers that are connected to the IOS. Messages relating to tape errors may appear on the UNICOS console as well as the `/var/logs/ioplog` file.

When you bring UNICOS up in multiuser mode the **tape daemon (tpdaemon)** will be started automatically. The commands that you will use to monitor and control the tape drives or tape controllers will communicate with this daemon.

To look at the state of the tape devices in the system use the **tpstat** command.

```
unicos$ tpstat
    userid jobid dgn  a stat    dvn     bx i  rl    ivsn   evsn   blks NQSid
                 TAPE + down    tape100 02 0
                 TAPE + down    tape101 03 0
                 TAPE + down    tape102 04 0
                 CART + down    cart509 07 0
                 CART + idle    cart300 10 0  is    WE0022 WE0022
                 CART + idle    cart301 09 0  is    WE0023 WE0023
    hew     113  CART + assn    cart302 11 0  ia    ISCAL  XXX     9
    jas     44   CART + assn+   cart303 0  in ISCNL ISCNL  30
```

In the example above, devices `tape100-cart509` are configured as down. Devices `cart300` and `cart301` are not assigned to a job, so they are idle. They have IBM-standard tapes `WE0022` and `WE0023` mounted on them. User `hew` with job id `113` has `cart302` assigned as ring-in, ANSI-labeled, with an external volume identifier equal to `XXX` and an internal volume identifier equal to `ISCAL`. Nine blocks of data have been processed.

User `jas` with job id `44` has been assigned `cart303`. It has been specified to be nonload (+), so that the tape will remain mounted after user `jas` issues an `rls` command. This tape has been assigned as ring-in, non-labeled, volume identifier equal to `ISCNL`, and with 30 blocks of data processed. AVR is set to on for all tapes.

There are various other states which are listed in the man page for **tpstat**. The other most useful columns are the **dvn** (which is the **d**evice **n**ame) and the **blks** (which is the number of blocks of data that have been read or written to the tape if it is assigned to a user). This number increments as data is being read or written.

Configuring tape drives up and down

When you start UNICOS, the tape drives may not automatically be configured up. Therefore, you may be required to manually configured certain tape drives up with the **tpconfig** command before users can access them. You need to specify the name of the tape device to be configured up. You can get a list of device names from the **dvn** column of the output from `tpstat`.

```
unicos# tpconfig -n devicename up
unicos# tpconfig devicename down
unicos# tpconfig devicename1:devicename2 up
unicos# tpconfig devicename1-devicename2 up
```

The `-n` option in the first example above configure a tape drive to **not** release itself when done. This allows the same user to request the drive again without requiring you to shut the drive door.

It is also possible to use the **tpconfig** command to configure tape controllers up and down. To do this you need to know the control unit number and IOS channel number.

```
unicos# tpconfig -u 20/1 down
```

The example above configures controller unit number 1 on channel 20 down.

If you are not sure which tape devices are on which controllers or which controllers are on which IOS channels, use the **tpdev** command to find out.

```
unicos$ tpdev
  dev name  dev grp   bnk  did  ios  dvst  cu   cust  ch   chst  loader
  ========  ========  ===  ===  ===  ====  ===  ====  ===  ====  ========
       130  CART        0    0    0  down    0  up      21  up    Operator
       131  CART        0    1    0  down    0  up      21  up    Operator
       132  CART        0    2    0  up      0  up      21  up    Operator
       133  CART        0    3    0  up      0  up      21  up    Operator
       140  CART        4    0    0  down    4  down    24  up    Operator
       141  CART        4    1    0  down    4  down    24  up    Operator
       142  CART        4    2    0  up      4  down    24  up    Operator
       143  CART        4    3    0  up      4  down    24  up    Operator
       210  TAPE        1    0    0  up      2  up      21  up    Operator
       211  TAPE        1    1    0  down    2  up      21  up    Operator
       212  TAPE        1    2    0  down    2  up      21  up    Operator
       213  TAPE        1    3    0  down    2  up      21  up    Operator
```

In the example above there are a number of cartridge and round tape drives, each connected to a controller. The controllers have unit numbers of 0, 4, and 2 and are on channel numbers 21 and 24. The controller unit on channel 24 is down. To configure it up again you would use the command **tpconfig -u**

Responding to tape messages through the **oper** facility

When a user wants to access a tape device, messages will be written on the UNICOS system console and possibly on the screens of anyone logged on as operator to indicate that there are operator messages waiting for attention. These messages will keep appearing until you respond to the messages. In order to read the messages, use the **oper** command. **oper**, by default, displays operator messages on a screen that is constantly refreshed. Each message has a unique message number. A tape mount message will ask you to mount a particular tape onto one of the tape devices. You can respond to the message by either mounting the tape as requested or by replying to the message. Each message will explain exactly what you are allowed to reply.

```
unicos$ oper              # the full path to this command is /usr/lib/msg/oper
command:msgd Page 1 [delay 10] Wed Jun 7 14:41:38 1989

Msg #   Time     System Message
=====   =====    ==============
 42     13:09    TM122 - mount tape X32.51(sl) ring-in on a TAPE device
                 for jlf 1293, () or reply cancel / device name

 43     13:25    TM122 - mount tape ADJ73(sl) ring-in on a CART device
                 for pmn 1301, () or reply cancel / device name

 44     15:10    TM011 - enter vsn for tape on device cart142

oper>
```

If you decide to reply to a message rather than carry out an action such as mounting a tape, then use the **rep** command. The format of the command is:

```
oper> rep message-number reply
```

For example, to cancel the mount request in message number 42, type the following:

```
oper> rep 42 cancel
```

To enter a volume serial number (vsn) for the tape job in message 44 (after the tape has been mounted):

```
oper> rep 44 ADJ73
```

The same messages you see and reply to in `oper` are also logged to a file called **/usr/spool/msg/msglog.log** on UNICOS. It is a circular file, which means that the oldest data is overwritten by new data. It can provide you with a record of messages you received.

More about oper

To use oper correctly, you need to set up your terminal type definition. This will normally be set for you each time you log into UNICOS or you will be prompted for it. The following examples show how to set your terminal type manually to use oper from OpenWindows.

For the Bourne shell:

```
unicos$ TERM=sun-cmd
unicos$ export TERM
```

For the C shell:

```
unicos% setenv TERM sun-cmd
```

If you usually use oper from the same terminal type you might want to set up the terminal definition in your **.profile** (Bourne shell) or **.login** (C shell) file.

As shown on the previous page, the oper command runs the msgd command by default which displays operator messages such as tape mount requests. However, since oper is a constantly refreshing display, it is especially good for use with commands that continuously give different output. Whenever you type in a command at the oper prompt, oper looks in a file called /usr/lib/oper.rc which contains a list of the eligible commands you can run from within oper. An example of this file is shown on the next page.

Normally, operators do not have permission to add their own commands to the usr/lib/oper.rc file. If you want oper to refer to your own set of commands instead, you can list them in a file called .operrc that resides in your home directory. oper looks for this file first.

If you run a command in oper and then want to see the operator messages again, type in the **msgd** command to display them.

There is also a **help** utility in oper and a **redraw** utility to redraw the screen if someone (or the system) writes a message over the top of your refreshing display. You can make a snapshot copy of what is on the refreshing screen by using the **snap** command followed by the name of a file to put the copy into. To get out of the oper command, just type **exit**.

For more details see the man pages for oper, msgd, and rep.

Example of the /usr/lib/oper.rc file

```
# USMID @(#)cmd/msg/oper.rc     61.0    09/03/90 12:16:44
#
#       COPYRIGHT CRAY RESEARCH, INC.
#       UNPUBLISHED -- ALL RIGHTS RESERVED UNDER
#       THE COPYRIGHT LAWS OF THE UNITED STATES.
#
#  System default "oper" configuration file
#  ****************************************
#
#  Commands to be executed as refreshing displays
#
cal
cat
df
egrep
env
fgrep
find
finger
grep
infd
ls
man
msgd
news
ps
qstat
tpbmx
tpdev
tprst
tpstat
tpgstat
uscplink
uscpques
uscpstat
uscpstrs
who
whom
#
#  Commands which require full control of the screen
#     (must be preceded with ":")
#
:csh
:dis
:ed
:edit
:ex
:ksh
:mem
:more
:oper
:pg
:qmgr
:screen
:sh
:vedit
:vi
:view
```

Example `oper` script

Perhaps a better way of watching for tape mount requests and monitoring the tape status at the same time is to have a script which does the following:

```
tpstat | grep -v down
echo
msgd
```

Since `oper` continuously refreshes the screen, the effect of the script running over and over above will be to show you the current tape status of only the tape drives that are up as well as any messages you may need to reply to.

Sorting out tape problems

Sometimes you will get tape problems which require operator intervention. Errors reported by the tape drives or tape controllers will always be displayed on the UNICOS console window. You will normally respond to these types of problems by asking the user to try again or by cleaning a tape drive or by configuring the drive down and calling an engineer, etc.

On other occasions you may want to force a user to release a tape drive for some reason. The commands to use in this case are **`tpgstat`** and **`tpfrls`**. Use the **`tpgstat`** command to display the user id and the job id that has the tape device assigned. Then use **`tpfrls`** to force release the device.

```
unicos# tpgstat
user     job id  dgn    w  rsvd  used  mins  NQSid
tpdaemon    3    TAPE       1     1   120
mary      123    TAPE       1     1    19
john      138    TAPE       2     1     5

unicos# tpfrls mary 123

unicos# tpgstat
user     job id  dgn    w  rsvd  used  mins  NQSid
tpdaemon    3    TAPE       2     2   120
john      138    TAPE       2     1     5
```

Any tapes that are not assigned to a user show a user of the tape daemon itself in the **`tpgstat`** display.

Starting and stopping the tape daemon

You will not normally want to start or stop the tape daemon because this will happen automatically when you start up or shut down the UNICOS system. If for some reason you do need to do so, use the **tpdstop** command to stop it and the **tpdaemon** command to start it again.

```
unicos# /usr/lib/tp/tpdstop
unicos# /usr/lib/tp/tpdaemon
```

If you are running UNICOS 6.0 you might be able to restart the tape daemon using the `sdaemon` command. Ask your administrator for the proper procedure.

NOTE: If you are using the UNICOS multilevel security (MLS) feature, you will have different procedures for restarting the tape daemon. Your administrator will provide you with those procedures.

Other useful tape commands

There are a number of other commands available in UNICOS to help you better monitor your tape subsystem. Listed below are some of them. Refer to the man pages for them and/or discuss their usefulness with your administrator.

tpbmx displays detailed tape device information

tpu used to unload tapes; it has no effect on a tape currently in use; it is most useful for unloading tapes and freeing tape drives on systems with automatic volume recognition (AVR); the **-g** option overrides the `tpconfig -n` (no release) setting

tprst displays reserved tape status for current job ID

tpmql displays the current mount request list for all users who have completed initial mount processing and have a mount request pending

tpapm requests the tape daemon to mount a volume on any available drive that may be serviced by an autoloader; this is useful when it is necessary to premount volumes that are required by some program (e.g., by a backup) that is performing a multivolume operation, and the required volumes reside within the autoloader storage unit (silo).

Performance Monitoring

The need to monitor the system

Many operating tasks in a UNICOS environment are driven by specific events and requirements, such as a **shutdown** request or a tape mount request. However, there is a need for constant monitoring just to make sure that the machine is busy when there is work for it to do.

Many commands have already been discussed to help you see what is happening on the system and you should use these commands regularly. Examples are `ps`, `who`, `qstat`, `tpstat`, and `netstat`. These commands respectively display information about processes, interactive logins, batch requests, tapes, and the network. Don't forget that you can run these commands (and many others) from within the `oper` utility to get an updating display.

There are other commands, such as **sar** and **crayperf** that are useful for giving you an idea about how "busy" UNICOS is. Based on the output of these commands you might decide to increase or reduce the number of running batch requests, for example. This section describes the use of these and other commands.

The `sar` command

You will normally run the system activity reporter (`sar`) command in one of two modes. The first mode examines performance data that has been collected between midnight last night and now.

`sar` with no options reads a file created by a `cron` job running a special administrator command. The exact length and start and stop times of the intervals may be different on your system and are likely to change throughout the day to give more data when the machine is at its busiest.

```
unicos$ sar                    # the full path to sar is /usr/bin/sar

sn1101 sn1101 5.1 upr.82 CRAY Y-MP     01/30/91

00:00:00     %usr     %sys    %wsem      #locks    %idle
01:00:00        0        1        0           0       99
02:00:00        0        1        0           0       99
03:00:00        5        1        0           0       94
04:00:00       25        2        0           0       73
05:00:00       49        7        0           0       44
06:00:00       61       11        0           0       28
07:00:00       68       12        0           0       20
08:00:00       80       14        0           0        6
08:20:00       85       14        0           0        1
08:40:00       90       10        0           0        0
09:00:00       86       14        0           0        0
09:11:09          unix restarts
09:20:00       22        6        0           0       72
09:40:00       82       15        0           0        3
10:00:00       31        1        0           0       68

Average        49        8        0           0       43
```

In this example the system is idle from midnight to about 3 a.m. but the usage increases from sometime between 3 a.m. and 8 a.m. onward. The time at the left of each row is the time when the `cron` job ran to gather the data. Notice that at 9:11:09 UNICOS was restarted. It may have crashed or was taken down for some reason. After restarting it, it took a little while for usage to increase again.

Look at the last entry for the time period 9:40 to 10:00. The usage has dropped dramatically. This may be an indication that there is a problem on the system that you, as system operator, need to investigate. Is the network OK? Are there NQS requests queuing up because some of the queues have not been started? If you are running USCP, are the stations logged on?

It is possible that there is just less work for UNICOS to do at the moment but you should monitor the system using **`sar`** to see if there is a problem.

More on sar

Using sar in the other mode gives you current data. You need to decide how long each of your time intervals should be in seconds and how many of those intervals you want to look at. The command will print out the heading immediately and then one line of data every time interval for the number of intervals you specified. The following example looks at data starting now for five intervals of ten seconds each.

```
unicos$ sar 10 5

sn1101 sn1101 5.1 upr.82 CRAY Y-MP    01/30/91

12:18:15     %usr     %sys    %wsem     #locks    %idle
12:18:25       51       49        0          0        0
12:18:35       66       34        0          0        0
12:18:45       72       28        0          0        0
12:18:55       66       34        0          0        0
12:19:05       78       22        0          0        0

Average        67       33        0          0        0
```

You might want to run sar in this mode to check that the load is increasing again after you have fixed the problem that caused the system to go idle.

There are many options on sar to give you additional information about how the system is performing but running with no options will give you a good summary of how busy it is. See the man page for details.

WARNING: If sar shows that UNICOS is idle it does not always mean that there is not enough work to do! Occasionally there may be a number of large processes in memory and some other large processes waiting to get into memory. UNICOS could be wasting too much time trying to juggle the processes in and out of memory. You should be able to determine that this is the case by using the ps and qstat commands to see if there is actually work to be done and also by using the crayperf command (described later in this section) to see what is in memory. Also, if you allow more NQS requests to start, the situation will get worse, so you will know that the problem is memory contention, not lack of work! If this situation happens frequently, tell your administrator so that UNICOS can be tuned to suit the workload better.

Monitoring UNICOS through the X Window System

A variety of Cray system monitors are now available which allow operators and administrators to view system performance dynamically; i.e., in graphic form. This is done through the X Window System, the same graphical user interface used on the OWS for normal operator functions. In fact, these commands can be run on any workstation running the X Window System software, including a Macintosh.

The rest of this section describes some of those commands and one other useful window tool. Each of these commands are good candidates to add to your pull-down menu choices (in the `.openwin-menu` file) on the OWS-E so you can run them with little effort whenever you need them.

Preparing the OWS for X Window System displays

Since you will be viewing the graphical output on a remote terminal (usually the OWS-E), you need to tell the X Window System software running under UNICOS where to send the output.

If you do not know how to set up the X Windows System on your workstation, ask an administrator to help you or read one of the many introductions to the X Window System (see Appendix A for a suggestion). The best thing to do is to get an administrator to help you set things up and give you a demonstration.

If you do know something about the X Window System, the following example will help you set this up to view the graphics output on the OWS-E.

In a local window on the OWS-E, type **one** of the following two statements:

```
owse% xhost + sn1101
sn1101 being added to access control list
                # this allows an X Window System program running on Cray sn1101 to send its
                # output to this login session on the OWS-E; sn1101 is the network name of the
                # Cray; use the name of your Cray in place of sn1101

owse% xhost +
all hosts being allowed (access control disabled)
                # this allows an X Window System program running on ANY remotely
                # attached system to send its output to this login session on the OWS-E
```

On the Cray login (where the graphics command will be run), type the following two lines:

```
unicos$ DISPLAY=owse:0.0
                # this sets a variable telling the X Window System running on the Cray
                # where to send the output of any X Window System program; in this
                # case, OWS-E is the network name of the operator work station

unicos$ export DISPLAY          # this exports the variable so the program will always find it
```

To avoid typing the above statements every time you want to run a graphics program, include them in the `.profile` file of the respective machine and they will always be set properly.

The `crayperf` command

`crayperf` is a UNICOS graphical tool for monitoring system performance in real time. It requires the X Windows System graphics package to work. Read the `man` page for `crayperf` and for the most detail on `crayperf` read through the section describing it in the UNICOS System Administrator's Guide (SG-2018 or SG-2113 for UNICOS 6.0) before you start.

To make running `crayperf` easier you should include its parent directory in your path. In UNICOS 5.0 and 5.1 `crayperf` is found in `/usr/adm/bin` and in UNICOS 6.0 it is located in `/usr/bin`.

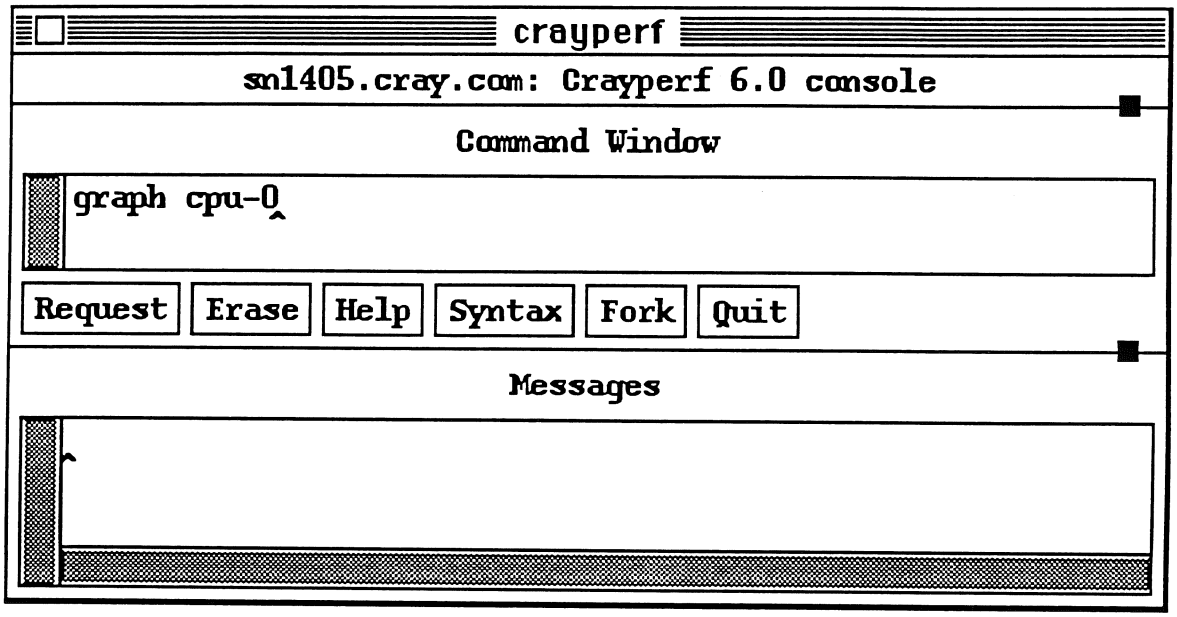
The next page shows the main `crayperf` window from which all performance windows are generated. The window request shown entered in the top part of the window is one which will generate a graph of cpu 0 usage (an example of that output follows).

The `crayperfd` daemon must be running in order to use `crayperf`. You may need to start it if it isn't.

NOTE: All the window examples shown in this workbook were generated on a Macintosh running the X Window System; therefore, the output you see on an OWS or Sun system will have a slightly different window box.

To complete the window request after typing it in, you must click in the `Request` box; pressing the Return key has no effect.

The `crayperf` main window

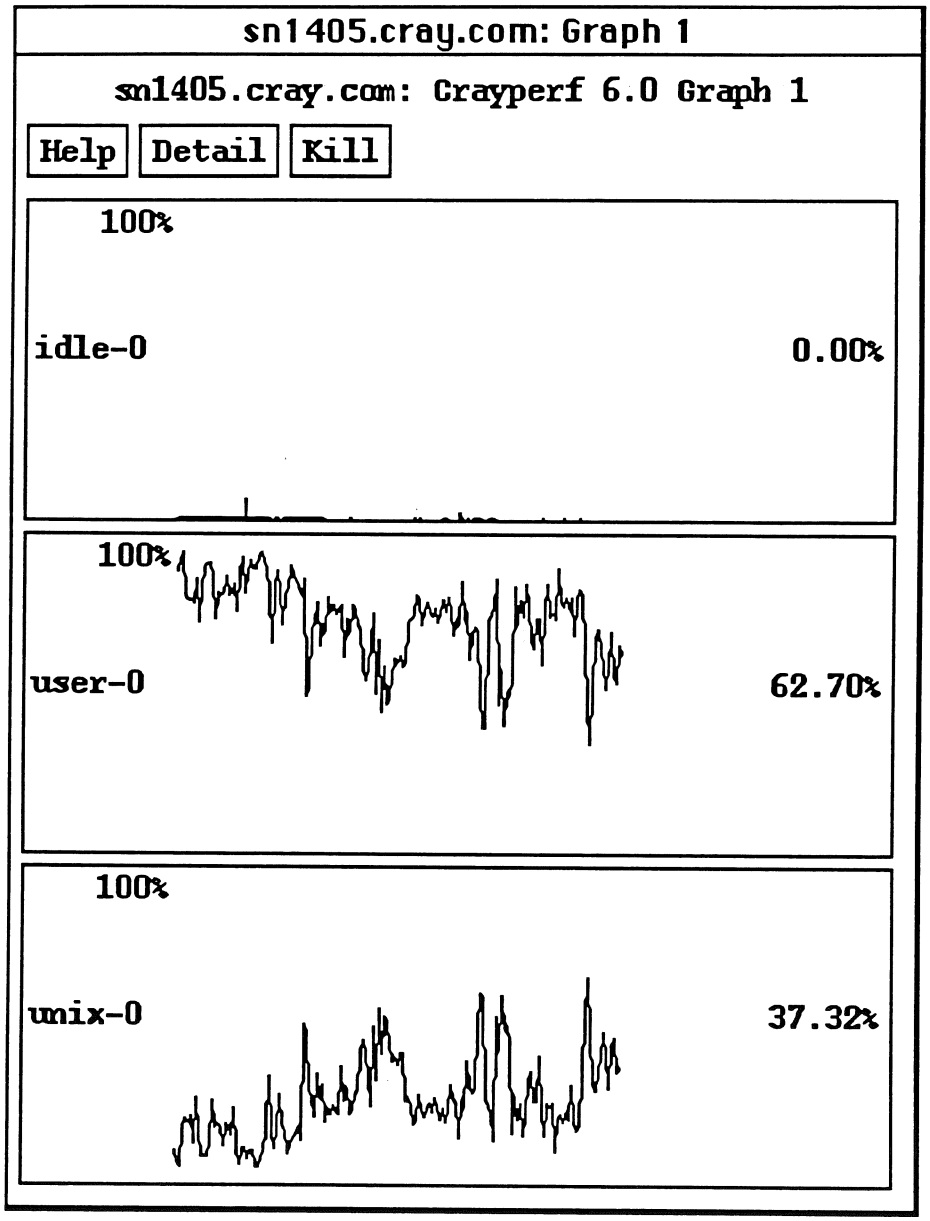


The `graph` window request in `crayperf`

The `graph` window request within `crayperf` allows monitoring many different parts of the UNICOS system including cpu usage, which is shown on the next page. This is basically the same output as the `sar` command, but in graphical form.

Use the `list graph` command at the main window to get a list of the graphs `crayperf` can generate.

Output from the `graph` window request



The `map` window request in `crayperf`

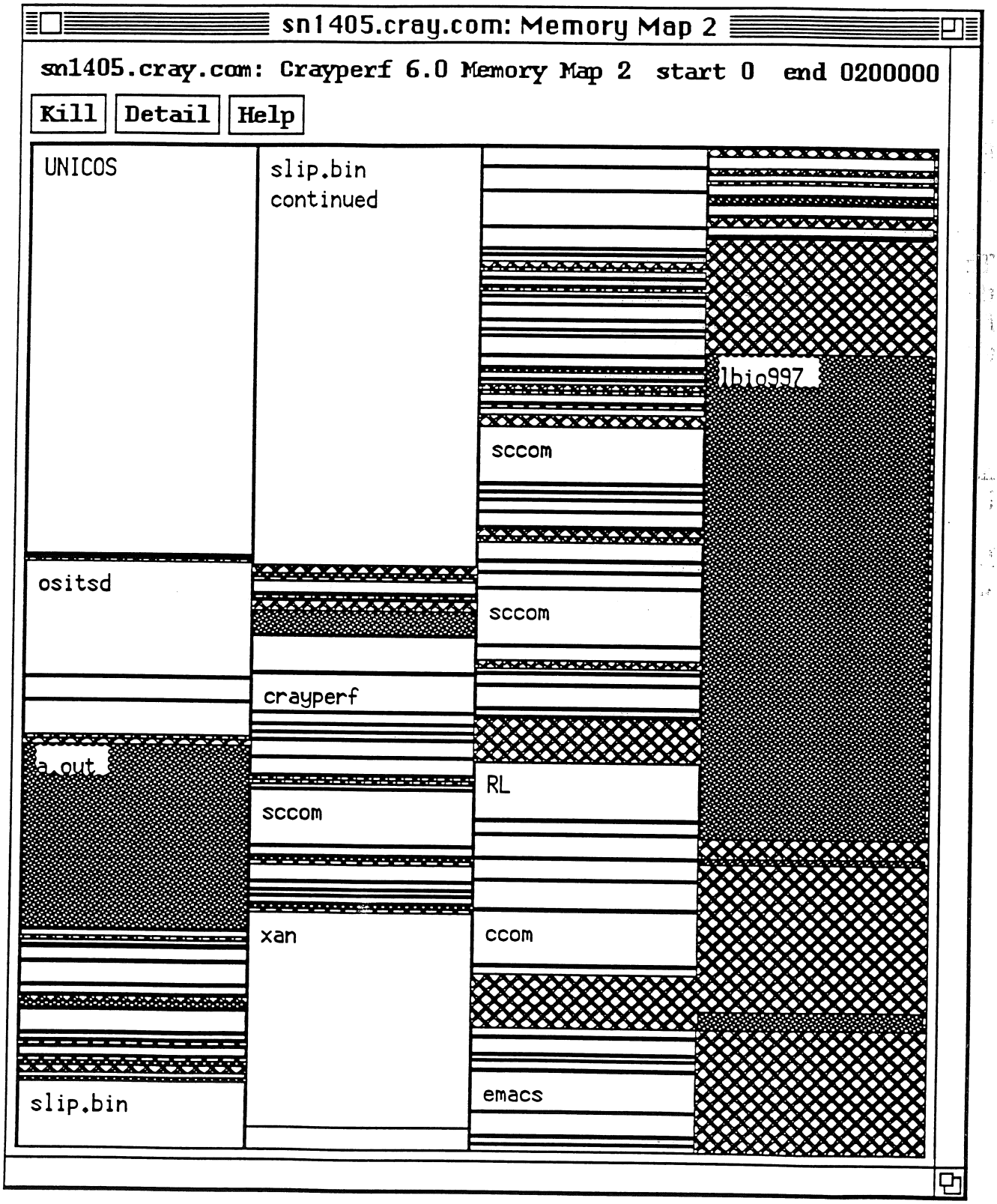
One of the most useful features of `crayperf` is the `map` window request which generates a display of memory showing processes moving in and out dynamically. It is easy to see how much free memory is available and how many processes are being swapped in and out of memory. This will help you verify a problem where the system is showing idle time due to excessive memory swapping, rather than just due to lack of work.

The next page shows example output of the `map` window request. The display in the window will automatically be readjusted if the window is resized. Also, from within this window, other more detailed windows can be generated allowing you to view small ranges of the system's memory. You can even view a single job in detail.

`crayperf` is a very useful tool for analyzing system performance but it does have some impact on performance itself! Notice how much memory `crayperf` is using in the example on the next page.

It's OK to use `crayperf` for investigating performance problems but you should probably not keep it running all day unless the system is idle anyway.

Output from the map window request



The xnetmon command

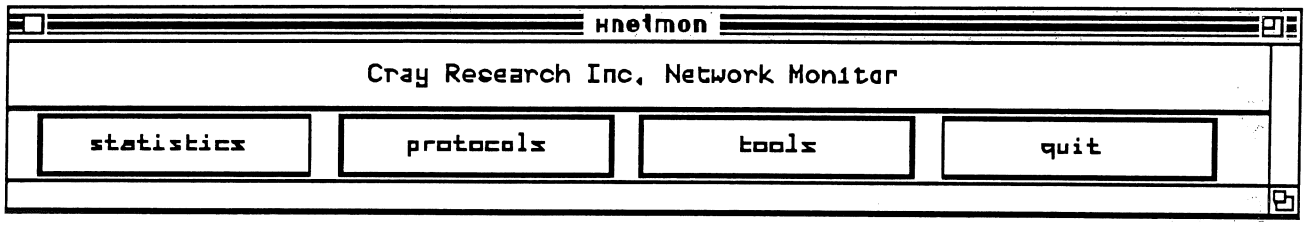
xnetmon is the new command with UNICOS 6.0, mentioned in the Connectivity section of this workbook, that is useful for monitoring most of the networking functions on a Cray system. Like `crayperf`, it has a main window from which other windows are generated. The main window is shown at the top of the next page.

The full pathname for `xnetmon` is `/usr/lib/monitor/xnetmon`.

From within the main window there are several pull-down menus including **statistics**, **protocols**, and **tools**. You will likely use the `protocols` menu the most, because it allows you to easily view current NFS, USCP, and TCP/IP usage in various forms. The example at the bottom of the following page shows the `uscpxmon` window, chosen from the `protocols` menu on the main window. That window is also showing the output of the **system status**, chosen from the **Status** menu.

There are too many options to discuss them all here. You need to work with your administrator in deciding which, if any, are of use to you. Refer to the man page on `xnetmon` for an overview of it and the UNICOS System Administrator's Guide (SG-2113) for a detailed description.

The xnetmon main window



Output from uscpxmon showing system status

uscpxmon

uscpxmon -- X11 based monitor of USCP

Status | Actions | Options | Help | Quit

jobname	jsq	nqsid	pid	q	dc	status	class	pri	fl	cpu	limit	mf	tid
ukrbkup	8863	11133	0	E	IN	QUEUED	b_30_1	20	0	0	30	M4	U0002
request	0	11315	0	E	IN	QUEUED	vers11B	20	0	0	500	??	vers
reqfile	0	11009	44242	E	IN	RUN	vers15a	20	377	0	500	??	vers
reqfile	0	11005	43499	E	IN	RUN	vers13E	20	376	0	500	??	vers
reqfile	0	11004	43486	E	IN	RUN	vers13E	20	376	0	500	??	vers
reqfile	0	11003	43467	E	IN	RUN	vers13E	20	376	0	500	??	vers
reqfile	0	10992	42287	E	IN	RUN	vers13C	20	376	0	500	??	vers
reqfile	0	10999	42936	E	IN	RUN	vers13D	20	376	0	500	??	vers
reqfile	0	10998	42913	E	IN	RUN	vers13D	20	376	0	500	??	vers
reqfile	0	10997	42899	E	IN	RUN	vers13D	20	376	0	500	??	vers
request	0	10988	41276	E	IN	RUN	vers13B	20	376	0	500	??	vers
NEWJOB	0	11109	57412	E	IN	RUN	b_600_8	20	7937	32	240	??	u3640
STDIN	0	11314	0	E	IN	QUEUED	b_30_1	20	0	0	30	??	vers
STDIN	0	11312	0	E	IN	QUEUED	b_30_1	20	0	0	30	??	vers
STDIN	0	11310	0	E	IN	QUEUED	b_30_1	20	0	0	30	??	vers
STDIN	0	11308	0	E	IN	QUEUED	b_30_1	20	0	0	30	??	vers
STDIN	0	11307	0	E	IN	QUEUED	b_30_1	20	0	0	30	??	vers
STDIN	0	11306	0	E	IN	QUEUED	b_30_1	20	0	0	30	??	vers
STDIN	0	11299	0	E	IN	QUEUED	b_30_1	20	0	0	30	??	vers
STDIN	0	11297	0	E	IN	QUEUED	b_30_1	20	0	0	30	??	vers
STDIN	0	11294	0	E	IN	QUEUED	b_30_1	20	0	0	30	??	vers
STDIN	0	11292	0	E	IN	QUEUED	b_30_1	20	0	0	30	??	vers
STDIN	0	11282	0	E	IN	QUEUED	b_30_1	20	0	0	30	??	vers

The `netperf` command

Another command useful for monitoring TCP/IP traffic is the **`netperf`** (network performance) command. It is used to monitor either NFS or TCP/IP traffic.

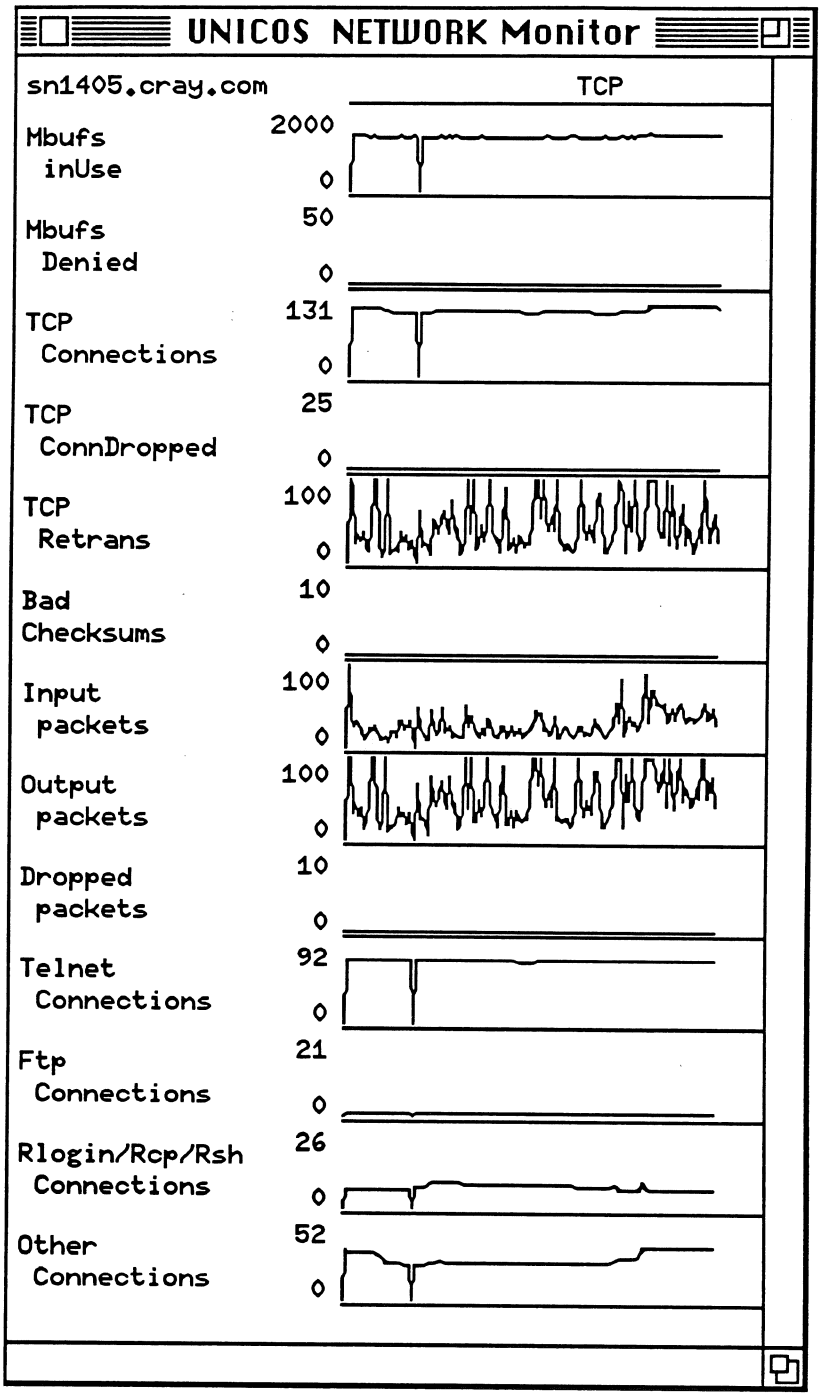
It can be run from the command line or from a menu within `xnetmon`. If it is run on the command line, its full pathname is `/usr/bin/netperf` in UNICOS 5.0 and 5.1. Its full pathname in UNICOS 6.0 is `/etc/netperf`. If it is run from within `xnetmon`, it is an NFS or TCP/IP menu choice under `protocols`.

`netperf` is dynamic in that it shows graphically how much work the network is receiving. The TCP/IP window can be useful to view, for example, how many mbufs the system is currently using and how many mbuf requests were denied. It also shows the number of TCP/IP connections, how many were dropped, retransmissions, and many others.

`netperf` sets up default ranges for the first pass across the window, then automatically adjusts the ranges to be more informative for each subsequent pass.

To quit `netperf` simply type **`q`** anywhere in the window and it will disappear.

Output from the `netperf` command



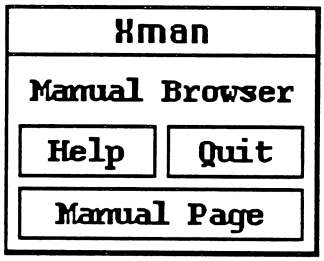
The xman command

Another useful tool worth learning about is **xman**, the X Window System version of the `man` command. It is a quick and easy way of viewing `man` pages without typing in very much. `xman` is located in `/usr/bin/X11/xman`.

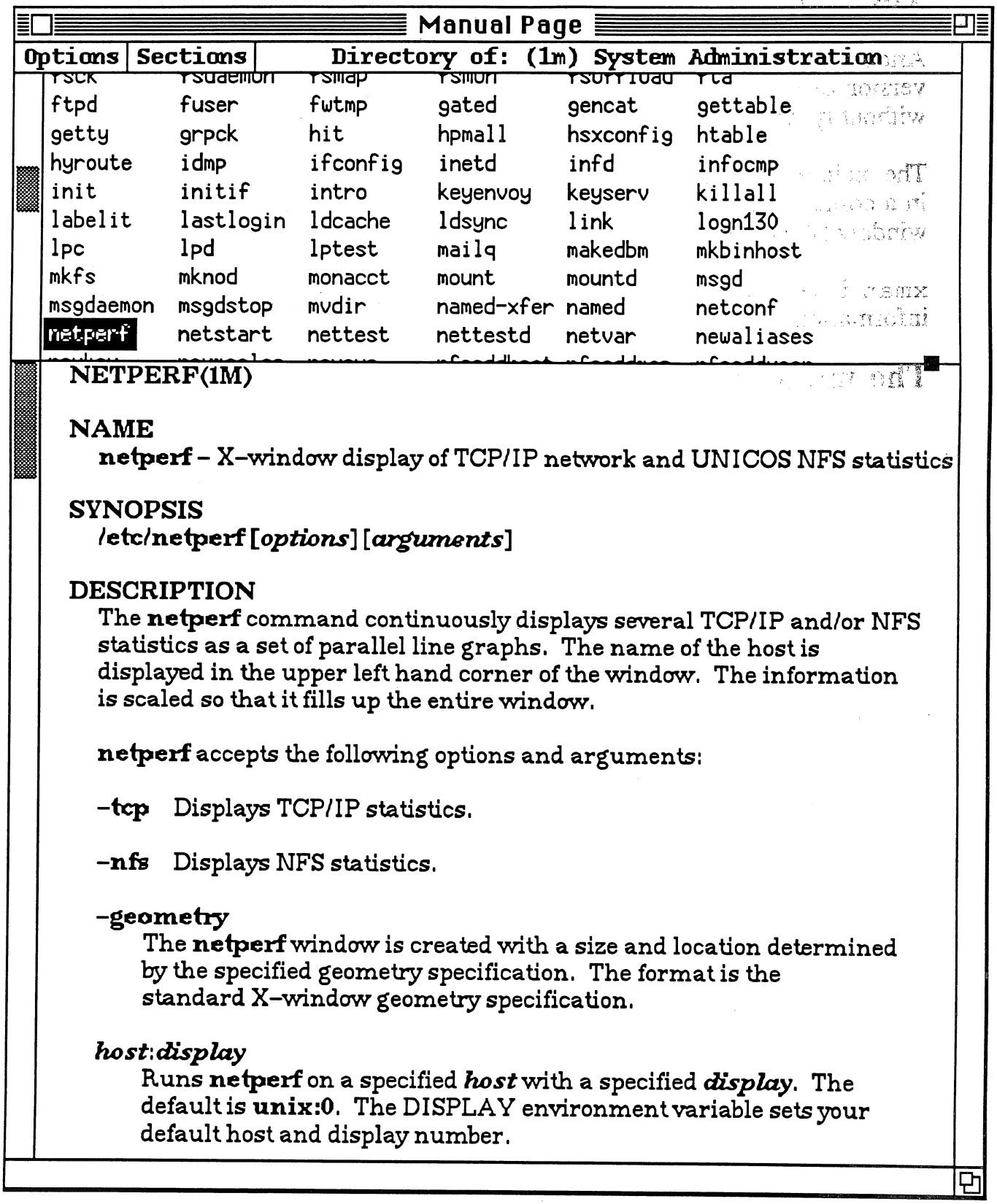
The main `xman` window, shown below, is very small and could be kept available in a corner of the screen. The data you want to view appears in another, larger, window (shown on the next page).

`xman` is customizable, so you may want to read the man page on it for more information.

The main xman window



The manual page window in xman



The `graphs` command

Another command exists on the OWS-E which provides the same basic output as the `graph` option in `crayperf`. It is the **`graphs`** command. It is very easy to run.

See the man page for `graphs` for more information or read the Monitors section in the OWS-E Operator Workstation Operator's Guide (SG-3078) for examples.

Useful Manuals

Useful manuals

These manuals should be kept near your operator consoles:

SR-2011	UNICOS User Commands Reference
SQ-2056	UNICOS User Commands Ready Reference
SQ-2060	UNICOS Shell and Variable Ready Reference
SQ-2054	UNICOS vi Text Editor Ready Reference Card
SR-2022	UNICOS Administrator's Command Reference
SR-3077	OWS-E Operator Workstation Reference Manual
SG-3078	OWS-E Operator Workstation Operator's Guide
SG-3079	OWS-E Operator Workstation Administrator's Guide
SG-2009	TCP/IP Network User Guide

Manuals for users with computers that are networked to the Cray:

SG-0057	CDC NOS Station Operator's Guide
SG-0058	CDC NOS/BE Station Operator's Guide
SC-0271	CDC NOS/VE Link Software Operator's Guide
SV-0063	DEC VAX/VMS Station Operator's Guide
SI-0037	BM MVS Station Operator's Guide
SI-0192	Superlink Operator's Guide
SG-2064	TCP/IP Network Administrator's Guide

X Window System manual

Quercia, Valerie, and Tim O'Reilly. *X Window System User's Guide for X11 R3 and R4*. Sebastapol, CA, 1988, 1989, 1990

Cray Research, Inc.
Software Training
655-A Lone Oak Drive
Eagan, MN 55121