Lucent Technologies
Bell Labs Innovations

3B20D and 3B21D Computers *UNIX*® RTR Operating System

Software Troubleshooting Guide

254-303-107
Issue 1
October 1997

Notice

Lucent Technologies is the successor to the business and assets of AT&T Network Systems Business Unit. Every effort was made to ensure that the information contained in this IP was complete and accurate at the time of printing. However, information is subject to change.

This IP describes certain hardware, software, features, and capabilities of Lucent Technologies products. This IP is for information purposes, and you are cautioned that the hardware, software, features, and capabilities described herein may not be the same as the product you have.

Trademarks

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

All other trademarks are identified as they appear in the text.

Ordering Information

This IP is distributed by the Lucent Technologies Customer Information Center in Indianapolis, Indiana. Orders for additional copies may be placed from the continental United States by calling 1 888-LUCENT8 (1 888 582-3688) or faxing to 1 317 322-6484. Customers outside the continental United States may place orders by contacting the in-country Lucent Technologies market manager. Lucent Technologies personnel outside the continental United States can order additional copies by faxing to 1 317 322-6699.

Information Support Telephone Number

Call 1 888-LTINFO6 (1 888 584-6366) to report errors or to ask questions about this IP.

Developed by Lucent Technologies Network Systems Customer Training & Information Products.

How Are We Doing?

Title: 3B20D and 3B21D Computers
UNIX® RTR Operating System
Software Troubleshooting Guide

Identification No.: 254-303-107 Issue 1 Date: October 1997

Lucent Technologies welcomes your feedback on this information product (IP). Your comments can be of great value in helping us improve our IPs.

1. Please rate the effectiveness of this IP in the following areas:

	Excellent	Good	Fair	Poor	Not Applicable
Ease of Use					/////////////////////
Clarity					/////////////////////
Completeness					/////////////////////
Accuracy					/////////////////////
Organization					/////////////////////
Appearance					/////////////////////
Examples					
Illustrations					
Overall Satisfaction					/////////////////////

2. Please check the ways you feel we could improve this IP:

□ Improve the overview/introduction
□ Improve the table of contents
□ Improve the organization
□ Include more figures
□ Add more examples
□ Add more detail

□ Make it more concise/brief
□ Add more step-by-step procedures/tutorials
□ Add more troubleshooting information
□ Make it less technical
□ Add more/better quick reference aids
□ Improve the index

Please provide details for the suggested improvement. ________________________________
____

3. What did you like most about this IP?
____
____

4. Feel free to write any comments below or on an attached sheet.
____
____
____
____

If we may contact you concerning your comments, please complete the following:

Name: ______________________________Telephone Number: ______________
Company/Organization: ____________________________Date: ______________
Address: ____

When you have completed this form, please fold, tape, and return to address on back
or Fax to: 910 727 3043.

Do Not Cut—Fold Here And Tape

Lucent Technologies
Bell Labs Innovations

DOCUMENTATION SERVICES
2400 Reynolda Road
Winston-Salem, NC 27199-2029

How Are We Doing?

Title: 3B20D and 3B21D Computers
UNIX® RTR Operating System
Software Troubleshooting Guide

Identification No.: 254-303-107 Issue 1 Date: October 1997

Lucent Technologies welcomes your feedback on this information product (IP). Your comments can be of great value in helping us improve our IPs.

1. Please rate the effectiveness of this IP in the following areas:

	Excellent	Good	Fair	Poor	Not Applicable
Ease of Use					////////////////////
Clarity					////////////////////
Completeness					////////////////////
Accuracy					////////////////////
Organization					////////////////////
Appearance					////////////////////
Examples					
Illustrations					
Overall Satisfaction					////////////////////

2. Please check the ways you feel we could improve this IP:

□ Improve the overview/introduction
□ Improve the table of contents
□ Improve the organization
□ Include more figures
□ Add more examples
□ Add more detail

□ Make it more concise/brief
□ Add more step-by-step procedures/tutorials
□ Add more troubleshooting information
□ Make it less technical
□ Add more/better quick reference aids
□ Improve the index

Please provide details for the suggested improvement. ______________________________
____

3. What did you like most about this IP?
____
____

4. Feel free to write any comments below or on an attached sheet.
____
____
____
____

If we may contact you concerning your comments, please complete the following:

Name: ____________________________________ Telephone Number: ________________

Company/Organization: ________________________________ Date: ________________

Address: ____

When you have completed this form, please fold, tape, and return to address on back
or Fax to: 910 727 3043.

---Do Not Cut—Fold Here And Tape---

Lucent Technologies
Bell Labs Innovations

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1999 GREENSBORO, N.C.

POSTAGE WILL BE PAID BY ADDRESSEE

DOCUMENTATION SERVICES
2400 Reynolda Road
Winston-Salem, NC 27199-2029

Contents

Contents

Contents

Figures

1 *browse* Command

3 *cx* Command

4 Generic Access Package

5 *idump* Command

Figures

Tables

About This Information Product

1 *browse* Command

4 Generic Access Package

6 Kernel Information

Tables

10 Descriptions of Register Layouts

11 Conversion Chart

About This Information Product

Purpose

The Software Troubleshooting Guide is an information product (IP) that is part of a documentation set used to support the *UNIX** Real-Time Reliable (RTR) operating system running on Lucent Technologies 3B20D and 3B21D computers.

Audience

This IP provides information for a variety of audiences. Anyone interested in the *UNIX* RTR operating system, 3B20D/3B21D computer, will find it useful. The primary audience for this IP includes the following:

- Craft or maintenance personnel
- System administrators
- 3B20D/3B21D computer documentation users.

* UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Reason for Reissue

This IP is being reissued to incorporate new information about the small computer system interface (SCSI) disk drives and to bring the IP into compliance with Lucent Technologies documentation standards. The major changes are as follows:

- Delete Chapter 6 "Microlevel Test Interface Program".
- Delete Chapter 7 "DART".
- Add Chapter 5 "***idump*** Command".
- Add Chapter 9 "Release 21 Hexadecimal Offset Charts".

Change bars on the far right-hand side of the page are used to denote changes made to the current issue. Change bars will help you understand at a glance what has changed from the previous issue. The position of deleted text is shown by a single asterisk (*). Not all editorial changes will be marked with change bars. Occasionally, change bars do not work in tables or figures; therefore, those changes will be shown by adding a change bar only to the table or figure title.

Change bars (|) on the far right-hand side of the text are used to denote changes made to the current issue. Change bars will help you understand at a glance what has changed from the previous issue. The position of deleted text is shown by a single asterisk (*). Some editorial changes will not be marked with change bars. At times change bars do not work in tables or figures. In those cases, changes will be shown by adding a change bar to the table or figure title only.

How to Use This Information Product

This IP, a guide to debugging software running on the Lucent Technologies 3B20D/3B21D computer, does not give detailed procedures; rather, it gives general information on how to use tools provided for debugging purposes. This IP is composed of independent chapters that are referenced in the Table of Contents. Also, specific topics are referenced in the index.

The chapters in this IP are organized as follows:

- About This Information Product
- Debugging Tools
  - — Chapter 1 — ***browse*** Command
  - — Chapter 2 — ***ibrowse*** Command

- Chapter 3 — ***cx*** Command
- Chapter 4 — Generic Access Package
- Chapter 5 — ***idump*** Command

Chapters 1 through 5 give information on using the software debugging tools available on the 3B20D/3B21D computer.

NOTE:
The remote debugging tools described in the following section should only be used if the 3B20D/3B21D computer has stopped processing or processing is seriously degraded.

- Kernel Information
  - Chapter 6 — Kernel Information

    Chapter 6 lists the operating system trap (OST) service routines provided for the kernel and supervisor processes and defines the segments and externally declared data areas in the kernel address space.

- Hexadecimal Offset Charts
  - Chapter 7 — Release 1 Hexadecimal Offset Charts
  - Chapter 8 — Release 6 Hexadecimal Offset Charts
  - Chapter 9 — Release 21 Hexadecimal Offset Charts

    Chapters 7 through 9 list the control block structures in the kernel, kboot, and file manager address spaces for Release 1, Release 6, and Release 21, respectively.

- Register Layouts
  - Chapter 10 — Brief Descriptions of Register Layouts

    Chapter 10 provides brief descriptions of registers used while troubleshooting sofware problems.

- Conversion Chart
  - Chapter 11 — Conversion Chart

    Chapter 11 gives an American Standard Code for information Interchange (ASCII)/Hexadecimal conversion chart for reference.

- Glossary

Conventions Used

Command, Filename, and Display Notations

The following notations are used to show commands and filenames in the text and displays.

NOTE:
System filenames and command names are case-sensitive, so you must enter them exactly as they are shown.

- Command names in text appear in **bold** type; for example, the **/usr/bin/ls** command. Command names in headings appear in ***bold-italic*** type.
- Filenames in text appear in *italic* type; for example, the */usr/lib/uucp/System* file. Filenames in headings appear in *italic* type.
- Text that you enter, such as a command or response to a prompt, appears in **bold** type; for example, the **ls -la** command.
- Variables that appear in a command line or file appear in *italic* type; for example, **grep** *username* **/etc/passwd**. In this example, *username* is a variable indicating a user's name is required.
- Screen displays and system messages appear in `constant width` type; for example, `Please enter your password.` Program code listings and file listings are also shown in `constant width` type. Input messages are shown in **`constant-width bold`** type.
- Comments and explanations within a display are indented and shown in *italic* type. These are for information only and will not appear on your screen.
- A line in a file or on the computer screen that is too long to be shown as it actually appears in this CIP will be shown with a backslash (\) at the end of the first line. This indicates the next line should be read as a continuation of the current line.
- Square brackets around an argument on a command line indicate that the argument is optional; for example, the **lpstat** [**-t**] command. In this example, the **-t** argument is optional and can be omitted.
- A vertical bar (|) between words in an argument on a command line indicates that one of the arguments is to be selected.
- The key identified on your keyboard as Return, Enter, or a bent arrow (↵) is referred to as the Return key. Occasionally, representations of this Return key will be boxed; for example, **Return**.

- There is an implied Return at the end of each command and menu response that you enter. Some examples do not explicitly show the Return. Where you may be expected to enter a Return (as in the case where you are accepting a menu default), the symbol <CR> is shown to indicate that you are to press the Return key.
- Key combinations appear in a hyphenated format; for example, Ctrl-d. Press and hold down the first key of a key combination while pressing the second key.
- Ellipses (three dots) on a command line indicate that the previous argument can be repeated; for example, **ls** [*file* ...]. In this example, multiple files can be listed after the command.
- References to manual pages are followed by their manual page location number in parentheses; for example, **mount**(1M).

Hexadecimal Notation

Hexadecimal (base 16) numbers are denoted with a **0x** prefix; for example, 0x00A is decimal 10.

Signal Designations

The term ‘‘asserted’’ is used in the descriptions to mean that a signal is driven to its active state. The term ‘‘negated’’ is used in the descriptions to mean that a signal is driven to its inactive state.

Signal names used in the diagrams and descriptions that end in a 0 or 1 indicate the active state of the signal. Names ending in ‘‘0’’ are ‘‘active low’’ signals; names ending in ‘‘1’’ are ‘‘active high’’ signals. For example, CCIOD(31-00)1 describes the Central Control Input/Output Data bits 31 through 00 which are ‘‘active high’’ signals.

Equipment Locations

A coordinate numbering system is used to identify Equipment Locations (EQLs) in units and cabinets. The origin is the lower left front of the cabinet or unit. Vertical increments are measured in inches. Horizontal increments are measured in eighths of an inch. The coordinate location of a circuit pack is expressed as the horizontal and vertical location of the center lines of the connector into which the circuit pack is inserted. The location of a unit in a cabinet is identified by the placement of the lower left corner of the unit in the cabinet.

For example, a connector at EQL 004-080 is located 4 inches above the origin and 10 inches (80 x 0.125 = 10.0) to the right of the origin.

Conventions Used

Admonishments

Admonishments are reminders used to assure the safety of personnel and to minimize service interruptions, loss of data, and damage to equipment, products, and software.

Three types of admonishments are used in Lucent Technologies documentation. The three types, in descending order of priority, are as follows:

1. **DANGER** indicates the presence of a hazard that **will** cause death or severe personal injury if the hazard is not avoided.
2. **WARNING** indicates the presence of a hazard that **can** cause death or severe personal injury if the hazard is not avoided.
3. **CAUTION** indicates the presence of a hazard that **will** or **can** cause minor personal injury OR property damage if the hazard is not avoided.

This document does not contain admonishments.

Related Information Products

Table 1 lists by IP number and Select Code the Lucent Technologies IPs supporting the 3B21D computer. IPs with Select Codes will be converted to nine-digit IP numbers as they are reissued.

Table 1. IPs Supporting the 3B21D Computer

NEW IP NUMBER	OLD SELECT CODE	TITLE
254-001-014	-	3B20D and 3B21D Computers Equipment Test List
254-303-100	-	3B21D Computer Growth/Retrofit Tasks
254-303-101	-	3B21D Computer Routine Maintenance Tasks
254-303-102	-	3B21D Computer Trouble Clearing Tasks
254-303-103	303-007	3B20D and 3B21D Computers *UNIX* RTR Operating System Processor Recovery Messages Guide
254-303-104	303-010	3B20D and 3B21D Computers
254-303-105	304-045	3B21D Computer Hardware Reference Manual
254-303-106	304-046	3B20D and 3B21D Computers *UNIX* RTR Operating System System Maintenance Manual
254-303-107	303-072	3B20D and 3B21D Computers *UNIX* RTR Operating System Software Troubleshooting Guide
254-303-110	303-080	3B20D and 3B21D Computers *UNIX* RTR Operating System PDS Input Messages Manual
254-303-111	303-081	3B20D and 3B21D Computers *UNIX* RTR Operating System PDS Output Messages Manual
254-303-112	303-082	3B20D and 3B21D Computers *UNIX* RTR Operating System MML Input Messages Manual
254-303-113	303-083	3B20D and 3B21D Computers *UNIX* RTR Operating System MML Output Messages Manual

How to Comment on This Information Product

Lucent Technologies welcomes your comments on this IP. Your comments will aid us in improving the quality and usefulness of Lucent Technologies documentation. Please use the Feedback Form provided at the front of this IP. If the Feedback Form is missing, mail your comments to the following address:

Lucent Technologies
Customer Training and Information Products (CTIP)
2400 Reynolda Road
Winston-Salem, NC 27106-4606

Lucent Technologies also provides a support telephone number to use for reporting errors or asking questions about information contained in this IP. The support telephone numbers is 1-888-LTINFO6 (1-888-584-6366).

How to Order Information Products

The ordering number for this IP is 254-303-107, Issue 1.

To place an order by mail, write to the following address:

Lucent Technologies
Customer Information Center
Attention: Order Entry Section
2855 North Franklin Road
P.O. Box 19901
Indianapolis, IN 46219

Telephone orders may be placed Monday through Friday. To place an order by telephone, call one of the following numbers:

Ordering by telephone within the United States, including Hawaii and Alaska, along with Canada:

1-888-LUCENT-8 (1-888-582-3688)

Ordering by fax within the United States, including Hawaii and Alaska, along with Canada:

1-800-566-9568

Mandatory Customer Information

The 3B21D computer is used as the Administrative Module (AM) in various switching system applications and is not provided as a stand-alone product. Therefore, the application documentation is responsible for providing this information.

Refer to the applicable application documentation for ‘‘Mandatory Customer Information.’’

browse Command

1

Contents

browse Command

1

Overview

The **browse** command allows the interactive perusal of low-level access (LLA) databases and the formatted output of user data and LLA internal structures. It is also used to:

- Verify data
- Find corrupted structures
- Gather information
- Repair damage.

The support computer versions of **browse** examine files in the software demand paging (SDP) address space format, files in loadf format, and ordinary files. The 3B20D computer version examines files in all of the previously mentioned formats and incore copies of loadf and ordinary files.

Invocation and Input

Under the Real-Time Reliable (RTR) shell, the **browse** command is invoked by entering:

browse [%][+-]*name*

where:

+ indicates *name* is an ordinary file.
- indicates *name* is a loadf file.
% indicates named ordinary file or loadf file resides in core (3B20D/3B21D computer only).

If *name* is not preceded by a + or -, *name* is in SDP address space format.

If *%* is used, *name* must have one of the following forms:

ecd
pmdb
<filename>,<index>,<segno>
<segname>,<segno>

where:

ecd =
ECD incore database (segment name defined in header file *<init_btdb.h>*).

pmdb =
Plant measurement incore database (segment name defined in header file *<init_btdb.h>*).

filename =
Segment names associated with the file system.
and
index

segname =
32-bit number by which the system names segment.

segno =
Index of the segment in the virtual address space.

The program has read only permission to the file, database, or segment.

Under the program documentation standard (PDS) shell, the **browse** command is invoked by entering:

RCV:MENU:BROWSE!

Under the man-machine language (MML) shell, the **browse** command is invoked by entering:

RCV:MENU:DATA,BROWSE!

The **browse** command may not be invoked from the maintenance teletypewriter (MTTY) terminal or a switching control center (SCC) terminal.

Specification of a database is the same as in the Bourne shell but must be done via the **browse** command **db**. (See the ‘‘Printing’’ section.) Table 1-1 summarizes the invocation modes.

Table 1-1. Database Invocation Modes

	Incore (% prefix)	**On disk (no prefix)**
<name>	unsupported	3B20D/3B21D
-<name>	3B20D/3B21D computer	3B20D/3B21D
+<name>	3B20D/3B21D computer	3B20D/3B21D

browse accepts commands from standard input as follows:

*<expr>***<cmd>***<str>*

Expressions

An expression *<expr>* has the syntax shown in Figure 1-1.

```
<expr> ::
           <expr> <op> <expr>

        | <constant>

        | /<format> <values>/

        | ?<format> <values>?

        | ( <expr> )

<op>  ::
        + | - | * | / | % | , | ;

<constant> ::
           <decimal digits>

        |0<octal digits>

        |0x<hex digits>

        |<char>

        |.

        |$
```

Figure 1-1. Expression Syntax

The operators and digit strings have their standard meanings.

Because printing, searching, and division share the slash character, that character’s use as an operator may require enclosure in parentheses to avoid ambiguity.

A <char> is a C-Language character representation as shown in Table 1-2. Therefore, *’8’-060-o* prints *010* because the value of character *’8’* is 070.

Table 1-2. ’c C-Language Character Format

’\0’	’ ’	’@’	’_’
’\01’	’!’	’A’	’a’
’\02’	’"’	’B’	’b’
’\03’	’#’	’C’	’c’
’\04’	’$’	’D’	’d’
’\05’	’%’	’E’	’e’
’\06’	’&’	’F’	’f’
’\07’	’\’	’G’	’g’
’\b’	’(’	’H’	’h’
’\t’	’)’	’I’	’i’
’\n’	’*’	’J’	’j’
’\013’	’+’	’K’	’k’
’\f’	’,’	’L’	’l’
’\r’	’_’	’M’	’m’
’\016’	’.’	’N’	’n’
’\017’	’/’	’O’	’o’
’\020’	’0’	’P’	’p’
’\021’	’1’	’Q’	’q’
’\022’	’2’	’R’	’r’
’\023’	’3’	’S’	’s’
’\024’	’4’	’T’	’t’
’\025’	’5’	’U’	’u’
’\026’	’6’	’V’	’v’
’\027’	’7’	’W’	’w’
’\030’	’8’	’X’	’x’
’\031’	’9’	’Y’	’y’
’\032’	’:’	’Z’	’z’
’\033’	’;’	’[’	’{’
’\034’	’<’	’\’	’|’
’\035’	’=’	’]’	’}’
’\036’	’>’	’^’	’~’
’\037’	’?’	’_’	’\177’

An item enclosed in slashes (/) is the next address with given values; an item enclosed in question marks (?) is the previous address with the given values; wraparound applies to searching in both directions. Therefore, */d 8/* locates the next short integer 8; *?d 8?* locates the previous integer 8. The value does not have to match the format: */d 010/* locates the next 8. The value can also be an expression; for example, */d ’8’-060/* also locates the next 8.

An empty *<format><value>* list refers to the list specified previously.

The dot (.) is the current address; the dollar sign ($) is the number of bytes in the file or address space. (Because browse addressing is zero based, $-1 is the highest address.)

Evaluation is left to right, with the only precedence established by parentheses. All computations are performed with long operands.

Format Usage

A slash (/) following an expression prints database information at the address equal to the value of the expression. The address formats, following the slash, control the printing and have the syntax shown in Figure 1-2.

```
<aformat>  ::          <item>
                     | <item> <aformat>
<item>  ::             <term>
                     | <count> <term>
<term>  ::             ( <aformat> )
                     | [ <aformat> ]
                     | < <aformat> >
                     | { <aformat> }
                     | <byteformat>
                     | <memformat>
                     | <numformat>
                     | <specialformat>
                     | <userformat>
                     | "any characters"
<byteformat>  ::     ´a | ’c | ’d | ’o | ’u | ’x | b | c
<memformat>  ::      .<C structure member identifier>
<numformat>  ::      D | d | I | i | O | o | U | u | X | x
<specialformat>  ::  A | B | C | H | L | Q | R | S | T | r | s
<userformat>  ::     E | F | G | J | K | M | N | P | V | W | Y | Z
<count>  ::          Positive decimal number
```

Figure 1-2. Format Syntax

The format letters and grouping characters have the meanings shown in Table 1-3 and Table 1-4, respectively.

Table 1-3. Format Letters

Letter	Meaning	Number of Bytes
A	access method	sizeof(ABLOCK)
’a	ASCII character	sizeof(char)
B	hash bucket	sizeof(struct bucket)
b	byte	sizeof(char)
C	SDP header	sizeof(struct SPACE)
c	character	sizeof(char)
’c	C character	sizeof(char)
D	decimal	sizeof(long)
d	decimal	sizeof(short)
’d	decimal	sizeof(char)
H	header	sizeof(DMLHEAD)
I	SDP id	sizeof(ITEMID)
i	integer	sizeof(int)
L	rid block	sizeof(RBHEAD)
O	octal	sizeof(long)
o	octal	sizeof(short)
’o	octal	sizeof(char)
Q	queue or stack	sizeof(struct quest)
R	record header	sizeof(RECHEAD)
r	record	-
S	set	sizeof(SETHEAD)
s	string	strlen(string)+1
T	btree	sizeof(struct bt_node)
U	unsigned	sizeof(long)
u	unsigned	sizeof(short)
’u	unsigned	sizeof(char)
X	hexadecimal	sizeof(long)
x	hexadecimal	sizeof(short)
’x	hexadecimal	sizeof(char)
=	current location	0

Table 1-4. Grouping Characters

Grouping	Meaning
()	establish scope of count
[]	go indirect from start of record
<>	suppress printing
{}	go indirect on current address

A count in front of an item is equivalent to repeating that item the given number of times. For example, 2(DX3(bc)) is equivalent to DXbcbcbcDXbcbcbc.

Format items enclosed in square brackets ([ ]) indicate that the item refers to an address offset from the start of the current record. The square brackets are useful for display of variable length data definition language (DDL) constructs (vectors and strings) which are stored offset from the fixed portion of a record.

Format items enclosed in corner brackets (<>) suppress the printing of their corresponding values. For example, to examine the first three characters of a 10-character array and the following integer, use *./3c<7c>d*.

Format items enclosed in braces ({ })apply the format to the address given by the value at the current address. Thus, */R* prints a record header at the current address; */{R}* prints a record header indirect from the current address.

The equal sign (=) prints in *I* format the current location (that is, current address plus current offset). Therefore, the command *./100(5X"\n"=":\t")* prints 100 lines of hexadecimal dump format, and each line is preceded by the address of the first word on the line.

Nonreserved capital letters (those in *<userformat>*) may be given definitions. For example, stating *J DbbX* defines a new format J which prints a long decimal, two bytes, and a long hexadecimal. The reserved format *I* is also settable. You may examine ITEMIDs in decimal, octal, unsigned, or hexadecimal by setting *I* to *D*, *O*, *U*, or *X*, respectively; the initial format is hexadecimal. The *I* format also controls current address printing, the *i* format, and ITEMID fields in structures.

The format associated with K applies to the key portion of the *buckets* of hash and *btree* access methods; the initial format is **n***b*, where **n** equals *KEYMAX,* the LLA *#define* constant giving the maximum storage allowed for keys. If the *K* format is empty, then the *B* and *T* formats print only the bucket and node headers respectively. In this instance, the *T* format indents the headers to reflect the depth of each node in the tree.

Arithmetic and character formats print individual bytes. The formats *d, 'o,* and *'x* print a byte in decimal, octal, and hexadecimal, respectively. The *b* format prints a byte in the last arithmetic byte format entered; the initial format is octal. The formats *'a* and *'c* print a byte in American Standard Code for Information Interchange (ASCII) mnemonic form and C-Language form, respectively. The ASCII mnemonics are those established in the file */usr/pub/ascii.* The *c* format prints a byte in the last character format entered; the initial format is ASCII mnemonic (see Table 1-5).

Table 1-5. ’a ASCII Mnemonic Character Format

nul	sp	@	_
soh	!	A	a
stx	"	B	b
etx	#	C	c
eot	$	D	d
enq	%	E	e
ack	&	F	f
bel	’	G	g
bs	(	H	h
ht	)	I	i
nl	*	J	j
vt	+	K	k
np	,	L	l
cr	-	M	m
so	.	N	n
si	/	O	o
dle	0	P	p
dc1	1	Q	q
dc2	2	R	r
dc3	3	S	s
dc4	4	T	t
nak	5	U	u
syn	6	V	v
etb	7	W	w
can	8	X	x
em	9	Y	y
sub	:	Z	z
esc	;	[	{
fs	<	\	|
gs	=	]	}
rs	>	^	~
us	?	_	del

browse echos literal text between double quotes ("). The backslash (\) affects the escape sequence shown in Table 1-6.

Table 1-6. browse Escape Sequences

Escape	Meaning
\b	backspace
\f	linefeed
\n	newline
\r	carriage return
\t	tab
\"	double quote
\\	backslash
\ddd	octal byte

browse prints strings with the *s* format using the same conventions.

Printing

A slash (/) following an expression causes database information to print at the address equal to the value of the expression. The formats following the slash control printing.

Printing database information depends on two automatically calculated values: the current address and the current offset. Each format item prints the information at its target location which is the current offset from the current address. A slash following an expression establishes the value of the expression as the current address and sets the current offset to zero. Generally, each format letter increments the current offset by the number of bytes it prints (see Table 1-2), and a carriage return alone increments the current address by the current offset. The effect of these computations is that stringing together format items prints sequentially through the database. An example of a formatted printout is shown in Figure 1-3 which illustrates printing seven items starting at address 100.

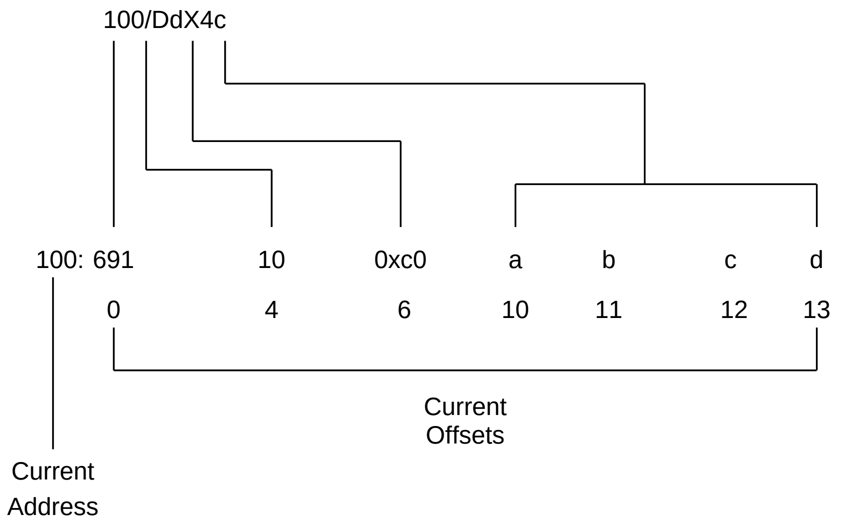


Figure 1-3. Formatted Print Example

browse prints the address followed by a colon (:) and then the information from the database in the indicated format. The vertical lines in the diagram show the relationship between format item and printed value. Pressing the return key again yields:

```
114:   692   11   xc1   b   c   d   e
```

if the fields starting at 114 have values one more than their corresponding fields starting at 100.

Four exceptions to the description of address calculations are as follows:

- Linked structures buckets, rid blocks, queues/stacks, and sets (formats *B*, *L*, *Q*, and *S*, respectively). In these formats, the computation of the current offset is made so that the next item printed is the next structure on the linked list, not the next sequence of logically contiguous bytes. If 044 is the address of the first set, the command *044/3S* prints the first three sets. Use right recursive format definition for these formats. If *J* is defined as *SJ*, the command *044/J* prints all the sets.
- Items in square brackets or braces. The items within square brackets are used to calculate a new target location offset from the current record by the value in the target location. To get a current record, use R format. The items within braces are used to calculate a new target location at the address given the current address itself. Printing begins at the new target location obtained by the indirection. The altered offset has effect only

within the brackets or braces. An item enclosed in brackets is treated as having length *sizeof(int);* an item enclosed in braces has length *sizeof(ITEMID)*.

Figure 1-4 is an example of a location printed with *R* format. It indicates that the user area of the record begins at address 100 in the database.

The [2c] prints two characters at offset 10 (the value at 104) from address 100 (the start of the record). The format item X following the [2c] resumes printing at 106 as though the bracketed item were a simple integer field. In this example, the **100/{c}[2c]X4c** command is equivalent to the consecutive commands **691/c** and **104/[2c]X4c.** The indirect formats are useful for printing LLA vectors, which consist of fixed and variable portions. The variable portions are located at positions determined by the fixed portion.

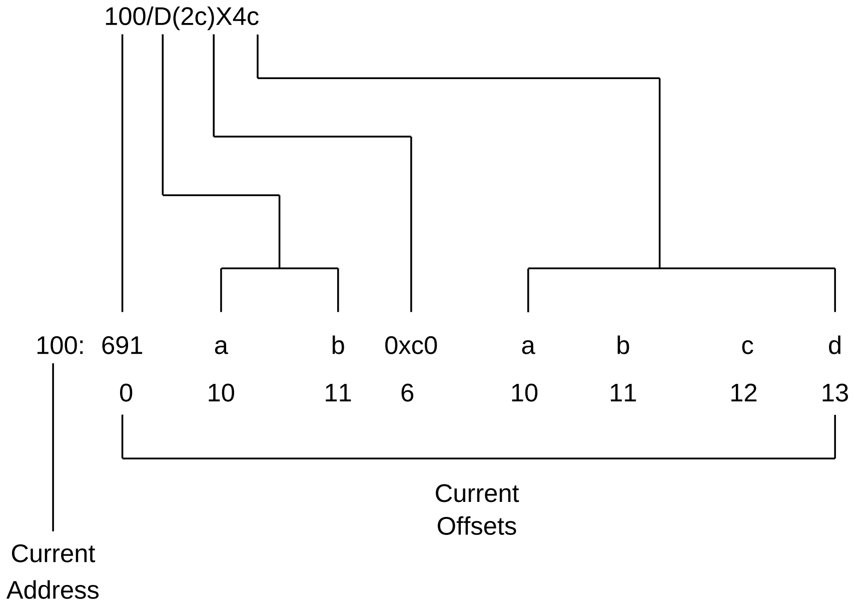


Figure 1-4. Indirection Print

- Btree access method (format *T*). The *T* format prints in depth-first order the subtree with root at the current address. The *T* format changes neither current address nor current offset.

- Formats for symbolic record printing. If an *r2a* process is started via the **dd** command, then at the address of a record, the *r* format prints all record members and their values. A dot (.) followed by a name prints only the names and values of record members with matching names. In either case, current address and offset are not affected.

browse Commands

browse commands take the following form:

<expr> **<command>** <parameters>

In the list below, default addresses are enclosed in braces and are *not* part of the command.

<

The **<** command causes **browse** to interpret a trailing string as a filename from which input is accepted. If the command itself appears in the named file, **browse** places the new file on a stack and switches input to the new file. An end of file (EOF) pops the stack. A missing name causes a switch to standard input.

>

The **>** command interprets a trailing string as a file name or a shell command to which standard output is directed. If the trailing string starts with an !, then it names a shell command; otherwise, it names a file. The special name *stderr* redirects output to stderr instead of to a file. If the trailing string is omitted, output returns to standard output. An interrupt redirects output to standard output unless a previous redirection was to *stderr*. The **>>** command is the same as **>**, except that output to a file is appended.

>>

The **>>** command is the same as **>** command, except that output is appended to the file.

{.}#/<formats>

The patch command substitutes the values in the given list in the indicated formats for the values at the given address. The value list is a space-separated list of expressions, each of which must correspond to a printing item in the format. The *s* format patches a number of bytes equal to the length of the string following the format. For a patch, **browse** prints:

<addrs>: <old value list> ← *<new value list>*

The format letter controls message printing. The *s* format patches a variable number of bytes and care must be taken with its use.

An example using the patch command is provided at the end of this section.

{last}=<format>

An equal sign (=) between an expression and a format controls expression value printing. The format may be any letter in <pformat> except *s*. In the character format, non-ASCII values are printed in octal; ASCII characters are printed either as C language constants or as ASCII mnemonics (depending on the last given 'a or 'c format). An octal format forces a leading zero in the print; a hexadecimal format, a leading *0x*. Thus, *4+6*2=x* prints *0x14*.

!<string>

Submits the string to the shell.

<cap>

Capital (**cap**) letter commands associate the letter with the trailing string. Appearances of the capital letter in formats are replaced by the associated string. The replacement is recursive: appearances of defined capitals may appear in other capital commands. Left or circular recursion in formats leads to disaster. If the trailing string is omitted, the current value of the macro letter is reported; if the string is white space, the macro is cleared.

db

If a name follows the database (**db**) command, then **browse** disconnects any currently attached database and attempts to attach the named database with no permission to patch. If no string follows the **db** command, then the currently attached database name and its access permissions are printed. [The code (#) indicates permission to patch.]

dbp

If a name follows the database patch (**dbp**) command, then **browse** disconnects any currently attached database and attempts to attach the named database with permission to patch. If no name follows the **dbp** command, then **browse** toggles the access permission of the currently attached database. In either case, **dbp** reports the currently attached database and its access permissions.

An example using the **dbp** command is provided at the end of this section.

dd

If a name follows the data dictionary (**dd**) command, then **browse** starts the named dictionary process. This process, which must be generated by *r2agen,* provides symbolic information about records. With a data dictionary process, the user may print entire records by member name and value via the *r* format, print single or related sets of record members with their values via the *.name* format, and patch single record members by name. The data dictionary process also augments the R and S formats by including the record and set names, respectively. The name may be followed with a number, interpreted as the number of tab positions after which to place values. This number is helpful in formatting values in record prints. If the **dd** command is not given a name, then it reports the name and tab stop of the current process.

An example using the **dd** command is provided at the end of this section.

files

Reports the stack of input files, most recent last.

framesize

Set the frame size for the internal pager. For an SDP address space, the frame size must be a multiple of the page size with which the space was generated.

g/<formats>

The global (**g**) command searches the addresses in the database which have the given values and performs the given command at those addresses. More than one command may be given on succeeding lines if a backslash terminates the previous line. The value list is a space-separated list of expressions, each of which must correspond to a printing item in the format. The delimiting slashes may be uniformly replaced by any other character except a backslash. For example,

g/X<4c>d 0x210 6/ .+8#/d 8/

looks for addresses that contain a long 0x210, any 4 characters, and a short 6 and then patches the 6 to an 8. Note that there are 2 printing formats (*X* and *d*) and 2 values *(0x210* and *6*). Because the values may be expressions, the 0x210 of the above example could be replaced by (256*2)+16.

h

Reports the address of the LLA header structure.

help

Prints a summary of commands and formats.

macinfo

Reports the values of the user-settable format letters.

q

Disconnects a currently attached database and exits. A **q** command is equivalent to an end of tape (EOT) (Ctrl-d) when there are no files from a **<** command on the input file stack.

r/<formats>

Given a trailing string composed of a format and a list of values, the record command searches the database for the occurrence of the values in a record and executes the given command when a match is found. The match is located at the beginning of the record. Multiple commands may be given on succeeding lines when the previous line ends in a backslash. The formats and values are the same as in the global search request. The default command prints the record identification (RIDS) numbers of matching records. An empty format/value list matches any record. Therefore, the command **r /r** matches all records (the first *r/*|) and prints *(/)* them symbolically (the second *r/*|).

Example of a ***browse*** Session

The following is an example of how to change the *u_admin* bit on a unit control block (UCB) record using **browse**. The *u_admin* bit is set by maintenance jobs such as diagnostics in order to reserve the unit. In some cases, the u_admin or reserve bit can get "stuck." The **/bin/ducb** command with the "rel" option will generally clear this bit for you, but the following procedure shows how to clear or unreserve a unit using **browse**.

Problem

Here an attempt is made to restore MHD 1. The restore fails because the u_admin (or reserve bit) is already set in the *ucb* for moving head disk (MHD) 1.

```
< rst:mhd 1
 PF
006856 97-06-23 13:31:14 sslcu3-m2

M 31 REPT MIRA CANNOT RESERVE UNITS FOR RST MHD 1
```

Prelimnary Steps

To use the **browse** display commands we need to first get the RID (record ID) number for the *ucb* for MHD 1. This is done by entering rcvecd on the incore ecd database and using the dbinfo form.

```
rcvecd -db incore
UNIX* RTR RCV (ODIN)  -  Data Entry
=================================

Enter Form Name: dbinfo

Populate the following fields of the dbinfo form:
1.dbinfo_opf:                                    /tmp/tmpfile
21.get_form_rid                                  y
```

* UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

```
22.form_info
        type_of_form         ucb
        keyfld1                                 blank
        keyfld2                                 blank
        keyfld3                                 MHD
        keyfld4                                 1
Execute the form

##
## The following is the dbinfo form with the required fields filled in.
##
                        dbinfo     (1/6)
Database Information Form                            (Execute Only)

  1.dbinfo_opf:/tmp/tmpfile_______________________________________

   *            **  INDEX  **             *
   *                    tab (>) to:    *
   *  1) UCB List Function            2       *
   *                                  *
   *  2) IOP Device List Function       8       *
   *                                  *
   *  3) Pointer List Function         14       *
   *                                  *
   *  4) Form Rid Function             21       *
   *                                  *
                                                          .
                                                          .
                                            Skip to page 4
                                                          .
                        dbinfo     (4/6)
21.get_form_rid:y

22.form_info

type_of_form  1)ucb______

           keyfld1:___________________
           keyfld2:___________________________________________
           keyfld3:MHD________________
           keyfld4:1___
##
## Enter "*" (or SHIFT/8) to execute the form
##
type_of_form  2)*________

           keyfld1:___________________
           keyfld2:___________________________________________
           keyfld3:___________________
           keyfld4:____
```

Dump the results of the dbinfo get_form_rid which have been placed in the temp file givin in field 1 of the dbinfo form.

```
# cat /tmp/tmpfile

***************  FORM RID  ***************
```

```
***   Dbinfo Rid Request   ***

     Rid of ucb form,    MHD 1, is:  0x54d0
```

The Record ID or RID of the *ucb* for MHD 1 is 0x54d0.

Actual **browse** Session

Enter **browse** on the incore ecd database. In the following exmaple the information that is echoed back to the user is preceded by a ">>".

```
# browse %-ecd
>> %-ecd
```

Enter the following command in order to start the data dictionary auxiliary formatter. This lets **browse** print symbolic information.

```
dd /usr/bin/ecd_aux
>> /usr/bin/ecd_aux
```

Enter the following command to enter patch mode to allow changes.

```
dbp
>> %-ecd   (#)
```

Each record in the database has a record header associated with it. The record header contains important accounting information about the rec. The following command prints out the record header for the *ucb* for MHD 1. The **browse** command to print a record header is "0xrid_address/R". The rid_address is obtained using dbinfo.

```
0x54d0/R
>> 0x54d0: 0x7234:
>> cluster:       no
>> #assoc sets:   one and univ
>> rec leng type:  fixed
>> ref count:      4
>> rec length:     144
>> rec def id:     1      (ucbr)
>> rid blk id:     0x54d0
>> set id:         0x24
>> user info:      0x7248
```

From the previous data, we can see that the record header data starts at address 0x7234. The actual data in the *ucb* record begins at the "user info:" address of 0x7248.

The next command will display the RID for the *ucb* for MHD 1 as a record. This will list the actual *ucb* record data. The **browse** command format to print a record is "**0xaddr/r**".

```
0x54d0/r
>> 0x54d0: 0x7234:
>> record type:    ucbr
>> l_ucb
       >> u_c_name[0]:    nul
       >> u_c_name[1]:    nul
       >> u_c_name[2]:    nul
       >> u_c_name[3]:    nul
       >> u_c_name[4]:    nul
       >> u_c_name[5]:    nul
       >> u_c_name[6]:    nul
       >> u_c_name[7]:    nul
       >> u_c_name[8]:    nul
       >> u_c_unit:       nul
       >> u_name[0]:      M
       >> u_name[1]:      H
       >> u_name[2]:      D
       >> u_name[3]:      nul
       >> u_name[4]:      nul
       >> u_name[5]:      nul
       >> u_name[6]:      nul
       >> u_name[7]:      nul
       >> u_name[8]:      nul
       >> u_unit: soh
       >> u_top:  OFF
       >> u_unique:       ON
       >> u_pseudo_node:  OFF
       >> u_restorable:   ON
       >> u_removable:    OFF
       >> u_rmvd: MANRMVD
       >> u_dport:        PT_DISK
       >> u_dtype:        DEV_MHD
       >> u_did:  0
       >> u_usable:       ON
       >> u_updated:      OFF
       >> u_inhibited:    OFF
       >> u_bypass:       OFF
       >> u_manrqst:      OFF
       >> u_boot: OFF
       >> u_rexinh:       OFF
       >> u_errlog:       OFF
       >> u_stat: S_OOS
       >> u_util: U_ATP
##
## In the next field listed, we see the u_admin set to A_RSV. This is the
## data that we would like to change.
##
       >> u_admin:        A_RSV
       >> u_addr
              >> fill1_addr:     0
              >> device_addr:    DEV1
              >> channel_addr:   CHAN11
       >> u_equip:        14
       >> u_path[0]:      p
       >> u_path[1]:      u
       >> u_path[2]:      /
       >> u_path[3]:      m
       >> u_path[4]:      h
       >> u_path[5]:      d
```

```
... (The rest of the ucb truncated to save space.)
```

In ecd.h, we find the following defines:

```
# define A_RSV   (0)
# define A_UNRSV        (1)
```

And from this partial hexoff (see Chapter 10) of a *ucb* we see that:

```
19      u_stat      :8            Uint
1a      u_util      :8            Uint
1b      u_admin     :8            Uint
1c      u_addr                    struct
```

The u_admin byte starts at address 0x1b from the beginning of the record. We should find a 0x00 at 0x1b off the beginning of our record. We will want to use the **browse** "patch" command to change the 0x00 to a 0x01 to unreserve the *ucb*.

```
0x7248/4X
>> 0x7248: 0x0     0x0     0x4d48  0x44000000
```

Above we have taken the "user info:" address from the dump of the record header and dumped the data there as 4 hex longs. The offset of the u_admin bit is at 0x1b from the beginning of the record so the data we want is at (0x7248 + 0x1b) = 0x7263. We need to change the word at 0x7260 from 0x80040300 to 0x80040301. Type a <CR> to dump the next 4 hex words.

```
<CR>
>> 0x7258: 0x1     0x52064700      0x80040300      0x2e5
0x7260
>> 0x7260: 0x80040300      0x2e5   0xe     0x70752f6d
```

The next command will overwrite the data at 0x7260. The command is in the following format:

```
{.}#/<format> <value list>/
```

where the "." represents the current address, the "#" is the **browse** patch command, the format is going to be a hex long, and the new value will be 0x80040301.

```
.#/X 0x80040301/
>> 0x7260: 0x80040300      <-      0x80040301
```

Now dump the record again and check that the u_admin field has changed.

```
0x54d0/r
>> 0x54d0: 0x7234:
>> record type:    ucbr
>> l_ucb
      >> u_c_name[0]:    nul
      >> u_c_name[1]:    nul
      >> u_c_name[2]:    nul
      >> u_c_name[3]:    nul
```

```
        >> u_c_name[4]:    nul
        >> u_c_name[5]:    nul
        >> u_c_name[6]:    nul
        >> u_c_name[7]:    nul
        >> u_c_name[8]:    nul
        >> u_c_unit:       nul
        >> u_name[0]:      M
        >> u_name[1]:      H
        >> u_name[2]:      D
        >> u_name[3]:      nul
        >> u_name[4]:      nul
        >> u_name[5]:      nul
        >> u_name[6]:      nul
        >> u_name[7]:      nul
        >> u_name[8]:      nul
        >> u_unit: soh
        >> u_top:  OFF
        >> u_unique:       ON
        >> u_pseudo_node:  OFF
        >> u_restorable:   ON
        >> u_removable:    OFF
        >> u_rmvd: MANRMVD
        >> u_dport:        PT_DISK
        >> u_dtype:        DEV_MHD
        >> u_did:  0
        >> u_usable:       ON
        >> u_updated:      OFF
        >> u_inhibited:    OFF
        >> u_bypass:       OFF
        >> u_manrqst:      OFF
        >> u_boot: OFF
        >> u_rexinh:       OFF
        >> u_errlog:       OFF
        >> u_stat: S_OOS
        >> u_util: U_ATP
##
## The ucb is now marked unreserved.
##
        >> u_admin:        A_UNRSV
        >> u_addr
                >> fill1_addr:     0
                >> device_addr:    DEV1
                >> channel_addr:   CHAN11
        >> u_equip:        14
        >> u_path[0]:      p
        >> u_path[1]:      u
        >> u_path[2]:      /
        >> u_path[3]:      m
        >> u_path[4]:      h
        >> u_path[5]:      d
             ... (The rest of the ucb truncated to save space.)
```

Type "q" to quit **browse**.

```
q
#
```

Check Results

Attempt another restore. This time it is successful.

```
< rst:mhd 1
 PF

M 34 RST MHD 1   TASK 5 MESSAGE STARTED

M 34 RMV MHD 1 STOPPED X’5

M 35 DGN MHD 1   COMPLETED ATP MESSAGE IN PROGRESS

M 35 RST MHD 1 IN PROGRESS

M 47 RST MHD 1 COMPLETED

M 47 DGN MHD 1   ATP  MESSAGE COMPLETE
```

Error Messages

The following error messages result from improper commands and do not terminate the **browse** session. All other messages are fatal and result from internal or other errors from which there is no recovery; for example, inability to read a file after a proper and successful open.

"bad alignment"
illegal data type alignment (for example, int at odd address)

"bad byte format"
format letter following ’ not a,c,d,o,u,x

"bad expression"
illegal expression construction (for example, missing operand)

"bad format"
illegal format (attempt to set I to other than D,O,U,X; bad count field, attempt print from nonmacro letter)

"bad frame size"
framesize not a multiple of page size

"bad grouping"
(), [], <>, or {} not balanced

"bad id"
address negative

"<adrs> : bad id"
address out of bounds

"bad operand"
nonnumeric operand in expression

"bad recdisp information"
incorrect information from auxiliary process

"bad segment specification"
incorrect segment specified for incore attach

"bad string"
string does not begin with a "

"can't allocate frames"
framesize too large

"can't connect <name>"
database <name> nonexistent or no permissions to <name>

"can't establish pipe"
cannot get pipe descriptor for ! command

"can't execute"
cannot execute named auxiliary process

"can't fork dd process"
fork failed before execution of auxiliary process

"can't fork"
cannot fork before execution of process named in ! command

"can't get segment"
getseg call failed on incore segment

"can't malloc space for search"
search command too complicated

"can't open file"
cannot connect to named file with +

"can't open input pipe"
cannot get pipe descriptor for <! command

"can't open input file"
trouble getting to command file for input

"can't read control structure"
cannot connect to file as a loadf file

"can't read control structure"
cannot connect to file as a loadf file

"can't read in get_c"
trouble with auxiliary process

"can't seek in data base"
file or loadf file corrupted

"can't seek to control structure"
bad loadf file

"dd process out of sync"
nonsense messages from auxiliary process

"id out of range"
address greater than file size

"improper value list"
value list in search or patch not of form /<formats><values>/

"line too long"
command line more than 80 characters

"lost out child"
a process spawned by the ! command has disappeared

"missing value list"
no values follow formats in search or patch command

"no closing <char>"
delimiters not balanced in search or patch command

"no data dictionary process"
the *r* format requires an auxiliary process

"no data base attached"
a command requires a database for its completion

"no match"
search failed to find values

"no permission to patch"
patch command attempted before dbp command issued

"no remembered command"
!! given before !<command>

"no remembered search string"
// given before /<format> <value list>/

"non-C character constant"
a C character constant not enclosed in single quotes (’)

"read error after fork"
auxiliary process gives/sends bad initial information

"search unsuccessful"
search command failed to find values in database

"<adrs> not bucket"
structure at <adrs> not a bucket

"<adrs> not a head"
structure at <adrs> not a dml header

"<adrs> not rec head, not indirect to one"
  <adrs> not a RID

"<adrs> not a rid block header"
  structure at <adrs> not a rid block

"<adrs> not queue or stack"
  structure at <adrs> not an access method queue or stack

"<adrs> not set header"
  structure at <adrs> not a set

"shared SDP not supported"
  cannot connect with %<name>

"too many input files"
  system limit reached on open input files

"unable to get segment code"
  database not associated with segment name.

"unable to open <name> for getseg"
  database <name> nonexistent or no permissions to <name>

"unable to open output file <name>"
  > or >> command unable to open or create file

"unbalanced patch delimiter"
  delimiters on /<format> <value list>/ in patch command not balanced

"unknown character format"
  encountered byte format other than 'a,'c,'d,'o,'u,'x.

"unknown enum name"
  auxiliary process has incorrect enumeration name list

"unknown reply"
  auxiliary process sends incorrect initial information

"unsupported member type for formatted patch"
  patching with auxiliary process is restricted to the following types: *char*, *enum*, *int*, *link*, *long*, *owner*, *short*, *string*, and *unsigned*

"wrong packet type in get_c"
  bad communication from auxiliary process

"zero or negative record length"
  bad communication from auxiliary process

ibrowse Command

2

Contents

ibrowse Command

2

Overview

ibrowse is an interactive tool that examines the address spaces of processes in memory. It operates on a file containing a contiguous portion of a 3B20D/3B21D computer physical memory spectrum beginning at word 0. This kind of operation allows use of **ibrowse** as an on-line debugging aid (by using **/dev/pmem**).

Basic *ibrowse* Features

This section describes features useful in any **ibrowse** session.

Invoking *ibrowse*

To execute **ibrowse**, enter the following command:

```
ibrowse [file]
```

db Command

The **db** command informs **ibrowse** of the file containing the physical memory spectrum. For example, **db /dev/pmem** causes **ibrowse** to reference the physical memory driver for subsequent requests. Entering this command is equivalent to invoking **ibrowse** with the name of the physical memory file as its argument.

The **db** command without an argument causes **ibrowse** to name the current physical memory file.

Displaying Virtual Addresses

After a physical memory file is specified, **ibrowse** can then display virtual addresses. Initially, these addresses are interpreted within the kernel's address space. A command for changing to the address space of any process in memory is described as follows:

Display commands in **ibrowse** are of the following form.

addr/format

where *addr* is any arithmetic expression evaluating to a virtual address. *Format* consists of a concatenation of the following format descriptors:

b —	byte
c —	character
'd —	one byte decimal
d —	short decimal
D —	long decimal
'o —	one byte octal
o —	short octal
O —	long octal
s —	string (null terminated)
'x —	one byte hexadecimal
x —	short hexadecimal
X —	long hexadecimal

In addition, any format descriptor preceded by a number causes that descriptor to be used the specified number of times.

For example, transfer vectors are stored at virtual address 0x760000 in the kernel. The command:

```
0x760000/10X
```

displays the first ten entries in this segment as long hexadecimal numbers.

Segment descriptor entries (sde) reside at address 0x1a0000 in the kernel. The command:

```
0x1a0000/XX’d’ddddd’d’dXXX
```

displays the fields of the first *sde* structure in this segment.

Internally, **ibrowse** supports three concepts useful in displaying structured data.

- Current address
- Next address
- Current format.

After a display command, the current address (available as "." in address calculations) is set to the first address displayed (0x1a0000 in the example.) The next address is set to the first address following the last item displayed (0x1a0020). The current format is set to the format used in the display. Successive carriage returns after a display cause **ibrowse** to use the current format to display the information starting at the next address. Therefore, similar structures stored in consecutive memory locations are easily displayed.

Changing Virtual Address Spaces

Ibrowse initially interprets addresses as references to the kernel's address space. The command:

```
pn N
```

references the address space of process *N* for successive displays. The command:

```
pn k
```

returns to kernel space, while:

```
pn
```

reminds the user of the current address space.

Ibrowse also directly addresses physical memory. The command:

```
pn p
```

enters physical addressing.

All Real-Time Reliable (RTR) user-level processes run at the supervision level, and the *sup* and *user* commands are excluded.

To peruse core dump files with *ibrowse*, enter the command:

```
pn c
```

This command treats the currently attached file as a formatted core dump. The data, text, stack, etc., of the late process may then be accessed with virtual addresses as though the process were still in memory.

Virtual-to-Physical Address Conversion

The command:

```
vtop N
```

returns the physical address corresponding to the virtual address *N*.

Searching

ibrowse searches forward or backward for a particular sequence of values in the current address space. The search pattern consists of two parts: a format and a sequence of values. **ibrowse** uses each format letter to determine the size of the corresponding value in the value list. For example, when the following pattern is specified:

/2X2x 1 2 3 4/

ibrowse scans forward in the current address space searching for a sequence of 12 bytes containing a long 1, long 2, short 3, and short 4, respectively. The same sequence enclosed in question marks causes **ibrowse** to search backwards for the sequence. (Note that the values still appear in ascending memory locations.)

When in virtual addressing mode, **ibrowse** limits its searches to the current segment. Therefore, if the current address is x760008, a forward search examines locations x760008 to x780000 first, followed by locations x760000 to x760008. In physical addressing, the entire range of the physical memory file is examined.

For example, to find the beginning of the kernels *Kvt* data structure (see Kernal Address Space in Chapter 9), the following commands can be used:

```
pn k
vtop x140000
bdc000
x1c0000/XXXX
x1c0000:     000009b8     000009c2     000009cc     000009d6

/XX x140000 xbdc000/
x1c35cc:     00140000     00bdc000     00be4000     00000100
```

The *Kvt* moves around from release to release but is in the kernel data segment (0x1c0000). The first work of the *Kvt* is the virtual address of the Dispatcher Control Table (DCT) (0x140000) and the next word is the physical address (found to be 0xbdc000), so this search is useful for examining off-line dumps.

Symbolic Addressing

The command:

```
sym [pfile]
```

reads the symbol table of the pfile. Functions and external variables may be subsequently referenced by name. The symbol section of the pfile must be swabbed correctly; otherwise, **ibrowse** will report "symbol not found."

For example, consider checking the file manager's tasks. The information needed, located in the *tasktab* array, consists of four word entries. The following commands display the first eight entries in the table:

```
pn 4                   /* enter fmgr's address
                          space */
sym /bootfiles/fmprc   /* attach to current file
                          manager. */
                        * (it may be necessary to run
                        * 3bswab to examine the symbols
                        * on the 3b)              */
"tasktab"/8(4X=)       /* quoted string causes symbol
                          lookup */
```

To find the virtual address of the inode table, enter:

```
"inode"=X
```

Occasionally, it is useful to convert a virtual address into a symbolic name (for example, looking up a program address). **Ibrowse** provides a special format (*a*) for this translation. If, for example, the address of the *inode* array previously mentioned was x2e0000, the command:

```
x2e0010=a
```

prints the string "inode+16" (the offset is always decimal).

Internal Buffering

To avoid excessive reads of the current file, **ibrowse** maintains a cache of recently accessed memory areas. To adjust the size of pages used for this memory management (initially 512 bytes), use the command:

```
framesize n
```

To disable buffering completely, use the command:

```
nobuf
```

The **nobuf** command is useful when examining volatile locations on a running machine. To restore buffering to its initial 512-byte frame size, use the command:

```
buf
```

Patching

ibrowse overwrites contiguous memory locations with a set of values. At present, however, the physical memory driver (**/dev/pmem**) does not support writing, so the feature may be unusable on a running 3B20D/3B21D computer. If the driver is changed to allow writes, or if there is reason to modify an off-line dump, patching is straightforward.

The core file must be reattached with permission to patch by entering the command:

```
dbp [file]
```

Modifications then take the form:

```
addr#"format value1 value2 ..."
```

The supplied values are written in memory beginning at the addressed location. As with searches, the format determines the size of each supplied value. As a result, 3X is equivalent to 3D or DXD, etc.; each value in the value list determines its own radix. For example, to replace three transfer vectors beginning at location 760010 with 120, 130, and 140, the patch command is:

```
x760010#"3X 120 130 140"
```

Calculations

ibrowse evaluates an arbitrary arithmetic expression to determine an address. To perform calculations, replace the backslash (/) in a display command with an equal sign (=). The command:

```
((3+5)/(2*1))*8=X
```

displays the result of the calculation in hexadecimal (0x20).

Shell Escape

When a line begins with an exclamation point (!), **ibrowse** invokes the shell for the remainder of the line.

ibrowse Escape

The **q** command terminates **ibrowse**.

Basic *ibrowse* Features

This section describes features useful in any **ibrowse** session.

Invoking *ibrowse*

To execute **ibrowse**, enter the following command:

ibrowse [file]

db Command

The **db** command informs **ibrowse** of the file containing the physical memory spectrum. For example, **db /dev/pmem** causes **ibrowse** to reference the physical memory driver for subsequent requests. Entering this command is equivalent to invoking **ibrowse** with the name of the physical memory file as its argument.

The **db** command without an argument causes **ibrowse** to name the current physical memory file.

Displaying Virtual Addresses

After a physical memory file is specified, **ibrowse** can then display virtual addresses. Initially, these addresses are interpreted within the kernel’s address space. A command for changing to the address space of any process in memory is described as follows:

Display commands in **ibrowse** are of the following form.

addr/format

where *addr* is any arithmetic expression evaluating to a virtual address. *Format* consists of a concatenation of the following format descriptors:

b —

byte

c —

character

’d —

one byte decimal

d —

short decimal

D —

long decimal

’o —

one byte octal

o —

short octal

O —

long octal

s —

string (null terminated)

’x —

one byte hexadecimal

x —

short hexadecimal

X —

long hexadecimal

In addition, any format descriptor preceded by a number causes that descriptor to be used the specified number of times.

For example, transfer vectors are stored at virtual address 0x760000 in the kernel. The command:

0x760000/10X

displays the first ten entries in this segment as long hexadecimal numbers.

Segment descriptor entries (sde) reside at address 0x1a0000 in the kernel. The command:

0x1a0000/XX'd'ddddd'd'dXXX

displays the fields of the first *sde* structure in this segment.

Internally, **ibrowse** supports three concepts useful in displaying structured data.

- Current address
- Next address
- Current format.

After a display command, the current address (available as "." in address calculations) is set to the first address displayed (0x1a0000 in the example.) The next address is set to the first address following the last item displayed (0x1a0020). The current format is set to the format used in the display. Successive carriage returns after a display cause **ibrowse** to use the current format to display the information starting at the next address. Therefore, similar structures stored in consecutive memory locations are easily displayed.

Changing Virtual Address Spaces

Ibrowse initially interprets addresses as references to the kernel's address space. The command:

pn N

references the address space of process *N* for successive displays. The command:

pn k

returns to kernel space, while:

pn

reminds the user of the current address space.

Ibrowse also directly addresses physical memory. The command:

pn p

enters physical addressing.

All Real-Time Reliable (RTR) user-level processes run at the supervision level, and the *sup* and *user* commands are excluded.

To peruse core dump files with *ibrowse*, enter the command:

pn c

This command treats the currently attached file as a formatted core dump. The data, text, stack, etc., of the late process may then be accessed with virtual addresses as though the process were still in memory.

Virtual-to-Physical Address Conversion

The command:

vtop N

returns the physical address corresponding to the virtual address *N*.

Searching

ibrowse searches forward or backward for a particular sequence of values in the current address space. The search pattern consists of two parts: a format and a sequence of values. **ibrowse** uses each format letter to determine the size of the corresponding value in the value list. For example, when the following pattern is specified:

/2X2x 1 2 3 4/

ibrowse scans forward in the current address space searching for a sequence of 12 bytes containing a long 1, long 2, short 3, and short 4, respectively. The same sequence enclosed in question marks causes **ibrowse** to search backwards for the sequence. (Note that the values still appear in ascending memory locations.)

When in virtual addressing mode, **ibrowse** limits its searches to the current segment. Therefore, if the current address is x760008, a forward search examines locations x760008 to x780000 first, followed by locations x760000 to x760008. In physical addressing, the entire range of the physical memory file is examined.

For example, to find the beginning of the kernels *Kvt* data structure (see Kernal Address Space in Chapter 9), the following commands can be used:

```
pn k
vtop x140000
bdc000
x1c0000/XXXX
x1c0000:     000009b8     000009c2     000009cc     000009d6

/XX x140000 xbdc000/
x1c35cc:     00140000     00bdc000     00be4000     00000100
```

The *Kvt* moves around from release to release but is in the kernel data segment (0x1c0000). The first work of the *Kvt* is the virtual address of the Dispatcher Control Table (DCT) (0x140000) and the next word is the physical address (found to be 0xbdc000), so this search is useful for examining off-line dumps.

Symbolic Addressing

The command:

```
sym [pfile]
```

reads the symbol table of the pfile. Functions and external variables may be subsequently referenced by name. The symbol section of the pfile must be swabbed correctly; otherwise, **ibrowse** will report "symbol not found."

For example, consider checking the file manager’s tasks. The information needed, located in the *tasktab* array, consists of four word entries. The following commands display the first eight entries in the table:

```
pn 4                    /* enter fmgr’s address
                           space */
sym /bootfiles/fmprc    /* attach to current file
                           manager. */
                         * (it may be necessary to run
                         * 3bswab to examine the symbols
                         * on the 3b)              */
"tasktab"/8(4X=)        /* quoted string causes symbol
                           lookup */
```

To find the virtual address of the inode table, enter:

```
"inode"=X
```

Occasionally, it is useful to convert a virtual address into a symbolic name (for example, looking up a program address). **Ibrowse** provides a special format (*a*) for this translation. If, for example, the address of the *inode* array previously mentioned was x2e0000, the command:

```
x2e0010=a
```

prints the string "inode+16" (the offset is always decimal).

Internal Buffering

To avoid excessive reads of the current file, **ibrowse** maintains a cache of recently accessed memory areas. To adjust the size of pages used for this memory management (initially 512 bytes), use the command:

```
framesize n
```

To disable buffering completely, use the command:

nobuf

The **nobuf** command is useful when examining volatile locations on a running machine. To restore buffering to its initial 512-byte frame size, use the command:

buf

Patching

ibrowse overwrites contiguous memory locations with a set of values. At present, however, the physical memory driver (**/dev/pmem**) does not support writing, so the feature may be unusable on a running 3B20D/3B21D computer. If the driver is changed to allow writes, or if there is reason to modify an off-line dump, patching is straightforward.

The core file must be reattached with permission to patch by entering the command:

dbp [file]

Modifications then take the form:

addr#"format value1 value2 ..."

The supplied values are written in memory beginning at the addressed location. As with searches, the format determines the size of each supplied value. As a result, 3X is equivalent to 3D or DXD, etc.; each value in the value list determines its own radix. For example, to replace three transfer vectors beginning at location 760010 with 120, 130, and 140, the patch command is:

x760010#"3X 120 130 140"

Calculations

ibrowse evaluates an arbitrary arithmetic expression to determine an address. To perform calculations, replace the backslash (/) in a display command with an equal sign (=). The command:

((3+5)/(2*1))*8=X

displays the result of the calculation in hexadecimal (0x20).

Shell Escape

When a line begins with an exclamation point (!), **ibrowse** invokes the shell for the remainder of the line.

ibrowse Escape

The **q** command terminates **ibrowse**.

ibrowse Command

NAME

ibrowse - examine a memory-resident *UNIX*® Real-Time Reliable (RTR) operating system process.

FORMAT

ibrowse *file*

DESCRIPTION

Ibrowse is an interactive tool that examines files containing a dump of *UNIX* RTR operating system physical memory. On a 3B20D/3B21D computer, this file is usually */dev/pmem* for the currently running processes, or */dev/ofln* for the contents of the off-line memory. Locations in the address space of any process in memory can be displayed. **ibrowse** also provides facilities for examining core dump files, as well as primitives to display ordinary unstructured files.

Commands

buf	Turn on internal memory management (default).
db *file*	Examine the contents of *file*.
null *n*	Set value for termination of indirect formats (initially 0).

* UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

pn *n*	Treat subsequent addresses from the perspective of process *n*.
pn k	Switch to kernel's address space.
pn p	Use physical addressing - no virtual address translation is performed. This mode is also used for examining unstructured files.
pn c	Treat currently attached file as a core dump file.
pn	Display current process.
sup	Switch to supervisor address space.
sym *pfile*	Use the symbols from *pfile* to allow symbolic addressing.
user	Switch to user address space.
vtop *n*	Convert virtual address to physical.

Displaying Values

Display commands in **ibrowse** are of the following form:

address/format

Address can be a number, arithmetic expression, search string, or variable name enclosed in quotes; for example, *"buffer"+30 is a legal address.*

If the / is repaced by =, the address is printed, rather than its contents. This allows use of **ibrowse** as a calculator.

Formats

A format consists of a concatenation of the following symbols:

a	Name of variable at address.
b	Byte.
c	Character.

’d, d, D Decimal of 1, 2, and 4 bytes, respectively.

’o. o. O Octal of 1, 2, and 4 bytes, respectively.

s Null terminated string.

’u, u, U Unsigned decimal of 1, 2, and 4 bytes, respectively.

’x, x, X Hexadecimal of 1, 2, and 4 bytes, respectively.

Format Control Constructs

{format} Treat the current value as an address. Display values at this address using *format.*

[n] Skip forward *n* bytes (*n* may be negative).

[kmark] Save the current location in *mark*, which should be a lowercase letter.

[’mark] Return to the saved mark.

<format> Suppress printing values that *format* would have displayed.

Macros

The capital letters unused by **ibrowse** are available for format macros. Macro definitions are of the following form:

letter definition

The defined macro can then be treated in the same way as any of the predefined format symbols.

Miscellaneous Commands

<file Take command input from *file.*

>file Redirect subsequent output of *file.*
If the argument is missing, output returns to its previous destination.

>>file Append subsequent output to *file.*

cx Command

3

Contents

cx Command

3

Overview

To print the segment images of a specified core file, use the **cx** command. The command has the following format:

cx [**-!**ahpsx] [**-v** vaddr [**-n** nbytes]] ... file

cx resides on the computer running Real-Time Reliable (RTR) (all platforms).

When invoked without options, **cx** prints all data from all nonexecutable segments in the given file.

A core file's format must be the same as that written by the process manager and must resemble executable pfiles. Therefore, **cx** can be used to print segment images from pfiles also. **cx** also works on shared library files.

Usually, the process manager places core files in */cdmp/name*, where *name* is the American Standard Code for Information Interchange (ASCII) name in the process control block (PCB) (pcb->p_name).

In actual core files, the segment images occur in the same order as they appeared in the process segment list. The PCB is first, and the stack is second. Those segment images in pfiles are empty.

More than one segment can be mapped to the same virtual address, but only one of them can be active. **-v** directives apply to all mapped segments, not just the active one.

cx is not interactive, it does not read symbol tables, and it does not allow you to define output formats.

The *cx* Options

The following options can be specified on the **cx** commands:

-! Print descriptions of each flag's meaning. Illegal command lines generate a terse summary of usage without the descriptions.

-a Print data for all nonexecutable segment images. This is the default action when no **-v, -h**, or **-x** flags are present.

-h Print a header for every segment. This flag also suppresses segments data printing. Use the **-a, -v**, or **-x** option to print the designated data.

-p Print *pfile* meaning of header attribute flags. Requests the *pfile* meanings which are different in a small number of cases. Each segment's header contains a set of attribute flags. **cx** normally decodes the flags using their *execution* time meanings.

-s Print segment header information and partial formatting of the stack segment.

-x Print data for all executable segments. (Without **-x**, the data are suppressed.)

-v *vaddr* Print data from the given address in the segment(s) containing virtual address *vaddr*. Unless **-n** is also given, the rest of the segment is printed. Using one or more **-v** specifications causes other segment images not to be printed. To override the **-v** specification, use **-a** or **-x** option.

-n *nbytes* Print *nbytes* bytes instead of printing all of a segment's data. A count of zero means the remainder of the image. This option applies only to the preceding **-v**, which must be present.

Values for *vaddr* and *nbytes* are interpreted as hexadecimal numbers. *cx* does not require **0x** or **0X** prefixes, but it accepts and ignores them.

Reading a User Process Core Dump

First, a word about how a core dump is created. Something goes awry in your program causing an exception. This exception is handled by the operating system and results in your process being faulted. Assuming this is a normal user process, the fault entry is contained in *libc*. The fault entry will take care of closing any open files, freeing any pending messages, and finally requesting process termination with a core image. This code is running on behalf of your process in your process. In other words, your process continues to run even after it has committed a fatal error.

What this means to you is that there is more involved in determining the address of the exception than simply examining the program address at termination. A stack-back trace to find the program address prior to the termination processing will be necessary. Fortunately, the core-examiner with the **-s** option will do most of the work most of the time. See Figure 3-1.

```
$ cd /cdmp
$ cx -s 208.cftshl
208.cftshl:        supervisor or unix core file
Segment 0:         0x1000 bytes, 0x1000 in file, index 0x30(48), vaddr 0x600000
          flags:   0x904 nn sbit rd

Segment 1:         0x1800 bytes, 0x1800 in file, index 0x35(53), vaddr 0x6a0000
          flags:   0x91e nn sbit pwrt stk rd wrt

   Argument  1:        0x1
   Argument  2:        0x6a0058
   Argument  3:        0x6a0060
   Program Address:    0xffffffff
   Argument Pointer:   0x0
   Frame Pointer:      0x0
   Unused:             0x0
   Register  0:        0x64e694
   Register  1:        0x6a01c8
   Register  2:        0x6a01b0
   Register  3:        0x6a0040
   Register  4:        0x6a004c
   Register  5:        0x2c
   Register  6:        0x6a005c
   Register  7:        0x622449
   Register  8:        0x2d6

   Argument  1:        0x1
   Argument  2:        0x6a0058
   Argument  3:        0x6a0060
   Program Address:    0x2c
   Argument Pointer:   0x6a0078
   Frame Pointer:      0x6a00b8
   Unused:             0x1
   Register  0:        0x6a0058
   Register  1:        0x6a0060
   Register  2:        0x0
   Register  3:        0x6486c0
   Register  4:        0x6a0094
   Register  5:        0x6a0051
```

Figure 3-1. Example Stack Trace Option Output (Sheet 1 of 4)

```
Segment 1 (Contd):  0x1800 bytes, 0x1800 in file, index 0x35(53), vaddr 0x6a0000
                    flags:         0x91e nn sbit pwrt stk rd wrt

  Program Address:   0x4a8
  Argument Pointer:  0x6a00b8
  Frame Pointer:     0x6a00f8
  Unused:            0x6a00b8
  Register  6:       0x6a0051
  Register  7:       0x620051
  Register  8:       0x6a0051
  Register  0:       0x6a00f8
  Register  1:       0x6a00f8
  Register  2:       0x622448
  Register  3:       0x622449
  Register  4:       0x32
  Register  5:       0x32
  Register  6:       0x32
  Register  7:       0x6a0078
  Register  8:       0x0

  Argument  1:       0x40000
  Program Address:   0xa70
  Argument Pointer:  0x6a02d4
  Frame Pointer:     0x6a0308
  Unused:            0x6a0308
  Register  0:       0x6a02d4
  Register  1:       0x6a0308
  Register  2:       0x1
  Register  3:       0x32
  Register  4:       0x32
  Register  5:       0x32
  Register  6:       0x6a0078
  Register  7:       0x4b1
  Register  8:       0x4b8

  Argument  1:       0x0
  Argument  2:       0x40000
  Argument  3:       0x0
  Program Address:   0xf41e
  Argument Pointer:  0x6a03fc
  Frame Pointer:     0x6a0434
  Unused:            0x6a0440
  Register  0:       0x622449
  Register  1:       0xd
  Register  2:       0x1
  Register  3:       0xe
  Register  4:       0x32
  Register  5:       0x6a0078
  Register  6:       0x0
  Register  7:       0x40000
  Register  8:       0x4b8
```

Figure 3-2. Example Stack Trace Option Output (Sheet 2 of 4)

```
Segment 1 (Contd):        0x1800 bytes, 0x1800 in file, index 0x35(53), vaddr 0x6a0000
                          flags:          0x91e nn sbit pwrt stk rd wrt

    Argument  1:          0x40000
    Program Address:      0xd48c
    Argument Pointer:     0x6a0438
    Frame Pointer:        0x6a0478
    Unused:               0x6a0478
    Register  0:          0x6a0478
    Register  1:          0x0
    Register  2:          0x0
    Register  3:          0x0
    Register  4:          0x0
    Register  5:          0x0
    Register  6:          0x0
    Register  7:          0x40000
    Register  8:          0x0

    Argument  1:          0x660ec4
    Program Address:      0x6459d6
    Argument Pointer:     0x6a0578
    Frame Pointer:        0x6a05b0
    Unused:               0x6425f8
    Register  0:          0x6a0578
    Register  1:          0x6a05b0
    Register  2:          0x294ed
    Register  3:          0x4134
    Register  4:          0x0
    Register  5:          0x32
    Register  6:          0x6a0078
    Register  7:          0x32
    Register  8:          0x0

    .                     .
    .                     .
    .                     .

    Argument  1:          0x6a083c
    Argument  2:          0xd1
    Program Address:      0x64e606
    Argument Pointer:     0x6a0800
    Frame Pointer:        0x6a0834
    Unused:               0x6a0754
    Register  0:          0x6a078c
    Register  1:          0x6a078c
    Register  2:          0x0
    Register  3:          0x1b5c
    Register  4:          0x1
    Register  5:          0xdfc00
    Register  6:          0x660a8c
    Register  7:          0x661f5c
    Register  8:          0x661394

    .                     .
    .                     .
    .                     .
```

Figure 3-3. Example Stack Trace Option Output (Sheet 3 of 4)

```
Segment 2:    0x10800 bytes, 0x10800 in file, index 0x0(0), vaddr 0x0
      flags:  0x935 nn sbit share pwrt rd exec

Segment 3:    0xb6f0 bytes, 0xb6f0 in file, index 0x1(1), vaddr 0x20000
      flags:  0x916 nn sbit pwrt rd wrt

Segment 4:    0xf400 bytes, 0xf400 in file, index 0x32(50), vaddr 0x640000
      flags:  0x935 nn sbit share pwrt rd exec

Segment 5:    0x1d00 bytes, 0x1d00 in file, index 0x33(51), vaddr 0x660000
      flags:  0x916 nn sbit pwrt rd wrt

Segment 6:    0x200 bytes, 0x200 in file, index 0x34(52), vaddr 0x680000
      flags:  0x937 nn sbit share pwrt rd wrt exec

Segment 7:    0x8940 bytes, 0x8940 in file, index 0x22(34), vaddr 0x440000
      flags:  0x925 nn sbit share rd exec

Segment 8:    0x1640 bytes, 0x1640 in file, index 0x23(35), vaddr 0x460000
      flags:  0x916 nn sbit pwrt rd wrt

Segment 9:    0x200 bytes, 0x200 in file, index 0x24(36), vaddr 0x480000
      flags:  0x937 nn sbit share pwrt rd wrt exec

Segment 10:   0x114c0 bytes, 0x114c0 in file, index 0x28(40), vaddr 0x500000
      flags:  0x935 nn sbit share pwrt rd exec

Segment 11:   0x1040 bytes, 0x1040 in file, index 0x29(41), vaddr 0x520000
      flags:  0x916 nn sbit pwrt rd wrt

Segment 12:   0x200 bytes, 0x200 in file, index 0x2a(42), vaddr 0x540000
      flags:  0x937 nn sbit share pwrt rd wrt exec

Segment 13:   0x1b1c bytes, 0x1b1c in file, index 0x2d(45), vaddr 0x5a0000
      flags:  0x934 nn sbit share pwrt rd

Segment 14:   0x1880 bytes, 0x1880 in file, index 0x2c(44), vaddr 0x580000
      flags:  0x936 nn sbit share pwrt rd wrt

Segment 15:   0x20000 bytes, 0x20000 in file, index 0x1e(30), vaddr 0x3c0000
      flags:  0x934 nn sbit share pwrt rd

Segment 16:   0x20000 bytes, 0x20000 in file, index 0x1f(31), vaddr 0x3e0000
      flags:  0x934 nn sbit share pwrt rd

Segment 17:   0x9cc bytes, 0x9cc in file, index 0x20(32), vaddr 0x400000
      flags:  0x934 nn sbit share pwrt rd

Segment 18:   0x800 bytes, 0x800 in file, index 0x2(2), vaddr 0x40000
      flags:  0x936 nn sbit share pwrt rd wrt
```

Figure 3-4. Example Stack Trace Option Output (Sheet 4 of 4)

However, in some cases the stack itself might be corrupted and the **-s** option will not work or will be incomplete. In those cases, the manual stack-back trace method will be needed to extract useful information from the stack. See Figure 3-2.

1. Obtain **cx** listing of the core image.
2. Locate the process control block for your process in the listing (generally this will be segment 0, regardless the virtual address is 0x600000) (<R1>, <R6>, and <R21>).
3. Double check to be sure that this is the correct process. In <R1>, the first word is the process id. The second word is the parent's process id. In <R6> and <R21> the process id is at offset 0x82c and the next word is process id of the parent. The process name begins at the next word (offset 0x834) and up to 16 characters.

```
#cx -v 600820 -n 20 208.cftshl
208.cftshl:          supervisor or unix core file
Segment 0:           0x1000 bytes, 0x1000 in file, index 0x30(48), vaddr 0x600000
                     flags:        0x904 nn sbit rd

00600820:   00000000   00000000   00000000   000a0027    .... .... .... ...’
00600830:   00000009   3230382e   63667473   686c0000    .... 208. cfts hl..
```

Figure 3-5. Release 21 Example

4. Look at 0x600095 <**<R1>**> 0x60084a <**<R6> and <R21>**> for one byte *. This is the fault code. Sometimes this is sufficient to determine what went wrong *. Always check this before proceeding further.
5. Before beginning the stack-back trace, refer to Step 7 and Figure 3-5 for the layout of a stack frame. The *arg* variables are the arguments that are passed on a function call. The ‘‘old’’ variables and save area are placed on the stack as a result of the function call itself. The *auto* variables are the local arguments to the called function.

   NOTE:
   The first five register variables declared in a C function are not placed on the stack, but rather stored in registers 8 through 4, respectively.

   When register variables are declared, the old register variables need to be saved on the stack. If for example, function *a()*, which has r register variables, calls function *b()*, which has 2 register variables, only 2 old registers will be saved on the stack (namely, registers 8 and 7). If function *b()* then calls function *c()*, which has 5 register variables, 5 old register variables will be saved on the stack [registers 8 and 7 from functions *b()*, and register 6, 5, and 4 from function *a()*].

NOTE:
If a function is called, which has no register variables, no registers are saved on the stack.

The basic strategy of a stack-back trace will be to:

a. Use cx -h file name to display the segment headers. Find the stack segment, that is:

```
Segment 1:          0x1800 bytes, 0x1800 in file, index 0x35(53), vaddr 0x6a0000
        flags:      0x91e nn sbit pwrt stk rd wrt
```

The ‘‘stk’’ flag indicates it is a stack segment.

b. Dump the stack using the vaddr and number of bytes obtained from the segment header (see the following example).

```
$ cx -v 6a0000 -n 1800 filename
cx -v 6a0000 -n 1800 208*
208.cftshl: supervisor or unix core file
Segment 1:          0x1800 bytes, 0x1800 in file, index 0x35(53), vaddr 0x6a0000
        flags:      0x91e nn sbit pwrt stk rd wrt
```

```
006a0000:  2f636674  2f62696e  2f636674  73686c00   /cft /bin /cft shl.
006a0010:  545a3d50  53543850  44540050  4154483d   TZ=P ST8P DT.P ATH=
006a0020:  3a2f6269  6e3a2f75  73722f62  696e004c   :/bi n:/u sr/b in.L
006a0030:  4348414e  3d747479  61004e49  43455641   CHAN =tty a.NI CEVA
006a0040:  4c3d3030  0053484c  50524d50  543d053c   L=00 .SHL PRMP T=.<
006a0050:  2000001b  00000000  006a0000  00000000    ... .... .j.. ....
006a0060:  006a0010  006a001b  006a002f  006a003a   .j.. .j.. .j./ .j.:
006a0070:  006a0045  00000000  00000001  006a0058   .j.E .... .... .j.X
006a0080:  006a0060  ffffffff  00000000  00000000   .j.‘ .... .... ....
006a0090:  00000000  0064e694  006a01c8  006a01b0   .... .d.. .j.. .j..
006a00a0:  006a0040  006a004c  0000002c  006a005c   .j.@ .j.L ..., .j.
006a00b0:  00622449  000002d6  00000001  006a0058   .b$I .... .... .j.X
006a00c0:  006a0060  0000002c  006a0078  006a00b8   .j.‘ ..., .j.x .j..
006a00d0:  00000001  006a0058  006a0060  00000000   .... .j.X .j.‘ ....
006a00e0:  006486c0  006a0094  006a0051  006a0051   .d.. .j.. .j.Q .j.Q
006a00f0:  00620051  006a0051  00800000  00000440   .b.Q .j.Q .... ...@
006a0100:  5c6e0000  00000000  00000000  00000000   0. .... .... ....
006a0110:  00000000  00000000  00000000  00000000   .... .... .... ....
*
006a0140:  00000000  00000000  00000000  5c202020   .... .... ....
006a0150:  20202564  20202574  20232567  5c6e0000     %d   %t  #%g 0.
006a0160:  00000000  00000000  00000000  00000000   .... .... .... ....
*
006a0190:  00000000  00000000  4f524947  494e4154   .... .... ORIG INAT
006a01a0:  494e4720  434f4d4d  414e4420  23203d20   ING  COMM AND  # =
006a01b0:  256e2e25  635c6e00  00000000  00000000   %n.% c0 .... ....
006a01c0:  00000000  00000000  00000000  00000000   .... .... .... ....
*
006a01e0:  00000000  74747961  00000000  002f6465   .... ttya .... ./de
006a01f0:  762f7474  79610000  00000000  00000000   v/tt ya.. .... ....
```

```
006a01e0:  00000000  74747961  00000000  002f6465   .... ttya .... ./de
006a01f0:  762f7474  79610000  00000000  00000000   v/tt ya.. .... ....
006a0200:  00000000  00000000  00000000  00000000   .... .... .... ....
*
006a0220:  00000000  00001900  00000000  00000000   .... .... .... ....
006a0230:  00000000  00008600  00320200  00500000   .... .... .2.. .P..
006a0240:  00000000  00000006  00000002  6e000000   .... .... .... n...
006a0250:  00000000  00000000  00660244  00647f4a   .... .... .f.D .d.J
006a0260:  0901020c  006a0258  00000008  006a024c   .... .j.X .... .j.L
006a0270:  006a024c  006a024c  006a0254  80054880   .j.L .j.L .j.T ..H.
006a0280:  00000001  00622448  006217e0  006217b0   .... .b$H .b.. .b..
006a0290:  00000030  00000000  00000000  006a0258   ...0 .... .... .j.X
006a02a0:  006a0290  00001880  00000000  008e1c2c   .j.. .... .... ...,
006a02b0:  0064eace  006a0258  006a029c  006a0280   .d.. .j.X .j.. .j..
006a02c0:  0064ece8  80054880  00660244  00660484   .d.. ..H. .f.D .f..
006a02d0:  0000000e  000004a8  006a00b8  006a00f8   .... .... .j.. .j..
006a02e0:  006a00b8  006a00f8  006a00f8  00622448   .j.. .j.. .j.. .b$H
006a02f0:  00622449  00000032  00000032  00000032   .b$I ...2 ...2 ...2
006a0300:  006a0078  00000000  00000002  00000001   .j.x .... .... ....
006a0310:  00000001  00010002  006a0310  006a0310   .... .... .j.. .j..
006a0320:  00000000  6100eda9  00000000  000e0000   .... a... .... ....
006a0330:  00001b1c  00da0824  240d0aff  ff5f0820   .... ...$ $... ._.
006a0340:  08ffff0d  3fffffff  ff64003f  0722085c   .... ?... .d.? .".
006a0350:  21ffffff  ffff2600  185001d6  00000ec5   !... ..&. .P.. ....
006a0360:  006a030c  006a0340  01010340  006a0360   .j.. .j.@ ...@ .j.‘
006a0370:  00000008  006a0308  006a0308  00000032   .... .j.. .j.. ...2
006a0380:  00000000  00000126  00000012  00000012   .... ...& .... ....
006a0390:  00000000  0064edb0  006a0354  00000000   .... .d.. .j.T ....
006a03a0:  00000000  00000000  fc008db8  0064eace   .... .... .... .d..
006a03b0:  006a0354  006a0398  000004b1  000004b8   .j.T .j.. .... ....
006a03c0:  00000000  00000000  00000001  00660484   .... .... .... .f..
006a03d0:  00660484  0064927c  006a0394  006a03d0   .f.. .d.
006a03e0:  00000000  008d9220  00000124  00064c10   .... ...  ...$ ..L.

006a03f0:  00046674  000406c9  81ed0000  00040000   ..ft .... .... ....
006a0400:  00000a70  006a02d4  006a0308  006a0308   ...p .j.. .j.. .j..
006a0410:  006a02d4  006a0308  00000001  00000032   .j.. .j.. .... ...2
006a0420:  00000032  00000032  006a0078  000004b1   ...2 ...2 .j.x ....
006a0430:  000004b8  000004b8  00000000  00040000   .... .... .... ....
006a0440:  00000000  0000f41e  006a03fc  006a0434   .... .... .j.. .j.4
006a0450:  006a0440  00622449  0000000d  00000001   .j.@ .b$I .... ....
006a0460:  0000000e  00000032  006a0078  00000000   .... ...2 .j.x ....
006a0470:  00040000  000004b8  0064de10  006a0440   .... .... .d.. .j.@
006a0480:  006a0474  00027b48  00027b48  00643ce8   .j.t ..{H ..{H .d<.
006a0490:  006a0440  00622449  00000032  006a0078   .j.@ .b$I ...2 .j.x
006a04a0:  00000000  000004b1  006a03fc  00000001   .... .... .j.. ....
006a04b0:  00000000  00000000  00660484  0064df70   .... .... .f.. .d.p
006a04c0:  006a0474  006a04ac  006a0474  006a04ac   .j.t .j.. .j.t .j..
006a04d0:  006a03fc  006a0434  006a04c0  00000032   .j.. .j.4 .j.. ...2
006a04e0:  006a0078  00000000  000004b1  000004b8   .j.x .... .... ....
006a04f0:  00027a38  00000000  000004b1  000004b8   ..z8 .... .... ....

006a0500:  000042dc  006a04bc  006a04f0  0002a5c8   ..B. .j.. .j.. ....
006a0510:  0002a370  0002a384  00000001  00622449   ...p .... .... .b$I
006a0520:  00008106  006a04e4  006a0518  006a0518   .... .j.. .j.. .j..
006a0530:  00000001  0000003f  00000005  00000006   .... ...? .... ....
```

```
006a0540:   fffffffe   00000001   0002a5c8   000077a6    .... .... .... ..w.
006a0550:   006a04bc   006a0500   006a04bc   006a0500    .j.. .j.. .j.. .j..
006a0560:   006a0500   414e4420   272f6366   742f7368    .j.. AND  '/cf t/sh
006a0570:   50460000   00000000   00040000   0000d48c    PF.. .... .... ....
006a0580:   006a0438   006a0478   006a0478   006a0478    .j.8 .j.x .j.x .j.x
006a0590:   00000000   00000000   00000000   00000000    .... .... .... ....
006a05a0:   00000000   00000000   00040000   00000000    .... .... .... ....
006a05b0:   00660ec4   006459d6   006a0578   006a05b0    .f.. .dY. .j.x .j..
006a05c0:   006425f8   006a0578   006a05b0   000294ed    .d%. .j.x .j.. ....
006a05d0:   00004134   00000000   00000032   006a0078    ..A4 .... ...2 .j.x

006a05e0:   00000032   00000000   00000000   00661b5c    ...2 .... .... .f.
006a05f0:   00000001   006450a4   006a05b0   006a05e8    .... .dP. .j.. .j..
006a0600:   006a05f4   00001508   006a05b8   00622449    .j.. .... .j.. .b$I
006a0610:   00000032   00000032   006a0078   00040000    ...2 ...2 .j.x ....
006a0620:   00040000   00660ec4   006494b0   006a05e8    .... .f.. .d.. .j..
006a0630:   006a0628   006a0628   00000002   0064bf0c    .j.( .j.( .... .d..
006a0640:   006a05f4   006a0628   00000032   006a0078    .j.. .j.( ...2 .j.x
006a0650:   00040000   00040000   00000040   0066080c    .... .... ...@ .f..
006a0660:   0064964c   006a0628   006a065c   006a05b8    .d.L .j.( .j. .j..
006a0670:   00000009   00052b60   00000000   00622449    .... ..
006a0680:   00000032   006a0078   00000001   0066080c    ...2 .j.x .... .f..
006a0690:   006a05e8   00000000   00000000   00000000    .j.. .... .... ....
006a06a0:   00000000   00000000   00661394   0066080c    .... .... .f.. .f..
006a06b0:   00649a9a   006a065c   006a0694   00000009    .d.. .j. .j.. ....
006a06c0:   006a0670   006a0628   0064eba4   00001b5c    .j.p .j.( .d.. ...
006a06d0:   00000005   006a0078   00000001   00661394    .... .j.x .... .f..
006a06e0:   0066080c   00660484   00646a96   006a06a8    .f.. .f.. .dj. .j..
006a06f0:   006a06e4   00001488   00000002   00000040    .j.. .... .... ...@
006a0700:   00027140   00027145   060a4d20   20202052    ..q@ ..qE ..M     R
006a0710:   45505420   00661394   00661394   4e544552    EPT  .f.. .f.. NTER
006a0720:   0064eee4   80054888   ff3da001   ff3da001    .d.. ..H. .=.. .=..
006a0730:   00000001   00649498   00660004   00001b5c    .... .d.. .f.. ...
006a0740:   00660484   006a0078   00000001   00000000    .f.. .j.x .... ....
006a0750:   00661394   006a06e4   006a071c   006a0720    .f.. .j.. .j.. .j.
006a0760:   006a06f0   006a06e4   0064e000   006a071c    .j.. .j.. .d.. .j..
006a0770:   006a0720   20000000   0064e5f6   006a06e4    .j.   ... .d.. .j..
006a0780:   006a071c   00000002   00027b58   003d5b24    .j.. .... ..{X .=[$
006a0790:   7effffff   006a06b0   00660484   006a0078     ... .j.. .f.. .j.x
006a07a0:   00000001   00000000   00661394   00000000    .... .... .f.. ....
006a07b0:   00000001   006a0078   00000100   20000000    .... .j.x ....  ...

006a07c0:   006a06b0   006a0760   7effffff   00660644    .j.. .j.‘  ... .f.D
006a07d0:   0064b00e   006a0790   006a07c4   20000000    .d.. .j.. .j..  ...
006a07e0:   006a06d0   00000000   00000001   006a0078    .j.. .... .... .j.x
006a07f0:   00000100   20000000   006a06e8   006a06e8    ....  ... .j.. .j..
006a0800:   0064ad6a   006a0770   006a07ac   006a07ac    .d.j .j.p .j.. .j..
006a0810:   006a07fc   006a07d0   00002b82   00001b5c    .j.. .j.. ..
006a0820:   00000001   006a0078   00000001   20000000    .... .j.x ....  ...
006a0830:   006a0720   00660644   006a0674   0064b104    .j.  .f.D .j.t .d..
006a0840:   80054880   ff3da001   ff3da001   0066068c    ..H. .=.. .=.. .f..
006a0850:   00649498   00660004   00001b5c   00000001    .d.. .f.. ...
006a0860:   000dfc00   00660a8c   00661f5c   00661394    .... .f.. .f. .f..
006a0870:   006a0800   006a0834   006a083c   006a0848    .j.. .j.4 .j.< .j.H
006a0880:   00037ed6   00660b34   ac000000   006a083c    .. . .f.4 .... .j.<
006a0890:   000000d1   0064e606   006a0800   006a0834    .... .d.. .j.. .j.4
```

```
006a08a0:   006a0754   006a078c   006a078c   00000000    .j.T .j.. .j.. ....
006a08b0:   00001b5c   00000001   000dfc00   00660a8c    ... .... .... .f..
006a08c0:   00661f5c   00661394   00027b58   005a0a2e    .f. .f.. ..{X .Z..
006a08d0:   005a0a40   00000016   ffffffff   005a03b6    .Z.@ .... .... .Z..
006a08e0:   006a0674   005a0a3f   00000001   00501260    .j.t .Z.? .... .P.‘
006a08f0:   006a0840   006a0880   00000000   00000000    .j.@ .j.. .... ....

006a0900:   00000000   00000000   00000000   00000000    .... .... .... ....
006a0910:   00000000   0000000d   00000000   0000003f    .... .... .... ...?
006a0920:   0000008b   0064b4b0   006a088c   006a08c8    .
006a0930:   0064b494   006a088c   006a08c8   00000000    .d.. .j.. .j.. ....
006a0940:   00000000   00000001   00000001   0000000b    .... .... .... ....
006a0950:   ffffffd1   006a083c   00010000   00000048    .... .j.< .... ...H
006a0960:   00000009   0064bc62   006a0920   006a0958    .... .d.b .j.  .j.X
006a0970:   006479d0   006a0920   006a0958   0064796c    .dy. .j.  .j.X .dyl
006a0980:   0000008b   ffffffd1   006a083c   00000000    .... .... .j.< ....
006a0990:   00660484   00660484   00660484   0064941c    .f.. .f.. .f.. .d..
006a09a0:   006a0958   006a0998   006a0770   006a07ac    .j.X .j.. .j.p .j..
006a09b0:   010102a5   006a0a1c   00000040   00000000    .... .j.. ...@ ....
006a09c0:   00000000   00000000   00000000   00000000    .... .... .... ....
006a09d0:   3230382e   63667473   686c0000   00000000    208. cfts hl.. ....
006a09e0:   00000000   00000000   00000000   20000000    .... .... ....  ...
006a09f0:   0064b104   00000000   00000000   00000000    .d.. .... .... ....

continues to 6a1800
```

c. Display the savestate of the process from the PCB. The hexoff structure for the PCB can be found in the appropriate per release chapter in this information product (IP). For this <R21> example, the address is 0x6008f0.

```
cx -v 6008f0 -n 50 208*
208.cftshl: supervisor or unix core file
Segment 0:             0x1000 bytes, 0x1000 in file, index 0x30(48), vaddr 0x600000
          flags:       0x904 nn sbit rd

006008f0:   0064ece8   80054880   ff3da001   ff3da001    .d.. ..H. .=.. .=..
00600900:   00000001   006a0b34   00000000   00000000    .... .j.4 .... ....
00600910:   00000001   00000001   0000008b   00000001    .... .... .... ....
00600920:   00660484   006a0998   006a09d0   006a09d0    .f.. .j.. .j.. .j..
00600930:   006a09a4   006a0998   0064e000   006a09d0    .j.. .j.. .d.. .j..
```

d. To begin, locate the PA where the program was executing when the core dump was made (this is at address 0600790 **<R1>** or 0x6008f0 **<R6> and <R21>**.)

NOTE:
This will be the address at which the request was made to produce the core dump, not the address which caused the fault. Examining a namelist of **libc** should show that this address is contained in the routine *%sendcmsg*.

e. Find the current frame pointer (FP) (R10 that is 006a09d0) and the current argument pointer (AP) (R9 that is 006a0998) at the point where the core dump occurred.

f. Use the FP mark with ‘‘]’’ (the beginning of the frame data) and use the AP mark with ‘‘a’’ (the beginning of the current functions arguments.

```
006a0930:   0064b494    006a088c    006a08c8    00000000    .d.. .j.. .j.. ....
006a0940:   00000000    00000001    00000001    0000000b    .... .... .... ....
006a0950:   ffffffd1    006a083c    (00010000   00000048    .... .j.< .... ...H
006a0960:   00000009)   0064bc62    006a0920    006a0958    .... .d.b .j.  .j.X
006a0970:   [006479d0   006a0920    006a0958    0064796c    .dy. .j.  .j.X .dyl
006a0980:   0000008b    ffffffd1    006a083c    00000000    .... .... .j.< ....
006a0990:   00660484    00660484]   (00660484)  0064941c    .f.. .f.. .f.. .d..
006a09a0:   006a0958    006a0998    [006a0770   006a07ac    .j.X .j.. .j.p .j..
006a09b0:   010102a5    006a0a1c    00000040    00000000    .... .j.. ...@ ....
006a09c0:   00000000    00000000    00000000    00000000]   .... .... .... ....
006a09d0:   3230382e    63667473    686c0000    00000000    208. cfts hl.. ....
  ...
```

g. You are now at the stack frame (Figure 3-3) for the routine which requested the core images. The next step is to locate its caller.

From the FP, (‘‘]’’), count back ten words reserved for preserving registers and mark the beginning of the register with a ‘‘[’’.

h. Count back and underline the next three words. These are the return addresses to the calling function [old program address (PA) or OPA], the calling functions argument pointer (OAP), and the calling functions frame pointer (OFP).

i. Put the ‘‘)’’ mark in front of the OPA to enclose the current functions arguments in parenthesis.

j. Iterate through this process using the OAP and OFP until the point is found where the problem program caused an exception.

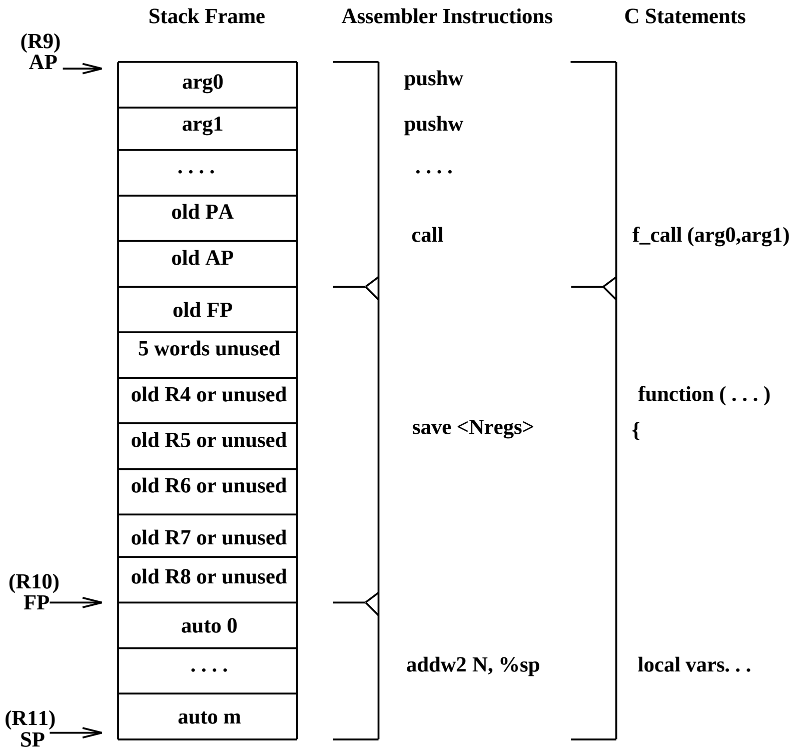


Figure 3-6. Stack Frame Layout

6. The next question is: When to stop? There are two methods to follow.

   a. You can trace back until you finally end up in your own code (that is, not 0x64nnnn). Of course, there may be many levels of shared library function calls.

   b. You can trace back until you reach the libc fault entry (*$fault*). This is the shorter method.

7. Assume we decided to trace back until we found the OPA inside of *$sfault*. In the current **libc** (<R21>), we should find an old PA of 0x64b4b0 (beware, this may vary from load to load). The stack frame for this is located. This routine accepts two arguments.

sfault (isp,fault)
*struct instat *isp;*
FCODE fault:

So the first argument (pointed to by AP) will be the pointer to an *instat structure*. The word after will be fault code (FCODE). The FCODE is a byte and should match that at 0x600095 (<R1>) 0x60084a (<R6> and <R21>). The *instat structure* is a save area for the registers at the time of the interrupt (in this case a fault).

```
/*
* Processor status saved on an interrupt in UNIX* RTR operating system.
*/
struct     instat{
           MADDR i_pa;,           /*program counter*/
           PSW i_psw;             /*psw*/
           SBR i_psbr;            /*primary segment base register*/
           SBR i_ssbr;            /*secondary segment base register*/
           MREG i_reg[NREG];      /*general purpose registers*/
};
```

The *i_pa* (the word pointed to by the address contained at AP) will be the PA at which the fault occurred. Further back tracing can be performed by picking up the *fp* from *i_reg[10]*.
The argument pointer, *ap*, is *i_reg[9]*.
The stack pointer, *sp*, is *i_reg[11]*.

```
006a07c0:   006a06b0    006a0760    7effffff    00660644
006a07d0:   0064b00e    006a0790    006a07c4    20000000
006a07e0:   006a06d0    00000000    00000001    006a0078
006a07f0:   00000100    20000000    006a06e8    006a06e8
006a0800:   ()0064ad6a  006a0770    006a07ac    [006a07ac
006a0810:   006a07fc    006a07d0    00002b82    00001b5c
006a0820:   00000001    006a0078    00000001    20000000
006a0830:   006a0720]   00660644    006a0674    +++isp=>+0064b104
006a0840:   80054880    ff3da001    ff3da001    0066068c
006a0850:   00649498    00660004    00001b5c    00000001
006a0860:   000dfc00    00660a8c    00661f5c    00661394
006a0870:   006a0800    006a0834    006a083c    006a0848
006a0880:   00037ed6    00660b34    ac000000    sfault(*isp=006a083c
006a0890:   000000d1    ()0064e606  006a0800    006a0834
006a08a0:   [006a0754   006a078c    006a078c    00000000
006a08b0:   00001b5c    00000001    000dfc00    00660a8c
006a08c0:   00661f5c    00661394]   00027b58    005a0a2e
006a08d0:   005a0a40    00000016    ffffffff    005a03b6
006a08e0:   006a0674    005a0a3f    00000001    00501260
006a08f0:   006a0840    006a0880    00000000    00000000
006a0900:   00000000    00000000    00000000    00000000
006a0910:   00000000    0000000d    00000000    0000003f
```

* UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Generic Access Package

4

Contents

Generic Access Package

4

Overview

The Generic Access Package (GRASP) in the 3B20D and 3B21D computers provides a set of utility functions used to read, move, and (with restrictions) overwrite data contained in any addressable location in the system. In addition, breakpoints can be set up to create and save procedures for later use.

GRASP capabilities include the following:

- Data transfer functions
- Breakpoints
- Breakpoint manipulation commands
- Override commands
- Traces.

GRASP uses the utility circuit/dual utility circuit (UC/DUC) hardware to access the 3B20D/3B21D computer. See Table 4-1.

Table 4-1. Utility Circuit/Dual Utility Circuit Hardware

Circuit Pack Name	UC or DUC*	Software Release	FTS Supported	Trace Memory Capacity	Process Platform
UN21	UC	R1 & Later	No	256 entries	3B20D computer (non-VLMM)
UN61	DUC	R6 & Later	Yes	2K entries	3B20D computer
UN615	DUC	R6 & Later	No	8K entries	3B20D computer
UN379	UC	R21 & Later	No	16K entries	3B21D computer

* A dual utility circuit (DUC) is accessible either across the backplane, like the utility circuit (UC), or externally by a port it provides for the field test set (FTS). The FTS was not updated for the UN615 circuit pack even though the UN615 provides a port accessible to the FTS.

Breakpoint functions appear the same with all circuit packs.

If the circuit pack is not installed in either the 3B20D or 3B21D computer or fails during use, or the field test set (FTS) is installed, GRASP clears all affected breakpoints and invalidates the trace mechanism. All other GRASP features are still available. In addition, GRASP rejects all new hardware breakpoint definitions.

NOTE:
The FTS is a separate debugging processor that can be connected to the UN61 circuit pack only. When the FTS is connected, it appears to the computer that the UN61 circuit pack is not installed.

On the 3B20D computer , GRASP must be notified when a working utility circuit is installed via the **INIT:UC** input command. After receipt of this input message, GRASP again allows trace and hardware breakpoint definitions. This procedure is not necessary on the 3B21D computer since the utility circuit is pre-initialized.

In the *UNIX** Real-Time Reliable (RTR) operating system, Release 6, the enhanced GRASP (EGRASP) feature is available in the 3B20D/3B21D computer. This feature is provided as an alternative to the FTS for real-time software debugging. EGRASP is a resident software package that provides on-line real-time software debugging capabilities. It supports an interface to the UN615 and UN379 circuit packs to provide all of the existing GRASP functions (for example, the capability to place multiple breakpoints in code, to read and write memory registers, and to dump the contents of memory), in addition to the new trace and matching functions.

All GRASP commands have man-machine language (MML) equivalents. The examples given here are in the program documentation standard (PDS). See the MML manuals for syntax details.

Data Transfer Functions

Table 4-2 lists the input messages that move, print, and with certain restrictions, write data into any addressable location in the system. See the *UNIX* RTR operating system manuals (refer to Table 1-1, IPs Supporting the 3B21D Computer in ‘‘About This Information Product" chapter) for details on any specific input message and for system responses.

* UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Table 4-2. Data Transfer Commands

To:	Use Command:
Transfer data in main memory to a utility variable (immediate)	**COPY:UID;UVAR** **PID**
Transfer data in main memory to a utility variable, main memory, or register (on breakpoint)	**COPY:ADDR;UVAR** **ADDR** **REG**
Transfer data in a utility variable to another utility variable (immediate)	**COPY:UVAR;UVAR**
Transfer data in a utility variable to main memory, another utility variable, or a register (on breakpoint)	**COPY:UVAR;UVAR** **ADDR** **REG**
Transfer data in a register to a utility variable (immediate)	**COPY:REG;UVAR**
Transfer data in a register to main memory, a utility variable, or another register (on breakpoint)	**COPY:REG;UVAR** **ADDR** **REG**
Display data in main memory at the maintenance terminal and print the data at the receive-only printer (ROP) (immediate)	**DUMP:UID** **PID**
Display data in main memory at the maintenance terminal and print the data at the ROP (on breakpoint)	**DUMP:ADDR**
Display data in the kernel's address space at the maintenance terminal and the ROP (immediate)	**DUMP:KERN**
Display data in a utility variable at the maintenance terminal and print the data at the ROP (immediate or on breakpoint)	**DUMP:UVAR**
Display data in a readable register at the maintenance terminal and print the data at the ROP (on breakpoint)	**DUMP:REG**

Table 4-2. Data Transfer Commands (Contd)

Display data in physical memory at the maintenance terminal and print the data on the ROP (immediate or on breakpoint)	**DUMP:PMEM**
Load a location in main memory with specified data (on breakpoint)	**LOAD:ADDR**
Load data into a utility variable (immediate or on breakpoint)	**LOAD:UVAR**
Load data into a writable register (on breakpoint)	**LOAD:REG**
Load a location in physical memory with specified data (immediate or on breakpoint)	**LOAD:PMEM**

Breakpoints

Breakpoints detect the existence of a set of specified conditions on the machine. The definition of a breakpoint has two parts: (1) description of conditions that are to be matched and (2) list of actions (maximum of five action clauses) to take place when the match occurs.

The **WHEN** command starts a list of GRASP commands that are performed when a specified breakpoint condition exists.

The list must be terminated with the **END:WHEN** command. The **END:WHEN** command is not counted as a part of the action list itself and does not count against the limit of five action clauses.

In PDS format, all commands in the *WHEN* list are terminated with an exclamation point ‘‘**!**’’.

In MML, all commands are terminated with a semicolon ‘‘**;**’’.

After a **WHEN** command, with its conditions and action list, is entered successfully, the breakpoint is assigned a number by GRASP. The breakpoint is then referred to exclusively by its number. Up to 20 different breakpoints can be defined in the system at any time. The numbers assigned to breakpoints during a debugging session are not reused unless the RESET option is specified when clearing breakpoints.

GRASP prints two output messages in response to a breakpoint definition after the PF (print follows) is given. The first message assigns a number to the breakpoint. This message should appear soon after the PF. The second message confirms that the breakpoint was set up successfully or indicates that the breakpoint was aborted and gives the reason.

When the breakpoint is set up, the second of the breakpoint messages is actually printed. The message indicates either that the breakpoint was set up successfully or, if unsuccessful, the reason for its failure.

When a breakpoint fires, its action list is executed from top to bottom. The **INH:UTILFLAG ME** command, if used, can be anywhere in the work list without affecting the rest of the actions being executed in the action list for that firing of the breakpoint. A count of the number of times the breakpoint has fired is kept. All output generated by the action list is labeled with the breakpoint number and the firing number as shown in the following example:

Example Breakpoint Output

Breakpoint Definition

```
WHEN: UID X’112, ADDR X’20130; W!
DUMP: REG PA!
DUMP: ADDR X’20130!
END: WHEN!
```

Output Produced When Breakpoint Fires

```
REPT GRASP BREAKPOINT FIRED
UTILID = X’112 PID = ________ BREAKPOINT = 1 FIRENUM =1
REGISTER:     CONTENTS(X’):
PA:           X’00005286
DUMP REG COMPL #G1
ADDRESS(X’):                 CONTENTS OF MEMORY(X’):
20130:                        0000016A
DUMP ADDR X’20130 COMPL #G2
```

NOTE:
Each time the breakpoint fires, *FIRENUM* increases by 1.

Breakpoint on Execution of an Instruction

Breakpoints that fire on execution of an instruction are called software breakpoints because of the way they are implemented. The breakpoint itself is an instruction that transfers control to GRASP when it is executed. See the input messages manual, **WHEN:UID** or **WHEN:PID** for details. Refer to Table 1-1, IPs

Supporting the 3B21D Computer, in ‘‘About This Information Product" chapter for a list of these manuals. One exception is when the action list contains any command that controls or affects a trace. (Refer to the ‘‘Trace’’ section.) When a trace is affected, the breakpoint is implemented in hardware rather than software. In *UNIX* RTR operating system Release 1 and later, starting a trace is implemented in software for execution (EXC) breakpoints.

Software breakpoints are set up [at the location specified by the utility identification (UID) or process identification (PID), and *ADDR* keywords of the **WHEN** command] as soon as possible after the breakpoint is defined. The opcode itself is not changed until the breakpoint is enabled.

Processes are described by the UID or the PID and, in some cases, a user process name. However, more than one process can be active with the same UID and process name. When this happens, GRASP sets up the first breakpoint in one of the matching processes at random. If another breakpoint is defined for the same UID or PID, GRASP sets up the breakpoint in the same process as the first.

If a process on the machine does not match the described conditions, the breakpoint is not set up. However, any of the commands for manipulating breakpoints listed in the ‘‘Breakpoint Manipulation Commands’’ section can be used.

The *OPC* parameter is required on software breakpoints to avoid the problems caused by out-of-date disassembly listings. Severe problems occur if a breakpoint is accidentally set up in the data portion of an instruction.

Because the breakpoint opcode is not placed until the breakpoint is enabled, a disabled software breakpoint does not fire and does not use any machine resources.

Because of the way software breakpoints are implemented, the breakpoint fires before the instruction where it is placed executes. The instruction in the original text is saved before it is overwritten by the breakpoint instruction. Only after the breakpoint fires and the action list is executed, is the displaced instruction executed. Execution then resumes with the instruction following the displaced one. For hardware implemented breakpoints, the breakpoint fires after the instruction where it is placed executes.

Table 4-3 summarizes the breakpoint implementation types (*H* - hardware, *S* - software), which depend on two factors: breakpoint mode (*EXC, R, W,* or *RW*) and the presence or absence of trace commands in the action list.

Table 4-3. Breakpoint Implementation Types

Mode	^Start or Stop Trace in the Action List*	No Trace Operations in the Action List
EXC	H	S
R	H	H
W	H	H
RW	H	H

* In *UNIX* RTR operating system Release 1 and later, starting a trace within the breakpoint is implemented as software breakpoint when mode is EXC.

Breakpoint on Access of Data

Breakpoints that fire on accesses of data are implemented with hardware using matchers on either the UN21, UN61, UN615, or the UN379 circuit packs. Hardware breakpoint functions appear, to the user, to be identical with all circuit packs.

To set up a hardware breakpoint, GRASP configures the matchers that are needed and supplies the values that are to be matched. The circuitry continually compares the values with what is taking place on the machine. If the breakpoint is enabled, the breakpoint fires when all the matchers specified during set up match. If a hardware breakpoint is disabled, the hardware passively tries to match but does not interrupt processing on the machine. Disabled hardware breakpoints do not use any resources of the machine.

Because the amount of hardware on the circuit pack is limited, a maximum of four hardware breakpoints can be defined at one time. Because the trace also uses hardware matchers, fewer breakpoints are available while a trace is defined.

If the particular matchers needed are not available, it is possible to have fewer than four hardware items defined but have a command rejected for lack of resources. On the 3B20D/3B21D computer, this is generally true when using an address range. Only one matcher on the utility circuit can be set up to match on a range of addresses. A second command requesting an address range is rejected even though a breakpoint using a single address is accepted. On the 3B20D/3B21D computer, there are three matchers on the utility circuit that can match on an address range. Therefore, all three hardware breakpoints may have an address range.

The following is a hardware breakpoint example and the resulting system response. See the appropriate input or output message manual for specific details. Refer to Table 1-1, IPs Supporting the 3B20D/3B21D Computer, in ‘‘About This Information Product" chapter for a list of these manuals.

- **Example 1**

  Breakpoint Definition

```
WHEN:UID  X´112, ADDR X´20130; W!
DUMP:REG  PA!
DUMP:ADDR X´20130!
END:WHEN!
```

  System Response

```
WHEN UID  X´112  ADDR  X´20130  STARTED HARD1 #G1
WHEN UID  X´112  ADDR  X´20130  COMPL DISABLED1 #G2
```

- **Example 2**

  Breakpoint Definition

```
WHEN:UID  X´112, ADDR X´86E, OPC X´41;EXC!
INH:UMEM!
END:WHEN!
```

  System Response

```
WHEN UID X´112 ADDR X´86E STARTED HARD1 #G1
WHEN UID X´112 ADDR X´86E COMPL DISABLED1 #G2
```

Breakpoint on External Condition

A breakpoint can be defined to fire upon receiving an active external event backplane signal. This is implemented using a hardware trigger and the circuit pack matcher. If the condition matcher is already being used for a trace, a trigger allocation error results if an attempt is made to define a condition breakpoint.

The condition breakpoint fires immediately upon receipt of the external event regardless of an executing process. The processor can in fact be idle when this occurs. In this event, any register copy and load commands inside of the breakpoint action list deal directly with the current machine registers, which may be of limited value. If a process is running when the breakpoint fires, register copy and loads refer to the saved values of the interrupted process. This feature would be most useful in connection with some external analysis equipment such as a logic analyzer triggering the event. The breakpoint will continue to fire if not inhibited inside the action list as long as the external event signal is active.

Example

Breakpoint Definition

```
WHEN:COND E!
DUMP:REG PA!
INH:UTILFLAG ME!
END:WHEN
```

System Response

```
WHEN COND E STARTED HARD 1 #G5
WHEN COND E COMPL DISABLED 1 #G6
```

Breakpoint Manipulation Commands

Breakpoints can be allowed or inhibited from firing, their definitions can be cleared, and a summary of all breakpoints can be printed. The commands to manipulate breakpoints are given in Table 4-4. See the input/output messages manuals for details on any specific input message and for system responses. Refer to Table 1-1, IPs Supporting the 3B21D Computer in ‘‘About This Information Product" chapter for a list of these manuals.

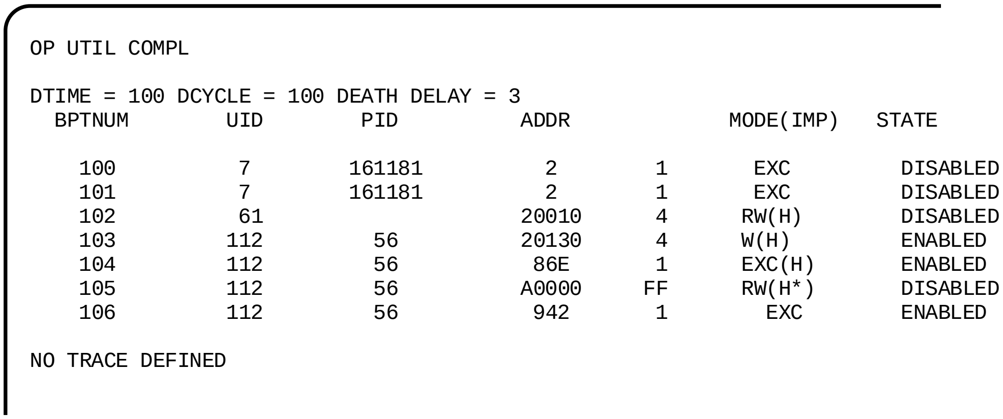
Table 4-4. Breakpoint Manipulation Commands

To:	Use Command:	System Response
Enable an individual breakpoint	**ALW:UTILFLAG** a **!** where a = breakpoint number assigned when breakpoint was defined	**ALW UTILFLAG** a **COMPLETED**
Enable all breakpoints	**ALW:UTIL !**	**ALW UTIL COMPLETED**
Disable an individual breakpoint	**INH:UTILFLAG** a **!** where a = breakpoint number assigned when breakpoint was defined	**INH UTILFLAG** a **COMPLETED**
Disable all breakpoints	**INH:UTIL !**	**INH UTIL COMPLETED**
Cause a breakpoint to disable itself in an action list	**INH:UTILFLAG ME !**	**INH UTILFLAG ME COMPLETED**

Table 4-4. Breakpoint Manipulation Commands (Contd)

To:	Use Command:	System Response
Display breakpoint status	**OP:UTIL !**	The **OP:UTIL** output includes the breakpoint number, the UID (in hexadecimal), the process id (in decimal), the address (in hexadecimal), length, the mode (*R, W, RW,* or *EXC*), and the state (*ENABLED* or *DISABLED*) for every breakpoint currently in the system. The mode is annotated with *H* for hardware items and the user process name, if specified. If no breakpoints are set up, the system response is: **OP UTIL COMPLETED** **NO FLAGS DEFINED**
Clear an individual breakpoint	**CLR:UTILFLAG** a **!** where a = breakpoint number assigned when breakpoint was defined	**CLR UTILFLAG** a **COMPLETED** (if successful) **CLR UTILFLAG** a NGINST (if unsuccessful) **CLR UTILFLAG** a **NOT STARTED** conflict with current system status (if breakpoint not defined)
Clear all breakpoints	**CLR:UTIL;[RESET]**	The **CLR UTIL COMPLETED** breakpoint number for any additional breakpoints during this GRASP session will begin at 1 if the RESET option is used.

See Figure 4-1 for an example and explanation of the **OP:UTIL** command.



```
OP UTIL COMPL

DTIME = 100 DCYCLE = 100 DEATH DELAY = 3
  BPTNUM        UID        PID          ADDR            MODE(IMP)   STATE

    100          7       161181           2        1       EXC         DISABLED
    101          7       161181           2        1       EXC         DISABLED
    102          61                     20010      4      RW(H)        DISABLED
    103         112         56          20130      4      W(H)         ENABLED
    104         112         56           86E       1      EXC(H)       ENABLED
    105         112         56          A0000     FF      RW(H*)       DISABLED
    106         112         56           942       1        EXC        ENABLED

NO TRACE DEFINED
```

Figure 4-1. Example *OP:UTIL* Command Output

As shown in Figure 4-1:

- Seven breakpoints are defined (20 are allowed).
- No traces are defined.
- Four of the breakpoints are hardware breakpoints; therefore, no triggers are left.
- Breakpoint 105 is using the range matcher.
- Even though breakpoint 104 is in the *EXC* mode, it is implemented with hardware (it starts or stops the trace)(Generic 2 only). After Generic 2, only stopping a trace causes a software breakpoint to be implemented with hardware.
- All of the breakpoints for a particular UID are planted in the same PID.
- Breakpoint 102 is not set up (implying the process is not currently active).
- Some of the breakpoints are *ENABLED* while others are *DISABLED*.

Override Command

The input message, **IN:DTIME**, overrides the GRASP default dynamic real-time time limit. This input message increases the maximum allowable time limit of the dynamic timer. See input/output messages manuals for details on any specific input message format and for resulting system responses. Refer to Table 1-1, IPs Supporting the 3B21D Computer, in ‘‘About This Information Product" chapter for a list of these manuals.

If GRASP uses all of the time it is allowed according to the value of the dynamic real-time limit, an output message is printed indicating that all GRASP breakpoints were inhibited. The breakpoints must be selectively reallowed.

The output message,

REPT GRASP DYNAMIC RESET

indicates that a GRASP real-time limit override has expired and has been reset to the normal installation default value.

Example

Overriding the dynamic timer

IN:DTIME 10000; UNTIL 2359

System Response

IN DTIME COMPLETED

Trace

GRASP supports a trace feature as a regular part of the *UNIX* RTR operating system releases. The trace feature permits you to record and view the flow of program execution on the machine. The trace can be used in either of two ways: (1) to record the events leading to a target event or (2) to record program flow following a target event.

Trace Operation

This section describes trace states and transitions, discusses trace hardware issues, and gives details on trace options.

States and Transitions

The five operations available to trace program flow and the commands to implement these operations are shown in Table 4-5.

Table 4-5. Trace Operations

Operation	**Command**
initialize	**INIT:UMEM**
start	**ALW:UMEM**
stop	**INH:UMEM**
dump	**OP:UMEM**
clear	**CLR:UMEM**

Any of these operations can be done as immediate operations. Only the commands to start and stop the trace are allowed in breakpoint action lists.

The trace can be in any one of the following states:

- Undefined
- New
- Running
- Stopped
- Dumped.

Before any trace command is executed, the trace state is checked and the command is rejected if it is logically incorrect for the trace state. The allowed transitions are shown in Figure 4-2.

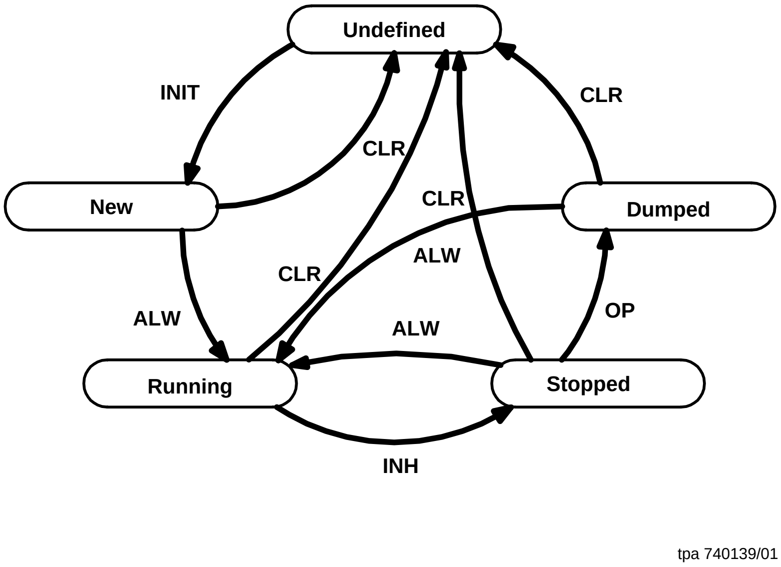


Figure 4-2. Trace State Diagram

For trace commands in breakpoint action lists, only minimal state checking is done when the breakpoint is defined. A command to start the trace is rejected if the trace is undefined. The full state check is done only at the time the breakpoint fires and the action list is executed. Rejected trace commands do not affect the rest of the action list processing.

The trace operations fall into two classes, slow and fast, according to the amount of data they move to or from the circuit pack. The slow operations initialize the trace and dump its memory. These operations currently take over 20 milliseconds and are performed at execution priority level 3. The other operations are fast and add little time when used in breakpoint action lists.

Hardware Issues

The trace is controlled by one to four circuit pack triggers depending on trace type. As long as a trace is defined, one of the circuit pack triggers is unavailable for breakpoints. The trigger used is one that allows a range of data addresses to be matched. On the 3B21D computer, two additional triggers are available for address ranges. The triggers are not available to set up two more traces. Hardware breakpoints set on an address range are marked with an asterisk in the **OP:UTIL** command output.

Hardware implemented execution breakpoints differ from software implemented execution breakpoints in one respect. That is, the breakpoint action list for a software implemented breakpoint is executed *before* the instruction where the breakpoint is set up. For a hardware implementation, the action list is executed *after* the instruction. Keep this in mind when defining the breakpoint and interpreting its output.

Because only one matcher that traps the execution of an address is available on the circuit pack, only one execution breakpoint that controls the trace can be defined at one time. In *UNIX* RTR operating system Release 1 and later, starting a trace from an execution (EXC) mode breakpoint is implemented in software. This allows control of the transfer trace with two execution breakpoints.

In summary, when a trace is defined on a 3B20D computer:

- The data access range matcher is unavailable for breakpoints
- Only one execution breakpoint can control the trace
- At most, only three data access breakpoints can be defined (two if an execution breakpoint controls the trace) depending on trace type.

At most, two access range matchers are available on the 3B21D computer.

Trace Options

Several options are available to tailor the exact type of information that is recorded in the trace memory. These options are described in the following paragraphs.

UID Trace

Because the trace memory is limited, the duration of the trace is inversely proportional to the amount of detail recorded. One way to get a long history of activity is to restrict the trace to store only the UIDs of the processes that run. This gives a good, long picture but little resolution. With this type of trace, the output indicates every process switch including those to the kernel and the special processes. For more detail on how to read the trace output, see the ‘‘Interpreting Trace Output Formats’’ section.

Transfer Trace

An alternate method (which is the default) is to store the addresses involved in every nonsequential change in execution flow. That is, for every branch, jump, call, and return instruction, the address of the instruction (or *from* address) and the destination (or *to* address) are recorded. In addition, whenever a change in process occurs, the new process UID is recorded so the *to* and *from* address can be interpreted in context. This gives more detail than the UID-only option.

Data History Trace

The data history mode allows recording of the program data accesses. Each time a data access occurs, the trace memory records the data, the data address, the current program address, and a flag indicating a read or write operation. All four trigger functions are capable of controlling the recording activity. When an address range is specified on the **INIT:UMEM** input message, the block matcher is used and the trace only records data when a memory location within the range is accessed.

Simultaneous Data History and Transfer Trace

A simultaneous data history and transfer trace records all data associated with a data history trace and a transfer trace. The read/write flag indicates data accesses and is only displayed on the 3B21D computer. The data history and transfer traces are described previously. When an address range is specified on the **INIT:UMEM** input message, the block matcher is used and the trace only records data when a read or write instruction or a transfer occurs within the address range.

Both trace types are set in the utility circuit control register, bit 1 for the transfer trace and bit 4 for the data history trace. In order to perform both traces simultaneously, bit 11 of the control register must also be set. On the 3B20D computer, the reason reads and writes cannot be distinguished for the data history trace data in the simultaneous trace is because the read/write flag (bit 0 of trace memory) is used to distinguish between the type of data recorded for each line of trace memory. On the 3B21D computer, the width of the trace memory was expanded to allow for the recording of the read/write flag.

Function Trace

The function trace memory mode records software function changes. The 3B20D computer native instructions SAVE and RETURN are set up using opcode matchers and any other conditions established by the **INIT:UMEM** input message. When a SAVE instruction is executed, the *CALL address* (the previous program address), the *SAVE address*, and the current UID value are recorded. Execution of the RETURN instruction allows trace memory to record the *RETURN address* (the current program address), the following program address, and the current UID value.

On the 3B20D computer, when using an address range with function trace, the range specified must be matchable with a circuit pack address matcher. This is more restricted than UID, transfer, data history, or simultaneous data history and transfer traces (these traces use the block matcher and can therefore match on any specified address range). In order to use the address matcher for an address range, the starting and ending addresses must be of a form where the leftmost hex digits of both are equal, and the rightmost digits of the starting address are all ‘‘0’’ and the rightmost digits of the ending address are ‘‘F’’ (for example, 0x123000 – 0x123FFF or 0x10000 – 0x1FFFF). On the 3B21D computer, an address range with an exact end address can be specified when setting up a function trace. A function trace uses two triggers.

Function with Parameters Trace

The trace of functions with parameters records call instruction address, save instruction address, and parameters pushed on the stack. The stack address and stack size may be specified with the **INIT:UMEM** input message. If these values are not supplied, default values will be used. Unlike the function trace, return instructions will not be recorded. The *ADDR* keyword may not be used with function and parameter traces to restrict the address range of function calls which are recorded. This is due to the difficulty in resolving which stack writes are due to function calls outside of an address range which would not be recorded.

On the 3B20D computer, an address matcher is used to detect stack writes. If a stack address and stack size are specified, they must specify an address range as described in the previous section. For example, the default stack address is 0x6A0000 and stack size is 0x1000. This specifies an address range of 0x6A0000 through 0x6A0FFF. A function with parameters trace uses two triggers.

Simultaneous Data History and Function with Parameter Trace

The simultaneous trace of data and functions with parameters trace records data history trace information. The read/write flag is only recorded on the 3B21D computer. The data history and function traces were described in previous paragraphs.

As with the previous trace type, on the 3B20D computer, if a stack address and range are specified, they must describe a range that can be matched with an address matcher. In addition, if a data history address range is specified, it too must be of this form (for example, 0x2000 – 0x2FFF). This trace uses three triggers.

On the 3B21D computer, an exact end address may be specified.

UID Restriction

The trace can be restricted to trace only while a particular process is running using the *UID* option. The UID specified on the input message is the *pcode* of the process to be traced. Any copy of the process with that pcode is traced, and since the UID recorded whenever the process changes includes the dct slot, multiple incarnations of a pfile can be distinguished.

ADDR - Address Range Selection

The *ADDR* keyword limits program tracing to the access of a specific word of memory or to a given range of addresses. When a trace uses a block matcher, any address range can be specified. This is the case for UID, transfer, data history, and simultaneous data history and transfer trace.

The function trace uses an address matcher to implement the ADDR range. This is more restricted and is described in the function trace section. The *ADDR* keyword is not implemented for function with parameter traces. For simultaneous data history and function with parameter traces, the *ADDR* keyword uses an address matcher to specify the data history address range.

Stop When Full Versus Circular

The trace can be set up either to automatically stop tracing when the memory fills up or to trace indefinitely, always replacing the old data with the new. This pair of options is used to set up the various scenarios of tracing as described in the next section. The options are independent of the type of data stored.

If the *STOP FULL* option is chosen, an output message is printed indicating that the trace stopped for that reason.

Stop Trace on Condition

The *COND* keyword may be specified along with any combination of E, MRF, and SAS to stop a running trace if one of the following conditions occur:

E — Stop the trace if an external event occurs. This is triggered by the external event backplane signal on the DUC/UC circuit pack.

SAS — Stop the trace if a hardware stop-and-switch occurs.

MRF — Stop the trace if a hardware maintenance reset function occurs.

Using the condition, matcher uses another trigger for the trace.

Trace Scenarios

The following paragraphs describe the most common trace scenarios. The trace input messages and associated output messages are shown in Table 4-6.

Table 4-6. Trace Input and Output Messages

Input Messages	Output Messages
INIT:UMEM	INIT UMEM
ALW:UMEM	ALW UMEM
INH:UMEM	INH UMEM
OP:UMEM	OP UMEM
CLR:UMEM	CLR UMEM

See the input/output messages manuals for details on any specific input message and system responses. Refer to Table 1-1, IPs Supporting the 3B21D Computer, in ‘‘About This Information Product" chapter for a list of these manuals.

Before Trace

To record the sequence of execution that precedes a known event, do the following:

- Decide what type of trace to use. There are seven trace types.
  - UID Trace
  - Transfer Trace
  - Data History Trace
  - Simultaneous Data History and Transfer Trace
  - Function Trace
  - Function with Parameters Trace
  - Simultaneous Data History and Function with Parameter Trace.
- Decide whether to restrict the trace to a particular UID or PID or to allow all processes to be traced. These decisions depend on the scope of the problem being debugged (system wide versus internal to a process) and the length of history needed.
- Start the trace in the circular mode and define a breakpoint to trap the target event and stop the trace. When the breakpoint fires and the trace stops, the history leading up to the event will be in the trace memory.

The command sequence to implement this scenario is shown in Table 4-7.

Table 4-7. Before Trace Scenario

Command	Action
INIT:UMEM [,UID __][,STORE UID]!	Initialize the trace. (Default = circular mode and store transfers)
ALW:UMEM!	Start the trace.
WHEN:UID...! . . . **INH:UMEM!** **END:WHEN!**	Set a breakpoint on the target event. See the **WHEN:UID** command in the input/output manuals for details. Refer to Table 1-1, IPs Supporting the 3B21D Computer, in ‘‘About This Information Product" chapter for a list of these manuals Add the command to stop the trace when the target event occurs and any other actions desired.
ALW:UTILFLAG__!	Enable the breakpoint just defined to fire.
OP:UMEM!	Dump the trace memory after the breakpoint fires and the trace is stopped.

This scenario can be repeated by restarting the trace (and re-enabling the breakpoint if **INH:UTILFLAG ME** was used in the action list), or it can be cleared with the

CLR:UMEM!

command. The breakpoint should also be cleared with

CLR:UTILFLAG ___!

at this time.

Execution of the **INH** command in the action list is ignored if the trace is not running.

After Trace

To see the sequence of execution that occurs after a target event, do the following:

- Decide what type of trace to use.
- Decide whether to restrict the trace to a particular UID or PID or to allow all processes to be traced.
- Configure the trace to stop when trace memory is full.
- Define a breakpoint to trap the target event and start the trace.

The proper sequence of commands for this scenario is shown in Table 4-8.

Table 4-8. After Trace Scenario

Command	Action
INIT:UMEM [,UID __] **,STOP FULL [,STORE UID]!**	Initialize the trace. (Default = store transfers)
WHEN:UID...! **.** **.** **.** **ALW:UMEM!** **END:WHEN!**	Set a breakpoint on the target event. See the **WHEN:UID** input command in the input/output messages manuals for details. Refer to Table 1-1, IPs Supporting the 3B21D Computer, in ‘‘About This Information Product" chapter for a list of these manuals. Add a command to allow (ALW) the trace and any other actions desired. Use **INH:UTILFLAG ME** when applicable. See Note.
ALW:UTILFLAG __!	Enable the breakpoint to fire. Output messages indicate when the trace starts, when the trace memory is full, and when the trace stops.
OP:UMEM	Dump the trace memory.
CLR:UMEM [,UCL]!	Clear the trace.
CLR:UTILFLAG __!	Clear the breakpoint.

Note:

Once the trace memory is dumped, the trace is restarted if the breakpoint fires again and the action list associated with the **ALW:UMEM** command is executed. To prevent this, insert **INH:UTILFLAG ME** in the breakpoint action list. In this case, the trace can only be restarted if the breakpoint is again enabled with **ALW:UTILFLAG __!**

Between Trace

A trace can be set up to record data between two target events (up to the memory limit of the circuit pack installed). The breakpoint used to trap one target event starts the trace and another breakpoint defined for the other event stops the trace. To record this information, do the following:

- Use the *STOP FULL* option on the **INIT** command to indicate whether any data gets lost because of the finite size of the trace memory. The data lost, if any, is the new data. If the new data is needed, repeat the trace in the *circular* mode. In the circular mode, the old data is lost, preserving the new data. Because lost data is not apparent in the *circular* mode, you should use the *STOP FULL* option.
- Use the *UID* option to restrict the trace to those processes from a particular pfile or the PID option to restrict the trace to a particular process incarnation.
- Because of circuit pack hardware restrictions, choose the two breakpoints carefully. Only one breakpoint that controls the trace is allowed to be of type *EXC*, that is, to trap the execution of a particular text address. The other breakpoint must be a data access breakpoint. On the 3B20D computer, because the address range matcher is unavailable while a trace is defined, any data access match must be a single address. However, on the 3B21D computer , any data access match can be an address range.
- In Generic 2, if the trace needs to be both started and stopped on the execution of text addresses, one of the breakpoints must be a data access breakpoint. Because it is usually easier to find a data address that is written for the first time near the address where the trace is to be started than it is to find a data address that is written for the first time near the address where the trace is to be stopped, the data access breakpoint is generally more suitable to starting the trace than to stopping it. An execution breakpoint that starts the trace is implemented in software. Thus, the above restriction is not required.

The command sequence to implement this scenario is shown in Table 4-9.

Table 4-9. Between Trace Scenario

Command	Action
INIT:UMEM [,UID __] **[,STORE UID],STOP FULL!**	Initialize the trace.
WHEN:UID ... ! **ALW:UMEM!]** **END:WHEN!** . . .	Set the breakpoint on the first target event. See the **WHEN:UID** command in the input message manuals for details. Refer to Table 1-1, IPs Supporting the 3B21 Computer, in "About This Information Product" chapter for a list of these manuals. Add the command to allow the trace and any other actions desired.
WHEN:UID ... ! . . . **INH:UMEM!** **END:WHEN!**	Set the breakpoint on the second target event. Add the command to inhibit the trace and any other actions desired.
ALW:UTILFLAG __! **ALW:UTILFLAG __!**	Enable the defined breakpoints.

Enable the **INH:UMEM** breakpoint first, then the **ALW:UMEM** breakpoint, or use **ALW:UTIL**.

When the start breakpoint fires, an output message reports that the trace started. Another output message reports that the trace stopped. It is either the message from the **INH:UMEM** action of the second breakpoint, or it is a *REPT GRASP* message resulting from the *STOP FULL* directive (if specified). Either way, dump the data with:

OP:UMEM!

If *STOP FULL* was specified and the **REPT GRASP** message indicated that the trace stopped before the second breakpoint fired, some data was lost before the firing of the second breakpoint. If this happens, rerun the trace in the *circular* mode to obtain the missing data. Then, clear the trace with

```
CLR:UMEM!
```

and reinitialize the trace as before but without the *STOP FULL* option. If the breakpoints specified **INH:UTILFLAG ME** in their action lists, re-enable the breakpoints.

After the output messages indicate that the trace has started and stopped, print the data. This printout will be missing the oldest data at the beginning of the trace. Compare both outputs to find the overlap and reconstruct the entire interval.

Interpreting Trace Output Formats

UID and Transfer Output Formats

The trace memory dumped by the **OP:UMEM** command is printed in order, oldest to newest, row by row. Every entry is one of three types: *utility id* marked with *U*, *from* address marked with *F*, or *to* address marked with *T*.

The utility id entries are 24-bit hexadecimal numbers that are presented in the format shown in Figure 4-3. The rightmost 12 bits (3 hexadecimal digits) are the process pcode. The leftmost 8 bits (2 hexadecimal digits) are the dct slot. The remaining hexadecimal digit is unused.

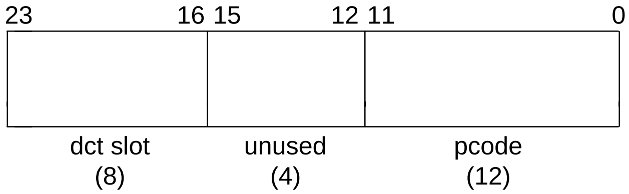


Figure 4-3. Utility id Print Format

As described in the *UNIX* RTR operating system header file, *pnum.h*, a process id (pid) consists of a dct slot or index (of which the higher order byte is always zero) and an incarnation count as shown in Figure 4-4.

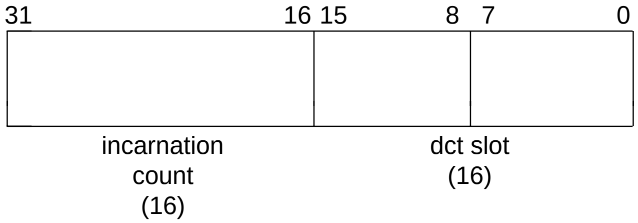


Figure 4-4. Process id

An easy correspondence can be made between the trace UID entries and the pids if the pids are expressed in hexadecimal. In kernel processes, the

OP:STATUS:PROCESS, ALLKERNS!

command (also *ps -k* to the Bourne shell) prints out the dct slot directly; however, it is in decimal and must be converted.

The address entries are all virtual addresses within the process indicated by the most recent preceding UID entry in the trace memory. Any *from* address is the address of a branch, jump, call, or return instruction that was executed. The following *to* address is the address to which control transferred. Occasionally, two *to* addresses will be recorded adjacent to each other. This implies that the first *to* address itself caused a transfer of control (not an uncommon occurrence in compiler generated code). Between any *to*, *from* pair, the code was executed without branching.

NOTE:

Although the disassembler decodes the *a1* opcode as a 4-byte return instruction, the microcode (and hence the trace output) treats it differently. The *a1* is really a 2-byte no-op instruction. The actual return instruction is the 2-byte *7b* instruction. As a result, the *from* address recorded for a return will be the address of the *7b* 2 bytes beyond the return indicated in the disassembly listing.

Typically with the UN21 circuit pack, several *to* and *from* addresses precede the first UID entry in a transfer trace. If it is important to know (but not obvious) what process they belong to, rerun the same trace scenario with the *STORE UID* option on the **INIT:UMEM** command. Working backwards from the end of the two dumps, UID entries can be matched to determine the UID of the early transfers in the first trace.

Data History Trace Format

The data history trace records the program address at which a specified address is being accessed, the data address, a read or write flag, and the data value. The format for this trace is displayed in the following sample.

```
                 DATA HISTORY TRACE

      PROGRAM     DATA
      ADDRESS    ADDRESS         DATA

      0x000076   0x3c0034   <-   0x00000004
      0x00005e   0x3c0034   ->   0x00000004
      0x00007c   0x3c0034   ->   0x00000004
      0x000090   0x020068   <-   0x61000000
```

The leftmost column contains the program address accessing a specified memory address or address within a specified range of memory addresses. The center column contains the data address, and the rightmost column contains the data value. A read operation on the data address is indicated by a right arrow -> and a write operation by a left arrow <-.

Function Trace Format

The function trace only records calls and returns and the address branched to. A sample of the output follows.

```
                      FUNCTION TRACE

  CALL OR                                   SAVE OR
    RETURN ADD.                               RETURN-TO ADDR.

  0x0000a8  call                             0x000248
    0x000272    call                           0x000420
    0x000260    reto                           0x000276
  0x000300    reto                          0x0000ac
```

Output lines contain the keywords call or reto in the second column to indicate a call or return. Calls and their respective returns are indented equal amounts to reflect nesting. For calls, the left column contains the address of the call instruction. The right column contains the save address branched to. For returns, the left column contains the address of the return instruction and the right column lists the program address being branched to.

Simultaneous Transfer Trace and Data History Format

This trace records ‘‘transfer trace’’ and ‘‘data history trace’’ data. A sample of the output follows.

```
              SIMULTANEOUS TRANSFER AND DATA HISTORY TRACE

     PROG ADD            DATA ADD      DATA VALUE (data history)
FROM-ADD     goto        TO-ADD        UID OF TO-ADD (transfer)

         0x000026        0x020010      ?    0x61000000
         0x00002e        0x020011      ?    0x00620000
     .
     .
     .
0x00029a    goto         0x00029c      u=0x17d (0x282fa0)
         0x000029e       0x6a0190      ?          0x000001a6
```

The indented output represents data history. The lines not indented represent program transfer trace data. The left column of the program trace is the *from program address.* The middle column is the *to program address*, and the right column is the *uid of the to address.* The data history’s left column is the *program address.* The middle column is the *data address*, and the right column is the *data*. On the 3B20D computer, it is not possible to know whether the data history trace is a read or a write, thus a question mark is inserted in data history trace output. On the 3B21D computer, the read or write arrow is recorded.

Function with Parameter Trace Format

The function with parameter trace records function calls and full-word data write accesses on the process stack. A sample of the output follows.

```
              FUNCTION TRACE WITH PARAMETERS PASSED

    CALL ADD          SAVE ADD     PARAMETERS/AUTOMATICS

    0x000120   call   0x0002d4     (0x00000019)
    0x0001a0   call   0x000258     (0x0000000a,0x00000020,
                                    0x00000000,0x00000002)
    0x000280   call   0x0002a0     ?(0x0000000,0x00000002)
```

The left column provides the program address of the call instruction. The next column contains the save instruction address. The remaining one or more columns enclosed within parentheses contain the parameter(s) pushed; where the last parameter pushed appears first in the list. The automatics from the previous function may also appear with the parameters. When it is unclear which parameters were pushed on the stack, a ? precedes the left parenthesis.

Simultaneous Data History and Function Trace Format

This trace records the data history and function trace data. A sample of the output follows.

```
      SIMULTANEOUS DATA HISTORY AND FUNCTION TRACE FORMAT

  CALL ADD          SAVE ADD       PARAMETERS/AUTOMATICS (function)
 PROG ADD        DATA ADD          DATA VALUE (data history)

   0x0000a4      call  0x0001d8    ?(0x00000005,0x00000045)
   0x000d0       call  0x0001a8    ?(0x00000002,0x00000063)
 0x000112         0x020038 <-      0x00000045
 0x00011c         0x02003c <-      0x61000000
 0x000126         0x020040 <-      0x00034567
```

The indented output is the function call with its parameters. Among these parameters, the automatics from the previous function may also appear. The left column is the call instruction address. The next column is the save instruction address. The next column is the parameter pushed, and the rightmost column is an automatic from the previous function.

The unindented output is the data history for the data range specified. The left column is the program address. The middle column is the data address, and the right column is the data value. On the 3B20D computer, it is not possible to know whether the data history trace is a read or a write, thus a question mark is inserted in data history trace output. On the 3B21D computer, the read or write is recorded.

UNIX® RTR Operating System Process Information (Utility IDs)

Table 4-10 lists information about the *UNIX* RTR operating system processes.

Table 4-10. RTR Operating System Process Utility IDs

UID	3B NAME	DESCRIPTION
0x004	/bootfiles/fmprc	file manager
0x005	/bootfiles/3bpmgr	3B process manager
0x008	/bootfiles/3bnub	the kernel
0x00a	/cft/shl/cmds/CFR/DUPLEXDISKS	configure duplex disks
0x00f	/prc/cdi	memory driver
0x010	/bin/getty	set terminal type, modes, speed, and line
0x011	/prc/unix	*UNIX†* system initialization process
0x012	/etc/login	sign on
0x013	/bin/sh	shell, the standard command
0x014	/etc/update	updates file system every 5 minutes
0x015	/etc/cron	task scheduler
0x016	/prc/pkillp	supervisor process to kill other processes
0x017	/prc/fda	FIFO driver
0x018	/etc/clrfs	constructs file system
0x019	/etc/clri	clear inodes
0x01a	/bin/df	report number of free disk blocks
0x01b	/etc/dgnnm	assigns diagnostic file name
0x01c	/etc/fsdb	file system debugger
0x01d	/etc/ichk	file system consistency check 0x01f:/etc/mknod:builds a special file
0x020	/mount	mounts file system
0x021	/bin/ps	report process status
0x022	/etc/udgnnm	generates the file set up by dgnnm
0x023	/etc/umount	unmounts file system
0x024	/etc/vcp	volume disk copy process
0x041	/prc/cdn	ECD manager
0x042	/diag/dgnc/mira	maintenance input request administrator

* UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Table 4-10. Operating System Process Utility IDs (Contd)

UID	3B NAME	DESCRIPTION
0x043	/cft/shl/cmds/STOP/DMQ	stop diagnostics
0x044	/cft/shl/cmds/ALW/DMQ	allow diagnostics
0x045	/cft/shl/cmds/DGN	diagnose hardware unit
0x046	/cft/shl/cmds1/EX	PDS interactive diagnostic
0x047	/cft/shl/cmds/INH/DMQ	inhibit diagnostic maintenance
0x048	/cft/shl/cmds/EX/LDPARM	PDS interactive diagnostic control command
0x049	/cft/shl/cmds/EX/LOOP	PDS interactive diagnostic control command
0x04a	/cft/shl/cmds/EX/PAUSE	PDS interactive diagnostic control command
0x04b	/cft/shl/cmds/RMV	remove hardware unit
0x04c	/cft/shl/cmds/RST	restore hardware unit
0x04d	/cft/shl/cmds/OP/DMQ	status of diagnostic maintenance
0x04e	/cft/shl/cmds/EX/STEP	PDS interactive diagnostic control command
0x04f	/cft/shl/cmds/EX/STOP	PDS interactive diagnostic control command
0x05f	/diag/dgnc/inhtimer	inhibit diagnostic timer
0x060	/diag/dgnc/ppdiamon	peripheral diagnostic monitor
0x062	/diag/dgnc/tlp	trouble location process
0x063	/diag/dgnc/dfdiag	DFC diagnostics control
0x064	/diag/dgnc/iodiag	I/O diagnostics control
0x065	/diag/dgnc/iormv	I/O diagnostics remove process
0x066	/diag/dgnc/dfrmv	remove process for DFC
0x067	/diag/dgnc/cudiagc	control unit diagnostics
0x06c	/diag/dgnc/dgntimer	diagnostic timer
0x080	/prc/olbexc	start off-line boot
0x081	/prc/olbswitch	start off-line boot switch
0x082	/prc/cdm	maintenance channel driver
0x083	/bootfiles/eih	error interrupt handler
0x084	/cft/misc/rmf	prints error messages and postmortem dumps
0x085	/bootfiles/pcpaud	CU audit
0x086	/bootfiles/inhadm	inhibit administration process
0x087	/etc/adp.af	automatic diagnostic process after fault process
0x088	/cft/shl/cmds/OP/CFGSTAT	output device status or configuration status
0x08a	/etc/fsmon	file system overflow monitor
0x08b	/prc/ularp	*UNIX* system level automatic restart process
0x08d	/etc/rex	routine exercisor
0x08f	/etc/adp.ab	automatic diagnostic process after boot branch

Table 4-10. RTR Operating System Process Utility IDs (Contd)

UID	3B NAME	DESCRIPTION
0x090	/cft/shl/cmds/ALW/REX	allow routine exerciser
0x091	/cft/shl/cmds/INH/REX	inhibit REX on a specified unit
0x092	/cft/shl/cmds/OP/REXINH	output REX inhibited units
0x093	/cft/shl/cmds/ALW/CONFLOG	allow configuration log
0x094	/cft/shl/cmds/INH/ERRCHK	inhibit errint, errsrc, hdwchk, and sftchk
0x095	/cft/shl/cmds/INH/CONFLOG	inhibit configuration log
0x096	/cft/shl/cmds/SW/CU	switch control units
0x097	/unixutil/cu/curstrmv	CU restore or remove process
0x098	/cft/shl/cmds/DUMP/CACHE	dump off-line cache into memory
0x099	/cft/shl/cmds/OP/MEMERRS	formatted memory error summary
0x0a1	/unixutil/disk/bootdiskchk	check if a disk is bootable
0x0c0	/prc/bdf	IOP driver process
0x0c1	/bootfiles/dkdrv	disk driver
0x0c2	/prc/scsd	SCSD driver
0x0c3	/cft/shl/cmds/LOAD/MHD/FIRMWARE	load firmware into SCSI MHD
0x0c4	/prc/s_update	disk restore
0x0c5	/cft/shl/cmds/CLR/FANALM	sends reset command to SCSDA
0x0c6	/cft/shl/cmds/LOAD/DFC/PUMP	download pumpcode to DFC RAM
0x0c7	/cft/shl/cmds/LOAD/MHD/DEFECT	load defect table
0x0c9	/cft/shl/cmds/CMPR/MHD	compare moving head disks
0x0ca	/prc/s_dskutil	disk compare
0x0cd	/cft/shl/cmds/OP/MHD/INFO	display moving head disk information
0x0ce	/cft/shl/cmds/INIT/MHD	format an MHD
0x0cf	/cft/shl/cmds/OP/DFCELOG	dump error log
0x0d0	/cft/shl/cmds/SET/IODRV	sets I/O driver options
0x00d1	/cft/shl/cmds/OP/IODRV	display active I/O driver options
0x0d2	/cft/shl/cmds/SW/PORTSW	port switch
0x0d3	/cft/shl/cmds/ALW/SCSD	allows SCSD points and signal
0x0d4	/cft/shl/cmds/INH/SCSD	inhibits SCSD function
0x0d5	/cft/shl/cmds/OP/SCSD	display SCSD points
0x0d6	/cft/shl/cmds/ORD/SCSD	set, clear, or flash points
0x0d7	/cft/shl/cmds/DUMP/MHD/VTOC	dumps volume table of contents
0x0d8	/cft/shl/cmds/DUMP/MHD/DEFECT	dumps defect tables
0x0d9	/cft/shl/cmds/DUMP/MHD/BLOCK	dumps a disk block
0x0da	/cft/dap/dkdip	display disk configuration and status
0x0db	/cft/shl/cmds/UPD/FLASH/PCFLASH	update flash RAM of PC

Table 4-10. RTR Operating System Process Utility IDs (Contd)

UID	3B NAME	DESCRIPTION
0x0dc	/cft/shl/cmds/STOP/DCI	terminates the dcidrv
0x0dd	/cft/shl/cmds/UPD/FLASH/DFCFLASH	update flash RAM of UN580 MHD
0x0df	/cft/shl/cmds/OP/IOP/INFO	display information about IOP subunits
0x0f1	/cft/shl/cmds/CLR/IOMEM	remove a file from I/O driver's cache
0x102	/usr/bin/perform	a parser for ucm
0x102	/cft/shl/cmds1/UPD	invokes a field update action
0x103	/prc/prchk	tests whether the target process can be reclaimed
0x104	/prc/dufr	overwrites memory image of non-killable process
0x105	/prc/kop	overwrites the tv segment
0x106	/usr/bin/idump	displays information of COFF and tracking problems
0x107	/prc/SUogen	creates a file to be used by oild and dufr
0x109	/prc/SUbldboot	controls creation boot image
0x10b	/prc/cdcmpr	compares text segments of disk and core image
0x111	/prc/gspovmon	overload monitor for GRASP
0x112	/prc/gspac	controls execution of GRASP
0x113	/cft/shl/cmds/OP/UTIL	lists GRASP breakpoints, status and trace
0x114	/prc/gspop	outputs GRASP messages
0x117	/prc/pldmon	updates maintenance times in all PMDB maintenance records
0x11d	/etc/mkdsk	make disk
0x11e	/bin/isgen	builds boot image on disk
0x11f	/usr/bin/browse	tool for examining database
0x121	/usr/bin/sdpcopy	copies database files
0x124	/prc/SUkopf	times for automatic backout using kop
0x125	/cft/shl/cmds/IN/REMOTE	monitors SCANS-2 file receive process
0x12b	/prc/bwmint	interface for field update session
0x12b	/prc/SUpupci	interface for field update session
0x12d	/prc/filerecv	receives and assembles transmitted files
0x12e	/cft/shl/cmds/IN/XFER	requests remote file transfers
0x131	/prc/SUrscans	controls SU file transfers
0x132	/usr/bin/pl_aux	data dictionary for the plant measurements database

Table 4-10. RTR Operating System Process Utility IDs (Contd)

UID	3B NAME	DESCRIPTION
0x133	/usr/bin/ibrowse	active process debugger
0x134	/usr/bin/lla_audits	audits ECD
0x162	/cft/shl/cmds/VFY/TAPE	invoke tape verification process
0x163	/audprc/pmsaud	audits plant measurements database
0x185	/cft/shl/cmds/OP/FNAME	audit command process
0x186	/prc/klmon	kernel monitoring process
0x187	/bootfiles/simprc	system integrity monitor
0x1c4	/prc/cdq	communicates with spy processes
0x188	/prc/suovprc	supervisor/user lockout monitor
0x190	/etc/sdlrtc	synchronous data link restore tool
0x191	/etc/sdlrtn	synchronous data link restore tool
0x192	/prc/fsaudit	file system audit
0x1a2	/audprc/ecdaud	ECD audits
0x1c5	/bin/cmpr	compares text segments of disk and core image
0x1c6	/prc/SUftrc	function trace
0x1c7	/prc/SUucm	update utilities controller
0x1c8	/prc/SUautomgr	software update manager
0x1c9	/prc/SUfilercv	software update file receive
0x1cb	/prc/SUgetty	software update getty
0x1cc	/prc/SUhlthchk	software update header checker
0x1cd	/prc/SUopedit	software update office profile editor
0x1ce	/prc/SUpctl	software update craft interface control loader
0x1cf	/prc/SUpgmgr	software update file receive
0x1d0	/prc/SUpswinit	software update getty
0x1d1	/prc/SUoild	creates new disk file
0x1d2	/prc/SUversion	display or update version files
0x1da	/prc/SUdisplay	displays specified update or BWM records
0x1db	/bin/compress	used to compress and uncompress files
0x1dc	/prc/SUexpand	process message to expand BWMs
0x1dd	/prc/SUedcud	CUD editor
0x200	/bin/errport	errport user process
0x201	/cft/bin/csdip	interface process between Sdlgshl and IODRV
0x202	/cft/dap/cia	critical indicators administrator
0x203	/cft/dap/dap	display administration process
0x204	/cft/dap/poker	DAP input process
0x206	/cft/rts	initialize RTS and receive messages

Table 4-10. RTR Operating System Process Utility IDs (Contd)

UID	3B NAME	DESCRIPTION
0x207	/bin/shlgetty	starts craft shells
0x208	/cft/bin/cftshl	craftshell
0x208	/cft/bin/cftshlA	craftshell (without initialization message)
0x209	/cft/spl/csop	craft spooler out process
0x20b	/cft/spl/sop	spooler output process
0x20d	/bin/cdgetty	starts poker
0x020f	/bin/splgetty	starts spooler
0x210	/cft/dap/inph	test process for DAP
0x211	/cft/dap/msgh	test process for DAP
0x212	/etc/ccdate	identify source version of code
0x214	/cft/shl/cmds/OP/LOG	prints log file entries
0x216	/cft/dap/starter	start reader on C/D input
0x217	/cft/shl/cmds1/TST	test command for pdshl
0x218	/prc/bdg	DAP driver
0x219	/cft/dap/fmctrl	runs 105 and 106 pages
0x21b	/bin/ciagetty	starts cia process
0x21d	/cft/misc/rpttime	time stamp for log files
0x21e	/cft/shl/cmds/CLR/IMCAT	clears imcatlog from core
0x220	/bin/dlggetty	starts dialog shell
0x221	/cft/shl/cmds/DLGAUTH	dialog authority file commands
0x222	/cft/shl/cmds/VFYAUTH	checks authority file
0x223	/cft/bin/Adlgshl	asynchronous dialog shell
0x224	/cft/bin/Sdlgshl	synchronous dialog shell
0x226	/cft/shl/cmds/CLK	sets or prints system clock
0x227	/cft/shl/cmds/UPD/OMDB	output messages database
0x228	/cft/shl/cmds/CLR/ACKDB	acknowledgements database
0x234	/cft/shl/cmds/OP/ABD	display status of Alternate Boot Disks
0x240	/bin/cat	lists and concatenates *UNIX* system files
0x246	/bin/chgrp	change group
0x247	/bin/chmod	change mode of file
0x248	/bin/chown	change owner of file
0x249	/bin/cmp	compare two files
0x24a	/bin/tr	translate input
0x24b	/bin/cpio	format of cpio archive
0x24c	/bin/cx	core image examiner
0x24d	/bin/crypt	generate encryption file
0x24e	/bin/date	print and set the date
0x24f	/bin/dd	convert and copy a file
0x250	/bin/diff	differential file comparator

Table 4-10. RTR Operating System Process Utility IDs (Contd)

UID	3B NAME	DESCRIPTION
0x251	/bin/dlsum	sum bytes in field mode
0x252	/bin/du	summarize disk usage
0x253	/bin/echo	repeat string
0x254	/bin/ed	line editor
0x255	/bin/env	set environment for command execution
0x256	/bin/expr	evaluate arguments as an expression
0x257	/bin/falloc	allocate space for an external file
0x258	/bin/fgrep	search a file for a pattern
0x259	/bin/find	find files
0x25a	/bin/fmove	move file into contiguous space
0x25b	/bin/fsize	prints size of files
0x25c	/bin/grep	search a file for a pattern
0x25e	/bin/id	print user, group, fair share group IDs and names
0x25f	/bin/kill	send a signal to a process or a group of processes
0x260	/bin/killp	kill user processes using a full pathname
0x263	/bin/line	read one line
0x264	/bin/ln	link files
0x265	/bin/logdir	get login directory
0x266	/bin/ls	list contents of directory
0x267	/bin/cp	copy files
0x267	/bin/mv	move a file
0x268	/bin/mail	send mail to users or read mail
0x269	/usr/lib/makekey	generates encryption key
0x26a	/bin/mesg	permit or deny messages
0x26b	/bin/mkdir	make a directory
0x26c	/usr/bin/mop	mount off-line partition
0x26d	/bin/newgrp	log in to a new group
0x26e	/bin/news	print news items
0x26f	/bin/nice	change priority of a process
0x270	/bin/nohup	run a command immune to hangups and quits
0x271	/bin/od	octal dump
0x272	/bin/passwd	user information file
0x273	/bin/pio	I/O to a traced process image
0x274	/bin/pr	print files
0x27a	/bin/pwd	print working directory name or path
0x27b	/bin/rm	remove files
0x27c	/bin/rmdir	remove directories
0x27e	/bin/run	run kernel processes

Table 4-10. RTR Operating System Process Utility IDs (Contd)

UID	3B NAME	DESCRIPTION
0x027f	/bin/sdiff	side-by-side difference program
0x280	/bin/sleep	suspend execution for interval
0x281	/bin/sort	sort and/or merge files
0x282	/bin/split	split a file into pieces
0x283	/bin/stat	get file status
0x284	/bin/stty	set the options for a terminal
0x285	/bin/su	change user ID
0x286	/bin/sum	print checksum and block count of a file
0x288	/bin/sync	update super-block
0x289	/bin/tail	deliver the last part of a file
0x28a	/bin/tee	pipe fitting
0x28b	/bin/time	get time
0x28c	/bin/touch	update access and modification times of a file
0x28d	/bin/tty	terminal device interface
0x28e	/bin/uname	print name of current system
0x28f	/bin/wc	word count
0x290	/bin/who	who is on the system
0x291	/bin/write	write on a terminal
0x292	/bin/sed	stream editor
0x293	/usr/bin/asa	interpret the asa control character
0x294	/prc/rcvryoff	prints PRM to extinguish recovery lamp
0x2c0	/cft/shl/cmds/COPY/ACTDISK	copies a file to an OOS disk
0x2c1	/prc/mntfs	mounts file systems in an OOS disk
0x2c2	/prc/supr/continue	system update continue handler
0x2c3	/usr/bin/PDSed	PDS editor
0x2c4	/cft/shl/cmds/ALW/FILESYS	file system maintenance
0x2c5	/prc/pmdbcopy	copies plant measurements database from core to disk
0x2c6	/cft/shl/cmds/OP/PMCR	invokes pmcrman
0x2c7	/prc/pmcrcol	updates common records
0x2c8	/prc/pmcrrep	generates PMS reports
0x2c9	/prc/pmcrman	process that controls execution of pmcrcol and pmcrrep
0x2ca	/prc/cpspdisk	copies a file to a spool disk
0x2cb	/prc/cpoosf	copies a file from an OOS disk
0x2cc	/cft/shl/cmds/COPY/CPSPDISK	invokes cpspdisk

Table 4-10. RTR Operating System Process Utility IDs (Contd)

UID	3B NAME	DESCRIPTION
0x2cd	/cft/shl/cmds/COPY/CPOOSF	invokes cpoosf
0x2ce	/cft/shl/cmds/CLR/PTN	invokes clearptn
0x2cf	/prc/clearptn	clears out a partition
0x2d0	/prc/chk_pmdb	sends check PMDB request to pmcrman
0x2d1	/cft/shl/cmds/COPY/BKTAPE	copy DAT tape to tape
0x2d5	/bin/urun	run user process
0x2d6	/prc/3btpwrt	writes disk image to tape in LDFT format
0x2d7	/cft/shl/cmds/COPY/BKDSK/ACK	invokes tpack
0x2d8	/cft/shl/cmds/STOP/BKDISK	stops the physical disk to tape writer
0x2d9	/cft/shl/cmds/COPY/BKDSK/START	invokes 3btpwrt
0x2da	/prc/tpack	acknowledge 3btpwrt that a tape is mounted
0x2db	/prc/tpstop	stop execution of 3btpwrt
0x2dc	/prc/supr/continue	restarts execution of a system update
0x2dc	/cft/shl/cmds/UPD/GEN/CONTINUE	invoke continue process
0x2dd	/prc/supr/applhook	application process used during system update
0x2de	/prc/supr/readlog	requests output of system update event log
0x2e0	/cft/shl/cmds/UPD/GEN/BACKOUT	invoke backout process
0x2e1	/cft/shl/cmds/UPD/GEN/COMMIT	invoke commit process
0x2e2	/cft/shl/cmds/UPD/GEN/ENTER	invoke enter process
0x2e3	/cft/shl/cmds/UPD/GEN/PROCEED	invoke proceed process
0x2e4	/cft/shl/cmds/OP/GEN/READLOG	invokes readlog
0x2e5	/cft/shl/cmds/UPD/GEN/RESTORE	invoke restore process
0x2e6	/cft/shl/cmds/STOP/GEN	invokes stop process
0x2e7	/prc/supr/stop	stops the system update command in process
0x2e8	/prc/supr/commit	overwrites old generic with new generic
0x2e9	/prc/supr/restore	restores the old generic to the system
0x2e9	/etc/mkstart	make disk acknowledgment program
0x2ea	/prc/supr/backout	backs out of new generic to old generic
0x2eb	/prc/supr/enter	system update process
0x2ec	/prc/supr/proceed	prepares system for booting from the new generic

Table 4-10. RTR Operating System Process Utility IDs (Contd)

UID	3B NAME	DESCRIPTION
0x2ed	/bin/prtcp	copies primary/backup partition
0x2ee	/prc/supr/suprint	invoke a system update process
0x2ef	/prc/supr/applproc	application process
0x2f0	/cft/shl/cmds/UPD/GEN/APPLPROC	invoke an application process
0x300	/prc/vfydiskecd	checks for RCV running with nreview
0x301	/usr/bin/loadf3b	creates a flatfile from a <> file
0x0302	/bin/sdfinfo	special device file information
0x303	/prc/psm	power switch monitor
0x304	bin/sdfrel	special device file release
0x305	/usr/bin/transgen	ECD evolution tool transgen
0x306	/usr/bin/treebld	ECD evolution tool treebld
0x307	/cft/shl/cmds/RCV/MENU	DB recent change menu
0x308	/usr/bin/evol	ECD evolution tool evol
0x30a	/usr/bin/rcvecd	recent change and verify ECD from *UNIX* system terminal
0x30b	/bin/ducb	display UCB in ECD
0x30f	/usr/bin/comparedb	compare databases
0x310	/usr/bin/fdiff	file differences
0x311	/usr/bin/keycmp	key compare
0x312	/usr/bin/keycomm	key common program
0x313	/usr/bin/keys	keys program
0x314	/usr/bin/newdb	create a new database
0x315	/usr/bin/printdb	print database
0x316	/usr/bin/printfrm	print forms
0x31e	/cft/shl/cmds/RCV/DMTECD	recent change and verify interface for ECD
0x31f	/usr/bin/createecd	creates skeleton of ECD database
0x320	/cft/shl/cmds/OP/ULARP	ularp request processor
0x322	/cft/shl/cmds/STOP/AUD	audit stop process
0x323	/cft/shl/cmds/ALW/AUD	allow audit
0x324	/cft/shl/cmds/AUD	begin audit process
0x325	/etc/siof	system integrity output formatter
0x326	/cft/shl/cmds/INH/AUD	audit inhibitor
0x327	/cft/shl/cmds/OP/AUD	audit report generator
0x32a	/cft/shl/cmds/CLR/EMERDMP	emergency dump
0x32b	/cft/shl/cmds/OP/EMERSTAT	emergency status
0x32c	/etc/op_stat	audit status process

Table 4-10. RTR Operating System Process Utility IDs (Contd)

UID	3B NAME	DESCRIPTION
0x32d	/cft/shl/cmds/OP/AUDERR	audit error process
0x32e	/cft/shl/cmds/INIT/ULARP	craft initialization to SIM
0x334	/cft/shl/cmds/VFY/FILE	verify file
0x360	/usr/bin/createsg	creates skeleton of SG database
0x361	/usr/bin/rcvsg	recent change and verify SG from *UNIX* system terminal
0x362	/usr/bin/rcvecdmcrt	recent change and verify ECD from MTTY
0x363	/usr/bin/rcvsgmcrt	recent change and verify SG from MTTY
0x364	/cft/shl/cmds/RCV/DMTSG	recent change and verify interface for SG
0x365	/usr/bin/vfydflt	perform default file verification via vfydflt
0x366	/usr/bin/vfydflt.p	perform default file verification via vfydflt
0x366	/usr/bin/iopadd.p	deletion of IOP
0x366	/usr/bin/iopdel.p	deletion of IOP
0x366	/usr/bin/links.p	review linkage orders
0x366	/usr/bin/mtadd.p	addition of maintenance terminal
0x366	/usr/bin/mtcadd.p	addition of maintenance terminal controller
0x366	/usr/bin/mtcdel.p	deletion of maintenance terminal controller
0x366	/usr/bin/mtdel.p	deletion of maintenance terminal controller
0x366	/usr/bin/sdladd.p	addition of SCANS distributor linkage
0x366	/usr/bin/sdlcadd.p	addition of SCANS distributor linkage controller
0x366	/usr/bin/sdlcdel.p	deletion of SCANS distributor linkage controller
0x366	/usr/bin/sdldel.p	deletion of SCANS distributor linkage
0x366	/usr/bin/slots.p	review slot assignments on UCB
0x366	/usr/bin/ttyadd.p	addition of tty
0x366	/usr/bin/ttycadd.p	addition of ttyc
0x366	/usr/bin/ttycdel.p	deletion of ttyc
0x366	/usr/bin/ttydel.p	deletion of tty
0x366	/usr/bin/vfydflt.p	perform vfydflt process
0x379	/bin/rmtgetty	remote dialog shell

idump Command

5

Contents

idump Command

5

Overview

The **idump** tool is primarily used for software update problem troubleshooting. However, it is also a very useful tool for general software troubleshooting because of two handy commands **a** and **p**. The **a** <symbol> command will provide information about the symbol including the starting address. It is much quicker than using the *sym* **ibrowse** option. It’s partner, **p** <address>, provides the information about the symbol that is associated with the address. This is the perfect command for tracking down strange addresses found in stack-back traces and various log file entries.

```
# idump /bootfiles/fmprc
IDUMP

     CURRENT FILE: /bootfiles/fmprc  Sat Apr 15 08:15:31 1995
F_PATCH F_AR32W F_LSYMS F_LNNO F_EXEC F_RELFLG
Magic   Nscns   Time/Date       Symptr         Nsyms       Opthdr  Flags
00551     134   0x2f8fc6f3     0x00027c32         4721       0x08ec  0x060f
:asmount

Symndx  Name       Value     Scnum    Type           Sclass  Numaux

[  298] smount    0x000230d0     1      ()INT          EXT       1
           s/u/e tag=0  fcn size=0x684  lptr=0x0  endx=308  tv=109
:p 0x230f0

Symndx  Name       Value     Scnum    Type           Sclass  Numaux

[  298] smount    0x000230d0     1      ()INT          EXT       1
           s/u/e tag=0  fcn size=0x684  lptr=0x0  endx=308  tv=109
:q
#
```

Figure 5-1. idump Example

```
< exc:envir:uproc,fn"/bin/sh",args("-c","echo ainode | idump /bootfiles/fmprc")
 PF
<
005372 70-01-05 15:06:33 sslcu6-m5
M 06 EXC ENVIR UPROC /bin/sh COMPLETED


     CURRENT FILE: /bootfiles/fmprc  Mon Jun 10 11:08:59 1996
  F_PATCH F_AR32W F_LSYMS F_LNNO F_EXEC F_RELFLG
  Magic    Nscns   Time/Date       Symptr     Nsyms    Opthdr  Flags
  00551      134   0x31bc489b    0x00027c32      2909    0x08ec 0x060f
  :
  Symndx      Name       Value  Scnum       Type       Sclass Numaux

  [ 2182]      inode
                 0x002e0000    78       []STRUCT       EXT   1
           s/u/e tag=0 array size=0x4c80  dimen=(204, 0, 0, 0)
  :
```

Figure 5-2. Non-Interactive Example

```
# echo "aKvt" | idump /bootfiles/3bsgen.kern|grep EXT |cut -d'x' -f2|cut -d' ' -f1
001c35cc
#
```

Figure 5-3. Example Extract Just the Kernel Address for Kvt

(Technique for use in **ibrowse** command shell scripts)

```
# idump /libc
IDUMP

      CURRENT FILE: /libc     Wed Apr 19 14:06:09 1995
F_PATCH F_AR32W F_LSYMS F_LNNO F_EXEC F_RELFLG
Magic   Nscns   Time/Date        Symptr          Nsyms       Opthdr  Flags
00550     11   0x2f955f21     0x0001164c         5620       0x07fc  0x060f
p 0x64964c

Symndx  Name       Value      Scnum    Type           Sclass  Numaux

[ 3279] %Rdwr     0x006495b4     1     ()INT          HIDDEN     1
            s/u/e tag=0  fcn size=0x17c  lptr=0x0  endx=3285  tv=541
:q
#
```

Figure 5-4. Tracking Down an Libc Address from a Core Dump

Description of Interactive Common Object Dumper (*idump*)

idump Command

NAME

idump - Interactive dump parts of a common object file

FORMAT

idump [-] [*file*...]

DESCRIPTION

idump allows a user to examine common object format files interactively. It is currently used for the following:

- *a.out* files (the output of *3bld*)
- *pfiles* and shared libraries (the output of *3bldp*)
- Minimal files (the output of *fextract*)
- Update files (the output of *ogen*)
- Simple object files (the output of *3bcc*).

idump permits the examination of multiple object files by specifying on the command line either a list of files or an archive that contains object file members.

The following is a brief description of all the **idump** commands:

<!command> Escape to the shell to execute a command.

a <symbol name> Dump all the symbol table entries with a specified symbol name.

b Reset the input base number to represent base 10 (decimal).

b <number> Reset the input base number to the base specified (16 for hex, 8 for octal, etc.).

c Close the current archive member and open the next member of an archive library.

d <storage class> Dump all the symbol table entries with the specified storage class.

e <type> Dump all the symbol table entries with the specified basic type.

f Dump the file header of the file you are currently in.

F Dump the .file symbol table entry for the current symbol table entry you are in.

F <filename> Dump the symbol table entry for the named source file.

g Open and dump the file header for the next file on the command line.

h Dump the section header for the last specified section.

h <section name> Dump the section header for the named section.

h <section number> Dump the section header for the specified section number.

h * Dump all of the section headers for the current file.

l Dump the line number entries for the last specified section.

l <function name> Dump the line number entries for the named function.

l <symbol index>
Dump the line number entries for the referenced symbol index.

l *
Dump all of the line number entries for all of the sections in the current file.

m
Print a list of **idump** commands.

m <idump command>
Print a description of the specified **idump** command.

n
Dump the next symbol table entry from the current position in the symbol table.

n <count>
Dump the symbol table entries for the next *<cont>* symbols.

n *
Dump the rest of the symbol entries from the current position in the symbol table.

o
Dump the entire optional header and patchlist.

o <a | o| p>
Dump part of the optional header: a = aout header, o = aout header and pfile header (library header), p = patchlist.

o *
Dump the entire optional header and patchlist.

p <program address>
Dump the symbol table entry for the function containing the given program address.

p <program address> <symbol name/index>
Dump all symbol table entries, starting with the function symbol that corresponds to the program address, up to and including the symbol named or indexed in the second argument.

q
Exit **idump**.

r
Dump the relocation entries for the last specified section.

r <section name>
Dump the relocation entries for the named section.

r <section index>
Dump the relocation entries for the referenced symbol index.

r *
Dump all of the relocation entries for all of the sections in the current file.

s
Dump the section contents for the last specified section.

s <section name>
Dump the section contents for the named section.

s <section number>
Dump the section contents for the named section within the specified range.

s <section name> <start addr> <end addr>
Dump the section contents for the named section within the specified range.

s <section number> <start addr><end addr>
Dump the section contents for the specified section number within the specified range.

t
Dump the symbol table entry for the next symbol.

t <symbol name>
Dump the symbol table entry for the named symbol.

t <symbol index>
Dump the symbol table entry for the referenced symbol index.

t <symbol name> <symbol name>
Dump the symbol table entries between the two named entries inclusive.

<symbol index> <symbol index>
Dump the symbol table entries between the two specified indexes inclusive.

t *
Dump all of the symbol table entries for the current file.

x
Exit **idump**.

*****
Dump the following for the current file:
1) Entire optional header and patchlist
2) All section headers
3) All section contents
4) All line number entries for all sections
5) All relocation entries for all sections
6) All symbol table entries.

The options of all the commands previously listed are described in the document named in "SEE ALSO" of this chapter.

idump dumps the information in an easily understood format.

An example of the **m** command is:

m g	open the next object file in argument list and dump the file header.

If the **m** is used alone, all the explanations for all the commands available in **idump** are listed.

HEADER FILES

none

FILES

none

SEE ALSO

Link Editor User's Manual (Revised)

DIAGNOSTICS

idump returns an exit code of zero. If an interrupt signal is received, **idump** returns to its command mode.

Diagnostics produced by **idump** are self-explanatory.

LIMITATIONS

none

LIBRARIES

none

Kernel Information

6

Contents

Kernel Information

6

OST Service Routines

Table 8-1 lists operating system trap (OST) service routines provided for kernel and supervisor processes. The routine names are the names used internally by the kernel and special processes and do not represent the client's interface. Internally, an OST service is called by issuing an IS25 OST instruction which appears in the text as x'd9nn where 'nn' is the OST number in hexadecimal.

Table 6-1. OST Service Routines

Routine	OST No.	Proc Type	Parameters	Description
addseg	x'01	sup	segnum flag	add a segment
adduser	x'02	sup	pnum	increment process user count
adopt	x'51	sup	pnum	assume parenthood of process
alockseg	x'03	sup	segnum	lock a segment and mark it as altered
alocmsg	x'01	kp	nbytes owner	allocate a message buffer
alocseg	x'04	sup	segnum size partition segname	allocate a segment

Table 6-1. OST Service Routines (Contd)

Routine	OST No.	Proc Type	Parameters	Description
atchchan	x’2a	kp	chand	attach a channel to an interrupt source bit
atchintr	x’02	kp	pnum ivect x’entryadd psw ident	attach to an interrupt
bkpt_ipm	x’23	kp	segid offset nbytes newcode savecode uf	insert a breakpoint
chgattrib	x’4b	sup	flag priority timeslice pclass1 pclass2 pname	change process attributes
chgpcb	x’52	sup	field value	change pcb value
clrevent	x’09	sup	eflags	clear event flags
clrname	x’4a	sup	segname	remove the name from a segment
conport	x’0c	sup	portnum	attach process to port
copyseg	x’0a	sup	segnum newid msident rtcnt	copy a segment
crb	x’06	sup	psync flag	conditional roadblock
dequeuem	x’03	kp	pnum owner	dequeue a message
detport	x’0d	sup	portnum	detach process from port
dctreset	x’18	sup	state pnum	set state in det of process

Table 6-1. OST Service Routines (Contd)

Routine	OST No.	Proc Type	Parameters	Description
dionotify	x’4e	kp	pnum flag	set flag to notify process when dlm state changes and returns current system DIOP status
disabintr	x’33	kp	pnum ivect	detach from an interrupt
dqlimit	x’1d	kp	pnum ltype utype owner	dequeue a message of a given range of types
dqtype	x’04	kp	pnum type owner	dequeue a msg type
dropseg	x’0f	sup	segnum	remove segment from virtual address space
dschmask	x’2b	kp	-none-	write the duel serial x’channel mask on offline side
dskduplex	x’45	kp	-none-	allow dlm essential processes to swap
dsksimplex	x’44	kp	-none-	lock dlm essential processes incore
dtchchan	x’48	kp	chand	detach channel from interrupt
dtchintr	x’05	kp	pnum ivect	detach from an interrupt
enabintr	x’07	kp	pnum ivect	enable a channel interrupt
enable_ev	x’34	kp	pnum eflags	enable an event(s)
enevent	x’10	sup	eflags	enable asynchronous entry on event(s)
ep_attach	x’43	kp	segindx pnum	get buffer into a first-in-first-out (FIFO) driver
err_rpt	x’15	sup	string size	log an error message
evclass	x’2d x’57	kp sup	class events	send an event(s) to all processes of a given class

Table 6-1. OST Service Routines (Contd)

Routine	OST No.	Proc Type	Parameters	Description
event	x’11	sup	pnum eflags	send an event(s) to a process
execute	x’12	sup	sp pcbindx stackindx pevp class	execute a new supervisor
fltclass	x’30	kp	class fcode	fault all processes of a given class
freemsg	x’06	kp	msgptr owner	free a message buffer
freeseg	x’13	sup	segnum mode	remove segment from segment list
fupatch	x’38 x’46	kp sup	action patch	manipulate the patch count
getclass	x’2e x’21	kp sup	pnum	get the class of a process
getime	x’08 x’16	kp sup	-none-	get the clock time
getmsg	x’17	sup	rcvbuf	dequeue a message
getnpas	x’49 x’53	kp sup	&word	get the number of pas segments
getpnum	x’4e	sup	utilid	return pnum matching utility ID
gettype	x’19	sup	rcvbuf	dequeue a message of x’a specific type
growseg	x’1a	sup	segnum nbytes oszptr	change the size of a segment
idlevent	x’3b	kp	pnum eflags	send event(s) to a process if system idles
inhibit	x’1b	sup	-none-	enter supervisor critical region
inhibitdv	x’3c	kp	ivect	inhibit a device from interrupting the system
iolock	x’09	kp	segid	lock a segment for input/output (I/O)
iomap	x’0a	kp	segid offset x’count	map segid to virtual address

Table 6-1. OST Service Routines (Contd)

Routine	OST No.	Proc Type	Parameters	Description
ioque	x’1c	sup	sndbuf	send an I/O message
ioqueuem	x’0b	kp	msgbuf owner	send a message to I/O device driver
jobchg	x’1d	sup	-none-	relinquish remainder of time slice
kconport	x’20	kp	portnum pnum	attach a process to a port
kdetport	x’21	kp	portnum pnum	detach a process from a port
kdlmnkill	x’47	kp	pnum flag	mark kernel process as dlm essential
kmsgwflt	x’1e	kp	msgptr fcode owner	queue a message and send a fault
kpagemap	x’31	kp	segnum fpage 1page permissions	change segment access permissions page by page
kportid	x’22	kp	portnum	get the pnum on a port
kpstart	x’08	sup	ex1v1 channel segnum pident class flag	start a kernel process
krfseg	x’26	kp	segnum	remove segment from address space
krmvseg	x’40	kp	pnum segnum	remove a segment at boot time only
ksegadd	x’3f	kp	pnum segnum segindx size permissions segname	add a segment at boot time only
ksegsize	x’3e	kp	segid	get a segment size
ksegublk	x’3d	kp	segid	remove blocked state x’from a segment
kshartxt	x’88	kp	segid syndx segnum	add segment to address space with execute permission

Table 6-1. OST Service Routines (Contd)

Routine	OST No.	Proc Type	Parameters	Description
ksfseg	x'27	kp	segid segndx segnum	add segment to address space
kvt_kp	x'36	kp	-none-	get pointer to the kernel's vector table
kvt_sup	x'45	sup	-none-	get pointer to the kernel's vector table
lbolt	x'07	sup	-none-	send wakeup event at lightning bolt interval
lockid	x'1e	sup	segid	increment lock count of (already locked) segment
lockseg	x'1f	sup	segnum	lock a segment
mask_ev	x'35	kp	pnum emask	disable an event(s)
maxintvl	x'4d	kp	pnum	establish a maximum
smaxintvl	x'56	sup	interval omsg prms failure	time-out interval
messink	x'0c	kp	msgptr owner	return an acknowledgment
mgetim	x'20	sup	rcvbuf	dequeue a message of a type within a range
move_ut	x'25	kp	addr	get message trace data
oldmsg	x'4c	kp	flags owner	get chain of dequeued messages
openseg	x'24	sup	segnum segid segflags mode	add a segment to caller's segment list
overload	x'37	kp	clparm1 clparm2 class pnum	report on or clear message buffer overloads
permit	x'25	sup	-none-	drop out of the supervisor critical region
pfork1	x'e	sup	segnum	fork a process first step
pfork2	x'23	sup	segnum	fork a process second step

Table 6-1. OST Service Routines (Contd)

Routine	OST No.	Proc Type	Parameters	Description
phase	x’32	kp	plevel alevel panic code utilid	phase the system
phasewop	x’42	kp	plevel alevel panel code utilid options	phase system with options
portid	x’3e	sup	portnum	get the pnum on a port
prctype	x’41 x’4c	kp sup	pnum	get a process type
prtimer	x’0d	kp	pnum intvl	request a repetitive time-out event
psignal	x’0e	kp	channel eflags	send event(s) to all processes on a control channel
psleep	x’0f	kp	pnum pattern	set a supervisor sleep bit pattern
pstart	x’26	sup	channel segnum iprior parent flag	start a supervisor process
pswap	x’27	sup	-none-	make the caller swappable
ptimer	x’10	kp	pnum intvl	request a single time-out event
punswap	x’28	sup	-none-	make the caller nonswappable
pwakeup	x’12	kp	pattern	send a wakeup event to every supervisor with a given sleep pattern
queuem2	x’13	kp	msgptr owner	queue a message
queuemn2	x’14	kp	msgptr owner	queue a message with no acknowledgment expected
rcevent	x’39	kp	pnum	read and clear event flags
riteback	x’15	kp	segid	set altered bit on a segment to force swapout
rmovseg	x’2b	sup	segm	remove a segment from x’callers address space

Table 6-1. OST Service Routines (Contd)

Routine	OST No.	Proc Type	Parameters	Description
rpaddress	x'24	kp	segid offset	return physical address
rtbp	x'29	kp	save opcode	return from a breakpoint
rtiflt	x'1f	kp	fcode	fault the last interrupted process
rtnint	x'16	kp	svstate	return to an interrupted state from a fault routine
rtoutset	x'2d	sup	intvl	request a repetitive time-out
sdionotify	x'58	sup	pnum flag	set flag to notify process when dlm state changes and returns current system DIOP status
sdlmnkill	x'50	sup	pnum	mark supervisor process as dlm essential
segname	x'17	kp	segid segname	get the name of a segment
send_err	x'28	kp	string size	lop an error message
sendcpmsg	x'2f	sup	sndbuf	send a capability message
sendevent	x'18	kp	pnum eflags	send an event(s) to a process
sendfault	x'30	sup	pnum fcode	send a fault
sendg*	x'7F	sup	sndbuf	queue a message and set group id flag
sendmsg	x'31	sup	sndbuf	queue a message
sendport	x'32	sup	sndbuf	queue a message to the process on a port
setclass	x'2f x'22	kp sup	pnum class	set the class of a process
setewait	x'33	sup	eflag opt	set condition for later conditional roadblock
setime	x'1a x'34	kp sup	time	set the clock time
setmap	x'35	sup	segnum access segndx	set control info on a segment

* Release 1 through Release 6.3 only.

Table 6-1. OST Service Routines (Contd)

Routine	OST No.	Proc Type	Parameters	Description
setprior	x’36	sup	prty	change the caller’s initial priority
settflag	x’4f	sup	pnum flag	set termination flag in process
shrfid	x’2a	sup	segid segndxu segnumu	share segment by segid
shrfseg	x’29	sup	pnum segndxf segndxu segnumu segidptr	share segment by virtual address within another process
sizeseg	x’38	sup	segnum	get the size of a segment
sleep	x’39	sup	pattern	set the caller’s sleep pattern
smsgwflt	x’14	sup	sndbuf fcode	send a message with a fault
sendfault	x’19	kp	pnum fcode	send a fault to a process
smaxintvl	x’56	sup	pnum interval omsg prms failure	establish a maximum time-out interval
sndfrom	x’3a	sup	sndbuf	forward a message
spacaloc	x’3b	sup	segnum	add a zeroed segment
srti	x’2c	sup	evmask svstate	return from an interrupt
ssegname	x’2e	sup	segid segname	get a segment name
sswap	x’3c	sup	segnum	make a segment swappable
sunswap	x’3d	sup	segnum	make a segment nonswappable
susppid	x’6b	kp	spid tpid	suspend the tpid by spid
suspuid	x’6a	kp	pid uid	suspend all process with uid by pid
suspclass	x’6e	kp	pid class	suspend all processes with class by pid

Table 6-1. OST Service Routines (Contd)

	OST	Proc		
Routine	**No.**	**Type**	**Parameters**	**Description**
suspusers	x’70	kp	pid essflag	suspend eligible user processes
sysdlm	x’46	kp	-none	enter full disk limp mode
termclass	x’2c	kp	class	terminate all processes of a given class
termutil	x’3a x’47	kp sup	utilid	terminate all processes with a specific utility ID
timleft	x’1b	kp	pnum	get time remaining until time-out is scheduled
toff_idlet	x’56	kp	percent	change turn off value for idle time to percent
top_pid	x’11	kp	-none-	get the pnum of the last interrupted process
toutset	x’3f	sup	intvl	request a single time-out event
uid2pid	x’8c	kp	uid buf size start_dct	convert a utility ID to a list of process IDs
ulockid	x’40	sup	sepid	decrement lock count on a segment
ulockseg	x’41	sup	segnum	decrement lock count on a segment
unblkseg	x’42	sup	segnum	remove blocked state from a segment
uniolock	x’lc	kp	segid	unlock a segment
uplockseg	x’0b	sup	segid	decrement the plock count on a segment
utilset	x’49	sup	utilid	modify the caller’s utility ID
wakeup	x’43	sup	pattern	send wakeup event to all processes on a pattern
writeseg	x’44	sup	segnum	mark segment altered to force a swapout

Kernel Address Space Segments

The following list defines the segments in the kernel's address space. The segment numbers and kernel virtual addresses associated with them may be found in va.h.

DCT
The dispatch control table (DCT) segment contains all information required for the kernel to schedule kernel and special processes and also the information required for the scheduler to service the supervisor and user processes. This segment consists of an array of DCT entries (DCTE) which are indexed by the low-order half-word of the process number and into which the dispq[ ] array points for each execution level. The number of available DCTEs is determined as an sgen parameter.

DCTEXT
The DCTEXT segment is an extension of the DCTE. Since the DCT is searched so frequently, the size of each entry must be kept on a power of two boundaries to ensure efficient addressing computation. This segment was added because the size of a DCTE would have to be doubled to add new fields and would significantly increase the memory requirements of the table.

ECDDATA
The ECDDATA segment is used by the kernel audit special process.

ECDLDATA
The ECDLDATA segment is used by the kernel audit special process.

ECDTEXT
The ECDTEXT segment is used by the kernel audit special process.

INTMEM
The INTMEM (interupt stack) segment is an interrupt stack that is used to save the registers of the currently running task when an interrupt occurs. Each element in the interrupt stack is an instat structure which has slots for the pa, psw, sbr and all general registers. A pointer to the top of the interrupt stack is located in firmware register 9 and the data in this segment is actually only found in cache.

KDATA
The KDATA segment contains all externally declared variables defined for the kernel and the special processes. See Section 8.3 of this document for a description of many of these fields.

KMSG
The KMSG (kernel message) segment is used to hold all messages sent between processes in the system. This segment is preformatted into 64 byte blocks which are assigned to users as requested in 1 to 7 block contiguous areas. All kernel processes have direct access to this segment but must use the appropriate OST handlers for allocation and queuing of

the messages. All supervisors have their messages copied into and out of this area via the appropriate kernel OST handlers.

KPATCH — The KPATCH segment will be used for patching the kernel and/or special processes via field update.

KPSHIFT — The KPSHIFT segment is used by the kernel and all special processes to address segments of their choosing. Normally this segment is used to address kernel process PCB segments but may be used for other segments such as supervisor PCBs (scheduler and memory manager) and text (utility manager breakpointing). The only restriction upon use is that the user must be in the critical region (level 15) during access.

KSHIFT0 — The KSHIFTO segment is a shift window used by the level 2 special processes to address segments of their choosing. Known users include the capability manager (supervisor PCBs), and the scheduler (supervisor PCBs).

KSHIFT1 — The KSHIFTI segment is used as a shift window during boot time; with it, *kboot* manipulates a PCB for purposes of creating the supervisor boot processes.

KSHIFTU0 — The KSHIFTUO segment is analogous to KSHIFT0. Provided for use with extended main memory module 1 segments.

KSHIFTU1 — The KSHIFTU1 segment is analogous to KSHIFT1. Provided for use with extended main memory module 1 segments.

KSTACK — The KSTACK segment is used as the ’C’ stack for the kernel, special processes, and most kernel processes. The size of the stack is an sgen parameter.

KTEXT — The KTEXT segment contains the executable code for the kernel and all special processes. Additionally some of the data areas required for *kboot* are in this segment.

MSGEXT — The MSGEXT (message extender) segment is used by the kernel to maintain the message queues. Unlike the message segment, this segment is not shared with other processes to maintain the integrity of the queues.

NPCB — The NPCB segment is a shift window used to address the segment defining the loading supervisor’s PCB segment. As the loading supervisor changes, the actual segment pointed to at this virtual address will change. See the memory manager documentation for further information.

PDT — The PDT segment contains the page descriptor table which has one element, a pde, for each physical page in the system. Each pde specifies whether or not the page is free and, if so, a pointer to the next free page; if not, there is a pointer to the sde which has the page allocated to it. The size of the PDT segment is dependent upon the size of main memory.

PDT2

PDT2 applies only to *UNIX*® Real-Time Reliable (RTR) operating system Release 6.9 and later. This is a continuation of the PDT segment into a second contiguous segment if more than 32K page descriptor table entries are required.

PDT3,PDT4

The PDT3 and PDT4 segments are reserved for PDT growth up to 256M.

PGT

The PGT segment contains the page tables (pgt) for each memory resident segment in the system and is pointed to by a sge (somewhere). The sge pointing to the pgt is dependent upon whose segment this is. Each PGT segment consists of a series of words, each of which is a pge which, when in use, defines one page within the segment. Each PGT is 256 bytes (64 words) long. The PGT is more fully documented under the memory manager.

PGT2

The PGT2 segment is a continuation of the PGT segment into a second contiguous segment if more than 512 pgt tables are required.

PGT3

The PGT3 segment is a continuation of the PGT segment into a third contiguous segment if more than 1,024 pgt tables are required.

PGT4

The PGT4 segment is a continuation of the PGT segment into a fourth contiguous segment if more than 1,536 pgt tables are required.

PGT5-8

The PGT5-8 segments are reserved for PGT growth up to 256M.

PORTS

The PORTS segment is an array of fullwords, each of which contains either the number of the process attached to the port number which indexes this word, or null if unattached. The number of words in the segment is determined as an sgen parameter.

SDT

The SDT segment contains the segment descriptor table which has one element: an sde for each possible segment in the system. An sde is allocated when a segment is created and contains such information as the page table address, the number of users, size, lock counts, and swap address. The size of the SDT segment is determined as an sgen parameter.

SPCB

The SPCB segment is a shift window used to address the segment defining the current PCB segment of the supervisor. As the current supervisor changes, the actual segment pointed to at this virtual address will change.

* UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

TVS — The TVS segment contains the transfer vectors used by the kernel and all of the special processes.

Kernel Address Space Data Areas

The following list defines many of the externally declared data areas in the kernel's address space.

Applev[ ] — This table defines the maximum application phase levels for each of the *UNIX* RTR operating system phase levels. The table is populated at boot time from sysgen input.

boot_mask[ ] — This table is used at boot time to initialize the interrupt masks so that no 'attachable' interrupts are allowed.

chimsk[ ] — A half-word array of channel masks which define the devices able to interrupt the system. This array is modified when interrupts are enabled or disabled.

chsrmsk[ ] — A half-word array of channel service request masks which define the devices able to interrupt the system. This array is modified when interrupts are enabled or disabled.

#chtoadr[ ]" — An initialized byte array mapping channel numbers to channel addresses.

cihtbl[ ] — This is an array of 20 cihe structures, one for each possible channel. Each entry identifies the channel type and has 16 pairs (one pair per possible device) of pointers to atchtbl entries, one for interrupts, and one for service request interrupts. These pointers are initially set to null and are filled in as processes attach to channel/device interrupts.

clientdctp[ ] — The kernel process to kernel process trap metering chain.

dctmem — The starting address of the first DCTE in the DCT segment; this value is defined by 'va.h' and is a separate segment in the kernel's address space.

dctnum — The total number of DCTEs located at dctmem.

devname — Passed via register from *bigboot*, this field contains the device and channel codes required by *kboot* in order to access the DMA.

dispq[ ] — An array of pointers to the dispatching queues of each execution level (note that entry 0 is null). The DCTEs at each level are chained together.

dvtoadr[ ] — An initialized byte array mapping device numbers to device addresses.

Dxtmem — A pointer to the start of the dct extension area.

ecdd_segid[ ] — The segment-id(s) of the ECD database. These values are passed to the process manager for assignment to kernel processes.

ecdl_segid — The virtual address (in the kernel) of the sdt entry for the ecd lla data segment. This value is passed to the process manager for assignment to kernel processes.

Ecdpath[ ] — The pathname under which *kboot* expects to find the ECD public library within the root file system.

ecdp_segid — The virtual address (in the kernel) of the sdt entry for the ecd patch segment. This value is passed to the process manager for assignment to kernel processes.

ecdt_segid — The virtual address (in the kernel) of the sdt entry for the ecd text segment. This value is passed to the process manager for assignment to kernel processes.

Free_blks — The running count of the number of free message blocks.

Freedcts — A running count of the number of free dct slots.

Idlepnum — The process number which receives an event when the system’s idle loop is reached.

Idlesim — If nonnull, the system integrity monitor (SIM) will receive this event when the system’s idle loop is reached rather than the process specified in ‘‘Idlepnum’’.

Idlevent — The event sent to ‘‘Idlepnum‘‘.

Ikbsegid — The segment-id of *kboot’s* initial segment table and page tables. After *kboot* completes, this segment is released back to the system.

imem — This structure is always pointed to by firmware register 8 (SYSBASE) and contains in contiguous order the 16 interrupt masks used for the execution levels, 36 interrupt transfer vectors for handling the various interrupt sources, and finally the instat structure found in the interrupt stack as its initial entry.

initmskt[ ] — This table is used to hold interrupt mask patterns set by the atchintr OST. When an enabintr OST is issued, before the real interrupt mask is modified, this table is checked, thus ensuring that an atchintr was issued. If the system’s fault entry is entered and the interrupt masks get reinitialized, this allows the attached processes to simply re enabintr their interrupts.

iptab — This table (which is defined in kboot.c) identifies the system processes and is used to initialize DCTEs for them. From the memory management viewpoint, each special process represents a subset of the kernel’s address space (text and data).

iptpnum The process number of the last kernel process trapped by a supervisor.

ishtbl[ ] This is an array of 32 ishe structures, one for each interrupt source bit, which identifies the interrupt source type and either the atchtbl entry attached to it or a channel mask which identifies the cihtbl entry. Only those interrupt sources denoted as ‘attachable’ will ever have valid entries.

kmsg_segid The virtual address (in the kernel) of the sdt entry for the kernel message segment. This value is passed to the process manager for assignment to kernel processes.

kost[ ] A count of the requests for each of the kernel process OST services and the amount of processing time spent in kernel processes.

kostab[ ] An initialized array defining the entry point and number of arguments for each kernel process OST service routine.

KPost[ ] This array contains an element for each execution level and the values represent the kernel process OST number currently being processed at each level. For each level that is not in an OST routine, the value will be 0xff. This array is used to determine which central processing unit (CPU) utilization counts are to be incremented.

KPROCESS[ ] This array (fullwords) contains the process numbers of the kernel process last dispatched at each execution level. This array is used to determine which CPU utilization counts are to be incremented.

kstk_segid The virtual address (in the kernel) of the sdt entry for the kernel stack. This value is passed to the process manager for assignment to kernel processes.

ktime The amount of processing time spent in the kernel.

Kvt A control block which provides a level of addressability to kernel data areas for supervisor and kernel processes. The address of the Kvt is passed via OST call and is an address in the kernel’s address space. The Kvt in turn contains pointers to other fields in the kernel (also a number of counters).

lbcnt The number of processes on the lightning bolt chain. These processes will receive an E_WAKEUP event at the fifth timer interrupt.

lb_head Pointer to the dct entry of the first process on the lightning bolt chain.

msfree Pointer to the first known free message block in the kernel’s message memory. This value may not be valid; it is used as a starting point for searching for free message blocks when allocation is to be done.

Msgelast A pointer to the last message buffer extender block.

Msgemem A pointer to the first message buffer extender block.

Msgestrt A pointer to the first normal priority, message buffer extender block. All prior message blocks are reserved for one block, high priority, operating system-initiated messages.

Msglast A pointer to the last message block.

msgmem A pointer to the starting address of message memory; this value is defined by ’va.h’ and points to a separate segment in the kernel’s address space.

msgnum An integer value that specifies the number of message blocks in the kernel’s address space. An sgen parameter.

msgseq A sequence number assigned to each message at allocation time, this number extends from 1 to 255.

Npde An integer value that specifies the number of pde blocks in the kernel’s address space. An sgen parameter.

Npgt An integer value that specifies the number of pgt blocks in the kernel’s address space. An sgen parameter.

Npir A running count of the total pir interrupts.

Nport The number of ports allowed in the system.

Nsde An integer value that specifies the number of sde blocks in the kernel’s address space. An sgen parameter.

ntcnt A running count of the interrupts that have occurred.

ov_head Pointer to the first DCTE which has a single time-out event pending for which the interval causes a wraparound on the ’tod’ counter. All such DCTEs are chained together in remaining interval increasing sequence (see also to_head listing).

Ovlddct If set to one, this character shows the system has exhausted all of its dct slots and SIM has been faulted.

Ovldm90 If set to one, this character shows the system has exhausted 90 percent of its message buffers and SIM has been faulted. The indicator is reset to 0 when 50% of the message buffers are free.

Ovldmsg If set to one, this character shows the system has exhausted 70 percent of its message buffers and SIM has been faulted. The indicator is reset to 0 when 50 percent of the message buffers are free.

Pa_pgt The physical address of the PGT segment.

Pa_plibseg The physical address of the public library mapping segment used by *kboot*. This segment is released back to the system at the conclusion of *kboot*.

PAS_ID[ ] The virtual address (kernel’s) of the sdt entries for the protected application segments.

Pa_vtoc The physical address of an incore copy of the vtoc for the root partition.

Pdefst A pointer to the starting address of the first pde in the PDT segment; this value is defined by ’va.h’ and points to a separate segment in the kernel’s address space.

Pgtfst A pointer to the starting address of the first pgt in the PGT segment; this value is defined by ’va.h’ and points to a separate segment in the kernel’s address space.

pircnt[ ] Running counts of the interrupts at each pir level.

pl1d_segid[ ] The segment-ids of the plant measurements database segments. These values are passed to the process manager for assignment to other processes.

pl1l_segid The segment-id of the plant measurements public library data segment.

pl1p_segid The segment-id of the plant measurements public library patch segment.

pl1t_segid The segment-id of the plant measurements public library test segment.

Plibpmsk A mask used by *kboot* to determine which public library segments are to be plocked and which are to be assigned swap space.

Plmapseg The segment-id of the public library mapping segment used by *kboot*.

Plmpath[ ] The pathname of the plant measurements public library which *kboot* uses to locate the library within the root file system.

Port The starting address of port memory; this value is defined by ’va.h’ and is a separate segment in the kernel’s address space.

rov_head Pointer to the first DCTE which has a repetitive time-out event pending for which the interval causes a wraparound on the ’tod’ counter. All such DCTEs are chained together (see also rto_head listing).

rparmv An rparm structure used as input to a write physical into low core of the last phase level.

rto_head Pointer to the first DCTE which has a repetitive time-out event pending. All such DCTEs are chained together (see also rov_head listing).

sdefst A pointer to the starting address of the first sde in the SDT segment; this value is defined by ’va.h’ and points to a separate segment in the kernel’s address space.

sostab[ ] An initialized array defining the entry point and number of arguments for each supervisor OST service routine.

stime The amount of processing time spent in supervisor processes.

supost[ ] Running counts of the supervisor OST service requests.

SUPost Contains the supervisor OST number that currently is being processed. If the system is not in a supervisor OST call, then this field will contain 0xff (or, rarely, a 0xfe). If the current supervisor has trapped to a kernel process, then the value will be 0xfe. This field is used to determine which CPU utilization counts are to be incremented.

to_head Pointer to the first DCTE which has a single time-out event pending. All such DCTEs are chained together in remaining interval, increasing sequence (see also ov_head listing).

Usrdct Pointer to the DCTE of the current supervisor/user process.

utchtbl[ ] This is an array of active structures which fully identify those processes which have attached-to interrupts. The array is 40 elements long, and initially, all elements are marked as ’free’.

utime The amount of processing time spent in user processes.

Release 1 Hexadecimal Offset Charts

7

Contents

Release 1 Hexadecimal Offset Charts

7

Kernel Address Space

This section lists the control block structures in the kernel address space. The lists name each field and give the hexadecimal offset to the field from the beginning of the structure.

Structure: ative - length: 0x1c

Source: os/kern/ative.h

Location: Found in the kernel’s data segment in an array called "atchtbl[ ]". Address is variable from load to load.

Use: Maintains the necessary information for the kernel to dispatch processes attached to interrupts.

+0	at_prc	attached process number
+4	*at_ent()	attached process entry point
+8	at_psw	psw value for interrupt
+c	at_ident	interrupt ID
+10	at_sbr	segment base register value
+14	at_is	interrupt source
+15	at_ch	I/O channel
+16	at_dv	I/O device
+17	at_dummy	unused at this time
+18	at_status	status of the attach point x’00000001’ - AT_FREE x’00000010’ - AT_BUSY

Structure: dcte - length: 0x80

Source: head/dcte.h

Location: In the kernel’s address space at 0x140000 (may vary per head/va.h).

Use: Is the central source of process related information in the kernel and special processes.

+0	d_flag	flag word x’00000001’ - DF_FAIL x’00000002’ - DF_LBOLT x’00000004’ - DF_PROFIL x’00000008’ - DF_RB x’00000010’ - DF_JC x’00000020’ - DF_TOUT x’00000040’ - DF_LOAD x’00000080’ - DF_READY x’00000100’ - DF_RLIST x’00000200’ - DF_SLEEP x’00000400’ - DF_REMOV x’00000800’ - DF_SWAP x’00001000’ - DF_NOSWAP x’00002000’ - DF_KPRC x’00004000’ - DF_SYS x’00008000’ - DF_RTOR x’00010000’ - DF_STOR x’00020000’ - DF_STATIC x’00040000’ - DF_TMPNR x’00080000’ - DF_NOTERM x’00100000’ - DF_RUN x’00200000’ - DF_NOFIT x’00400000’ - DF_MSGHOG x’00800000’ - DF_DLMESSNTL x’01000000’ - DF_DLMNONSWP x’02000000’ - DF_UNXTERM x’04000000’ - DF_FLTMSG x’08000000’ - DF_MAXINTVL x’10000000’ - DF_DIOCHG

+4	*d_link	ptr to next dcte on same execution level
+8	*d_lblk	ptr to l_bolt chain
+c	*d_stmlk	ptr to single time-out chain
+10	*d_rtmlk	ptr to repetitive time-out chain
+14	d_rtime	time interval for repetitive time-out
+18	*d_msg	ptr to 1st message (extender block) on queue
+1c	*d_msgend	ptr to last message (extender block) on queue
+20	d_stout	real-time value (msec) for single time-out
+24	d_rtout	real-time value (msec) for repetitive time-out
+28	d_evflag	event flags
		x’00000001’ - E_USR16
		x’00000002’ - E_USR15
		x’00000004’ - E_USR14
		x’00000008’ - E_USR13
		x’00000010’ - E_USR12
		x’00000020’ - E_USR11
		x’00000040’ - E_USR10
		x’00000080’ - E_USR9
		x’00000100’ - E_USR8
		x’00000200’ - E_USR7
		x’00000400’ - E_USR6
		x’00000800’ - E_USR5
		x’00001000’ - E_USR4
		x’00002000’ - E_USR3
		x’00004000’ - E_USR2
		x’00008000’ - E_USR1
		x’00010000’ - E_SYS16
		x’00020000’ - E_SYS15
		x’00040000’ - E_SYS14
		x’00080000’ - E_SYS13
		x’00100000’ - E_SYS12
		x’00200000’ - E_SYS11
		x’00400000’ - E_UTIL
		x’00800000’ - E_RTIMEOUT
		x’01000000’ - E_INIT
		x’02000000’ - E_ABORT
		x’04000000’ - E_QIT
		x’08000000’ - E_INT
		x’10000000’ - E_HUP
		x’20000000’ - E_MSG
		x’40000000’ - E_TIMEOUT
		x’80000000’ - E_WAKEUP
+2c	d_pn	process number
+30	*d_pcbid	pcb segment number (kern or sup processes)
+30	(*d_sproc)()	special process entry point
+34	d_sleep	sleep bit pattern
+38	d_ucnt	user count

+3a	d_fcode	fault code
		x’00’ - NOFLT
		x’10’ -> x’80’ reserved for *UNIX*® real-time reliable (RTR) operating system applications
		x’b1’ - FLT_DREP
		x’b2’ - FLT_PICP
		x’b3’ - FLT_CCP
		x’b4’ - FLT_DRED
		x’b5’ - FLT_ADRD
		x’b6’ - FLT_PICD
		x’b7’ - FLT_CCD
		x’c3’ - FLT_CMI
		x’c4’ - FLT_CMA
		x’c5’ - FLT_CMB
		x’c6’ - FLT_CMC
		x’c7’ - FLT_CMD
		x’c8’ - FLT_CMAN
		x’c9’ - FLT_UCLRMV
		x’cb’ - FLT_SOFTSW
		x’d1’ - FLT_PINV
		x’d2’ - FLT_PIND
		x’d3’ - FLT_SINV
		x’d4’ - FLT_SIND
		x’d5’ - FLT_BADOST
		x’d6’ - FLT_PROT
		x’d7’ - FLT_ADDR
		x’d8’ - FLT_PRIV
		x’d9’ - FLT_OPCD
		x’e1’ - FLT_SINIT
		x’e2’ - FLT_SCRIT
		x’f0’ -> x’ff’ reserved for *UNIX* RTR operating system applications
+3b	d_chan	control channel
+3c	d_cprior	current priority for supervisor process process ID for kernel process
+3d	d_iprior	initial priority for supervisor process execution level for kernel process
+3e	d_age	1/2 sec units process waiting for scheduling
+3b	d_unused	unused at this time
+40	d_pcode:11	pcode portion of the sup/kp utilid
	d_ucode:11	pcode portion of the user utilid
	d_spare:10	spare bits

* UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

+44	d_class	process class flag x’00000001’ - DC_DMERT x’00000002’ - DC_ESSEN x’00000003’ - DC_NONES
+48	d_eent	event entry (psw and function)
+50	d_oent	OST entry (psw and function)
+58	d_fent	fault entry (psw and function)
+60	d_sbr	segment base value
+64	d_enable	enable flag for event entry
+68	d_slptr	slist entry pointer *
+6c	d_psize	current size of this process
+70	d_audmap	audit flag
+71	d_aud1	audit spare
+72	d_msgcnt	total message blocks on queue
+74	d_start	process start time
+78	d_otime	time spent in OSTs
+7c	d_ptime	time spent servicing this process

Structure: dctext - length: 0x10

Source: head/dcte.h

Location: In the kernel’s address space at 0x340000 (may vary per head/va.h).

Use: An extension of the dct entry which, like the dcte, contains process related information.

+0	de_dctndx	the index of the dct with which this extender block is associated.
+2	de_state	creation/termination states Termination: 0x0064 - use count decremented 0x006e - term_dct() called 0x0078 - GRASP informed 0x0082 - unlinked from dispatch 0x008c - atb flushed 0x0096 - ack msg created 0x00a0 - unlinked from slist 0x00aa - incarnation cnt bumped 0x00b4 - forwarded to CMGR 0x00be - forwarded to PMGR 0x00c8 - caps removed 0x00d2 - sup segs removed 0x00dc - kp segs removed 0x012c - core dump started 0x0136 - forwarded from PMGR to CMGR Creation: 0x01f4 - kp pcreat started 0x01fe - kp dcte linked 0x0208 - E_INIT sent 0x0258 - sp pcreat started 0x0262 - sp dcte linked 0x02bc - fork started 0x02c6 - dupcaps sent to FMGR 0x02d0 - pfork2 started 0x02da - wakeup to child
+4	de_tstamp	time of last state change
+8	de_pmap1	1st word of the public library class map (reserved for applications use)

+c	de_pmap2	2nd word of the public library class map (reserved for *UNIX* RTR operating system use) 0x00000001 - ECD 0x00000002 - PLM 0x00000004 - KCONFIG 0x00000008 - *UNIX* RTR operating system 0x00000010 - CRAFT 0x00000020 - LLA incore 0x00000040 - LLA general

Structure: instat - length: 0x50

Source: head/instat.h

Location: Found on the interrupt stack for preempted processes and occasionally on a process's stack depending upon the activity of the process.

Use: Contains all system register values required to resume execution of a preempted process.

+0	i_pa	program address at interrupt
+4	i_psw	psw
+8	i_psbr	primary segment base register
+c	i_ssbr	secondary segment base register
+10	i_reg[0]	general purpose reg 0
+14	i_reg[1]	general purpose reg 1
+18	i_reg[2]	general purpose reg 2
+1c	i_reg[3]	general purpose reg 3
+20	i_reg[4]	general purpose reg 4
+24	i_reg[5]	general purpose reg 5
+28	i_reg[6]	general purpose reg 6
+2c	i_reg[7]	general purpose reg 7
+30	i_reg[8]	general purpose reg 8
+34	i_reg[9]	general purpose reg 9 (argument pointer)
+38	i_reg[10]	general purpose reg 10 (frame pointer)
+3c	i_reg[11]	general purpose reg 11 (stack pointer)
+40	i_reg[12]	general purpose reg 12
+44	i_reg[13]	general purpose reg 13
+48	i_reg[14]	general purpose reg 14
+4c	i_reg[15]	general purpose reg 15

Structure: kpcb - length: 0x800

Source: head/kpcb.h

Location: One segment in each kernel process address space. All kpcb segments can be located via the kernel’s dispatcher control table (DCT) entries.

Use: Contains all process specific information not needed directly by the kernel. Of most significance is the process’ segment table which is used by the microcode to define the process’ virtual address space.

+0	k_sbr	segment base register
+4	k_size	number of entries in segment list
+6	k_tflag	nonzero implies death of child msg requested
+7	k_ttype	death of child msg type
+8	k_eent	event entry vector, psw and pa
+10	k_oent	OST entry vector, psw and pa
+18	k_fent	fault entry vector, psw and pa
+20	k_pn	process number
+24	k_utilid	utility ID
+28	k_sgt	segment table
+228	k_seglist	segment list (128 - 8 byte entries)
	k_segflg sf_segndx:8 sf_flag:24	
		x’00000000’ - KF_FREE x’..000001’ - KF_EXEC x’..000002’ - KF_WRT x’..000004’ - KF_RD x’..000008’ - KF_STK x’..000010’ - KF_PWRT x’..000020’ - KF_SHARE x’..000040’ - KF_IOMAP
	k_segid	segment ID (pointer)
+628	k_parpn	parent process number
+62c	k_tident	death of child msg ident
+630	k_name[ ]	name of the process
+640	k_chan	control channel
+641	k_cspare	unused
+642	k_sspare	unused
+644	k_profad	profiling address

+648	k_uo[]	spare unsigned integer
+688	k_io[]	spare integer
+784	k_s0[]	spare short integers
+7c4	k_c0[]	spare characters

Structure: kvt - length: 0x208

Source: head/kvt.h

Location: Found in the kernel’s data segment in an external declaration called "Kvt". The exact address will vary from load to load.

Use: Contains all spy package metering data.

+0	*a_dctmem	virtual address of the DCT segment
+4	*a_dctpa	physical address of the DCT segment
+8	*a_depa	physical address of the DCTEXT segment
+c	dctcnt	total number of DCT entries
+10	*a_dctfree	address of the DCT free count
+14	*a_dispq	address of the dispatching queues
+18	*a_portmem	virtual address of the PORT segment
+1c	*a_portpa	physical address of the PORT segment
+20	portcnt	total number of ports
+24	*a_stckpa	physical address of the KSTACK segment
+28	stacksize	size of the KSTACK segment
+2c	*a_msgpa	physical address of the KMSG segment
+30	*a_mepa	physical address of the MSGEXT segment
+34	msgcnt	total number of message buffers
+38	*a_msgfree	address of the free msg blk count
+3c	*a_Ovldfg	address of the msg buf overload flag
+40	pdecnt	total number of physical pages in mod 0
+44	pde1cnt	total number of pages in mod 1
+48	*a_pdefree	address of the free page count
+4c	a_pde1free	total number of physical pages in mod 1
+50	*a_sdemem	virtual address of the SDE segment
+54	*a_sdepa	physical address of the SDE segment
+58	sdecnt	total number of SDT entries
+5c	*a_sdefree	address of the free SDE count
+60	*a_disksize	address of the disk swap size
+64	*a_diskfree	address of the free disk swap blk count
+68	*a_swapsize	address of the swap size
+6c	*a_swapmin	address of the swap size minimum
+70	*a_Swapis	address of the segs swapped in count
+74	*a_Swapos	address of the segs swapped out count
+78	*a_Swapib	address of the bytes swapped in count
+7c	*a_Swapob	address of the bytes swapped out count

+80	Sktime[ ]	16 word array depicting time spent in the kernel at each execution level
+c0	Skptime[ ]	16 word array depicting time spent in kernel processes at each execution level
+100	Sstime[ ]	total central processing time (CPU) spent in supervisors at each level (0 and 1)
+108	Sutime	total CPU time spent in user processes
+10c	*a_Tidle	address of idler loop counter
+110	*a_prevtod	address of last tod clock tick
+114	sdis_lev[ ]	16 words representing the dispatching counts of supervisors within the relative priority groupings
+154	sdis_dif[ ]	16 words representing the differences between supervisor initial and current priorities at dispatch time
+194	pcra_cnt	number of processes created
+198	pkil_cnt	number of processes killed
+19c	*nkost	address of a 150 word array of counts of the kp OST executions
+1a0	nKPost	address of kp activity array
+1a4	*nsupost	address of a 150 word array of counts of the supervisor OST executions
+1a8	nSUPost	address of level 2 activity indicator
+1ac	*nuost	address of a 100 word array of counts of the user OST executions
+1b0	*nint	address of a 17 word array of counts of the attachable interrupt occurrences
+1b4	*npir	address of a 16 word array of counts of the pir interrupt occurrences
+1b8	unused	

Structure: msghdr - length: 0x14

Source: head/msghdr.h

Location: Contained in each allocated message buffer in the kernel's message segment (shared with kernel processes). This segment (KMSG) is located at 0x620000 but may vary depending upon changes to head/va.h. Supervisors use local copies of this structure.

Use: Contains message control information used by the kernel as well as the sending and receiving processes.

+0	*ms_link	ptr to next msg on input queue (maybe)
+4	ms_from	sending process number
+8	ms_to	receiving process number
+c	ms_nblks	msg size in 64 byte blocks (max = 7)
+d	ms_flags	msg header flags x'01' - MS_CAP x'02' - MS_ULOCK x'04' - MS_NACK x'08' - MS_ALOC
+e	ms_type	message type x'00' FM_BADMSG MSMIN x'01' FM_READ P_CREAT IOREAD x'02' FM_WRITE IOWRITE x'03' FM_OPEN IOOPEN x'04' FM_CLOSE P_INIT IOCLOSE x'05' FM_EXEC P_WAIT x'06' FM_FORK P_INMEM x'07' DELCAP P_UNINMEM x'08' FM_CREAT ADDCAP x'09' FM_LINK MSTERM x'0a' FM_UNLINK MSGROW x'0b' FM_UTIME MSLOAD x'0c' FM_CHDIR MSPLOCK x'0d' FM_INIT MSKADD x'0e' FM_MKNOD MSKRMV x'0f' FM_CHMOD MSCMPCT x'10' FM_CHOWN x'11' FM_SYNC x'12' FM_STAT x'13' FM_SIZE x'14' FM_FSTAT x'15' FM_MOUNT x'16' FM_UMOUNT

+e	ms_type	message type x’17’ FM_MOVE x’18’ FM_ALLOC x’19’ FM_MNTSTAT x’1a’ FM_TASKAUD x’1b’ FM_UNFORK x’1c’ FM_ACCESS x’1d’ FM_USTAT x’1e’ FM_SEGCODE x’1f’ FM_TEMP x’20’ FM_BACKOUT x’21’ FM_PERM x’22’ FM_MV x’23’ FM_BUFRD x’24’ FM_BUFWRT x’25’ FM_LSEEK x’26’ FM_PIPE x’27’ FM_ATOMSW x’28’ FM_PERF x’2F’ FM_FSLAUD x’30’ FM_FSBAUD x’31’ FM_AUD x’32’ IOCANCEL x’33’ FM_FUAUD FM_LAST x’61’ IOSYSDLM x’64’ MSFAULT x’65’ MSRCVMSG x’6c’ T_LIBMSG x’7e’ ECDCHNG x’fb’ WAITMSG x’fc’ TRCSND x’fd’ TRCRCV x’fe’ MSSIG x’ff’ MSACK MSMAX
+f	ms_stat	message status x’00’ MSNOERR x’3f’ BADTYPE x’40’ SYSERR x’e0’ MSOLD x’e1’ MBOLOAD x’ff’ MSDEAD MSPFAIL
+10	ms_size	msg size in bytes
+12	ms_otype	original type before ack
+13	ms_seqnum	message sequence number
+14	ms_ident	message ID used by sender x’fffffffc’ - TRCMSG x’fffffffd’ - USRMSG x’fffffffe’ - SIGMSG x’ffffffff’ - UNXMSG

Structure: pcb - length: 0x7f4

Source: head/pcb.h

Location: One segment in each supervisor process address space. All pcb segments can be located via the kernel's dispatcher control table (DCT). The most currently running supervisor will have it's pcb segment in the kernel's address space at 0x420000 (may vary per header va.h).

Use: Contains all supervisor specific information not needed directly by the kernel.

+0	p_pn	process number
+4	p_parpn	parent process number
+8	p_wait	scheduler flag x'00000000' - P_DONTCARE x'00000001' - P_INCORE x'FFFFFFFF' - P_OUTCORE
+a	p_chan	control channel number
+b	p_prior	initial priority
+c	p_tocnt	time slice runout count
+d	p_crflag	time slice runout in critical region
+e	p_tflag	message to parent at termination flag
+f	p_ttype	message type at termination
+10	p_tident	message ID to parent at death
+14	p_name	ASCII name of the process
+24	p_ttg	time to go on time slice
+28	p_slice	time slice
+2c	p_size	number of entries in segment list
+2e	p_cwait	event expected flag
+30	p_ktime	time spent in the kernel
+34	p_kptime	time spent in kernel process
+38	p_stime	time spent in supervisor mode
+3c	p_utime	time spent in user mode
+40	p_runtime	accumulated run time up to 60 ms
+44	p_spsbr	supervisor process psbr
+48	p_upsbr	user process psbr
+4c	p_topsd	psd save area at preemption or timeout
+4c	ps_psw	psw at preemption
+50	ps_pa	program address at preemption
+54	p_tosave	register save area

+94	p_semafor	not used
+95	p_fcode	fault code
+96	p_static	static scheduling priority
+97	p_evopt	event wait option
+98	p_evwait	mask for event wait flags x’00000000’ - P_EWANY
+9c	p_evflg	event flags x’00000001’ - E_USR16 x’00000002’ - E_USR15 x’00000004’ - E_USR14 x’00000008’ - E_USR13 x’00000010’ - E_USR12 x’00000020’ - E_USR11 x’00000040’ - E_USR10 x’00000080’ - E_USR9 x’00000100’ - E_USR8 x’00000200’ - E_USR7 x’00000400’ - E_USR6 x’00001000’ - E_USR4 x’00002000’ - E_USR3 x’00004000’ - E_USR2 x’00008000’ - E_USR1 x’00010000’ - E_SYS16 x’00020000’ - E_SYS15 x’00040000’ - E_SYS14 x’00080000’ - E_SYS13 x’00100000’ - E_SYS12 x’00200000’ - E_SYS11 x’00400000’ - E_UTIL x’00800000’ - E_RTIMEOUT x’01000000’ - E_INIT x’02000000’ - E_ABORT x’04000000’ - E_QIT x’08000000’ - E_INT x’10000000’ - E_HUP x’20000000’ - E_MSG x’40000000’ - E_TIMEOUT x’80000000’ - E_WAKEUP
+a0	p_evmsk	event mask
+a4	p_evect	entry vector to event handling routine
+a4	pe_psw	psw at event entry
+a8	*pe_pa()	address of event entry
+ac	p_evpsd	psd save area at event entry
+b4	p_fvect	entry vector to fault handling routine
+bc	p_fpsd	psd save area at fault entry
+c4	p_ovect	entry vector to OST handling routine
+cc	p_initsp	initial stack pointer value

+d0	p_svect	starting entry vector
+d8	p_clist	capability list (22 - 2 word entries)
+d8	cp_owner	owner process
+dc	cp_cap	capability
+188	p_sutilid	supervisor utility ID
+18c	p_ssgt	supervisor segment table
	st_cntl:4	control bits x'1' - ST_VALID x'2' - ST_EXEC x'4' - ST_WRT x'8' - ST_RD
	st_pgtln:6	# words in page table - 1
	st_ptad:22	physical address of page table
+38c	p_seglist	segment list (128 - 8 byte entries)
	p_segflg	
	sf_segndx:8	
	sf_flag:24	
		x'00000000' - SF_FREE x'..000001' - SF_EXEC x'..000002' - SF_WRT x'..000004' - SF_RD x'..000008' - SF_STK x'..000010' - SF_PWRT x'..000020' - SF_SHARE x'..000040' - SF_NOLD x'..000080' - SF_NONSW x'..000100' - SF_SBIT x'..000200' - SF_UBIT x'..000400' - SF_NXT x'..000800' - SF_NN
	p_segid	segment ID (pointer)
+78c	p_fup	field update patch count
+790	p_state	
	i_pa	
	i_psw	
	i_psbr	
	i_ssbr	
	i_reg[16]	
+7e0	p_profaddr	profiling address
+7e4	p_uO	spare

Structure: psw - length: 0x04

Source: head/psw.h

Location: The current psw is located in the "PSW" special register, preempted psw values are found on the interrupt stack, entry point psws are found in the DCT for kernel processes and in PCB segments for supervisors, and attachable entry point psws are found in the atchtbl[ ] array (of ative structures) in the kernel's data segment.

Use: Used by the microcode to determine the required system environment for the currently running process.

+0	w_mode:2	processor mode
		x'0.......' - W_MKRN
		x'4.......' - W_MKP
		x'8.......' - W_MSUP
		x'c.......' - W_MUSR
+0.5	w_exlev:6	execution level
		x'.0.......' - level 0
		x'.1.......' - level 1
		" "
		" "
		x'.f.......' - level f
+1	w_prvlg:4	privilege bits
		x'..1.....' - W_SETEX
		x'..2.....' - W_NMIO
		x'..4.....' - W_SYSIO
		x'..8.....' - W_WPSW
+1.5	w_emcntl:4	emulation control
+2	w_ssbr:3	secondary sbr
	w_psbr:3	primary sbr
	w_flag:6	bit flags
		b'..00,0001....' - W_KSTK
		b'..00,0010....' - W_SPARE
		b'..00,0100....' - W_ISTK
		b'..00,1000....' - W_MMON
		b'..01,0000....' - W_SRC
		b'..10,0000....' - W_DEST
+3.5	w_cond:4	condition codes
		x'.......1' - W_CBIT
		x'.......2' - W_NBIT
		x'.......4' - W_VBIT
		x'.......8' - W_ZBIT

Structure: sde - length: 0x20

Source: head/sde.h

Location: Found in the kernel's address space starting at segment index 13 (0x1a0000). This address is dependent upon header va.h and may move from load to load.

Use: Memory management routines use the SDT to map all segments known to the system, either in memory or on the swap device.

+0	*s_pgtptr	virtual address of the page table
+4	*s_link	link for swappable segment or free sde's
+8	s_plkcnt	process lock count
+9	s_lkcnt	I/O lock count
+a	s_nswcnt	nonswap count
+c	s_active	# don't swap list proc's allocating segment
+e	s_users	# of processes having allocated the segment
+10	s_lstpgsz	number of bytes in the last page
+12	s_tlpg	total number of pages
+13	s_inmmpg	total number of pages in memory
+14	s_stat	segment status word x'00000000' - SS_FREE x'..000002' - SS_BROKE x'..000004' - SS_WRT x'..000008' - SS_GBCLT x'..000010' - SS_PURGE x'..000020' - SS_REMOV x'..000040' - SS_UTLY x'..000080' - SS_IOFAIL x'..000100' - SS_IOIN x'..000200' - SS_IOOUT x'..000400' - SS_LOCK x'..000800' - SS_NONSW x'..001000' - SS_ALT x'..002000' - SS_NEXT x'..004000' - SS_ACT x'..008000' - SS_PLOCK x'..010000' - SS_NSWSP x'..020000' - SS_BRKDN x'..040000' - SS_KPCB x'..080000' - SS_SPCB x'..100000' - SS_BLOCK x'..200000' - SS_NEW x'..400000' - SS_PGPRT x'..800000' - SS_ALLOC x'10000000' - SS_ONFL x'20000000' - SS_MOD1 x'40000000' - SS_MOD0
+18	s_swapaddr	starting block # on swap device
+1c	sde_name	segment name

Kboot Address Space

This section lists the control block structures in the kboot address space. The lists name each field and give the hexadecimal offset to it from the beginning of the structure.

Structure: bootab - length: 0x1060

Source: head/sgenbt.h

Location: In the kernel’s address space somewhere near the end of the text segment under the external name "kbootab".

Use: Contains the mapping information used by kboot to create the segments and processes making up the boot.

+0	bt_ecdversion	ECD version number
+4	bt_nseg	number of valid entries in the bt_seg[ ] array
+6	bt_nprc	number of valid entries in the bt_prc[ ] array
+8	bt_ksdx[ ]	indices of the kernel’s bt_seg[ ] entries
+40	bt_seg[ ]	boot image segment descriptors - each is a bsegdes structure
+a40	bt_prc[ ]	boot image process descriptors - each is a bprcdes structure
+bc0	bt_kparm	kernel dynamic memory parameters - see the btkparm structure
+c04	bt_npaths	number of processes to be pcreated
+c06	bt_nlibs	number of public libraries to be loaded
+c08	bt_upath[ ]	pathnames of pcreated processes
+c60	bt_libpath[ ]	pathnames of boot public libraries

Structure: bprcdes - length: 0x18

Source: head/sgenbt.h

Location: In the kernel’s address space somewhere near the end of the text segment under the external name "kbootab" which contains an array of bprcdes structures.

Use: Contains the mapping information used by kboot to create the processes making up the boot.

+0	pd_pnum	fixed process number
+4	pd_class	process class
+8	pd_pcbsdx	index of the process’ (k)pcb segment in bt_seg[ ]
+a	pd_flags	process flags 0x00000001 - kernel process 0x00000002 - supervisor 0x00000200 - shares segment with child 0x00000400 - shares segment with parent 0x00000800 - process being notified 0x00001000 - dlm essential 0x00002000 - static 0x00004000 - noterm
+c	pd_prior	supervisor initial priority or kernel process execution level
+e	pd_spare	unused
+10	pd_nseg	number of boot image segments
+12	pd_nints	number of interrupts to attach
+14	pd_libflags	public library bit map

Structure: bsegdes - length: 0x14

Source: head/sgenbt.h

Location: In the kernel’s address space somewhere near the end of the text segment under the external name "kbootab" which contains an array of bsegdes structures.

Use: Contains the mapping information used by kboot to create the segments making up the boot.

+0	sd_kbva	virtual address in the kboot address space of the segment initial image
+4	sd_segsize	size of the segment in bytes
+8	sd_segndx	true segment index of the segment
+a	sd_users	number of processes using the segment
+c	sd_segflgs	segment flag word 0x00000007 - segment protection flags 0x00000200 - LDP ’shared’ segment 0x00000400 - LDP ’common’ option 0x00000800 - PAS segment 0x00002000 - ECD segment 0x00008000 - segment is part of the kernel
+10	*sd_segid	segment ID of the true segment. This field is filled in by kboot.

Structure: btkparm - length: 0x44

Source: head/sgenbt.h

Location: In the kernel’s address space somewhere near the end of the text segment under the external name "kbootab". This structure is contained in kbootab.

Use: Defines the kernel’s dynamic memory segments.

+0	km_nmsg	number of message blocks to be allocated
+2	km_nport	size of the port segment (in words)
+4	km_nprc	number of dct entries
+6	km_nsegecd	number of ecd segments
+8	km_nseg	number of SDT entries
+c	km_npgt	number of page tables
+10	km_npage	number of PDT entries
+14	km_istkb	size of the interrupt stack (in bytes)
+18	km_kstkb	size of the kernel stack (in bytes)
+1c	km_swstart	starting block of swap area
+20	km_swblks	size of swap area (in blocks)
+24	km_swmin	swap size of largest supervisor (in pages)
+28	km_intlen	initialization interval
+2c	km_maxlevs[ ]	application phase levels
+30	km_PASndx	segment index of low PAS
+34	km_1PASndx	segment index of high PAS
+38	km_PASdm	PAS dump/nodump flag
+3c	km_part	partition boundary between mod 0 and mod1
+40	km_sched	scheduler’s time-out value

File Manager Address Space

This section lists the control block structures in the file manager address space. The lists name each field and give the hexadecimal offset to the field from the beginning of the structure.

Structure: bdevtab - length: 0x4c

Source: head/fmgr/fmgr.h

Location: An externally declared array called ‘‘bdevtab’’.

Use: One entry for each block device driver (indexed by dcn). Internal file manager buffers are chained off an array of pointers using a hash selection of ’and’ing 0x7 to the block number.

+0	d_proc	process number of the driver
+4	boflag	if nonzero then the driver process is to get an open or close message with each open or close request. If zero then the driver only gets one open and one close.
+8	d_bchain[8]	buffer device chain pointer array
+8	b_forw	forward chain pointer
+c	b_back	backward chain pointer
+48	b_extra[4]	unused

Structure: buf - length: 0x40

Source: os/fmgr/head/buf.h

Location: An externally declared array called ‘‘buf’’. Free buffers are chained off of ‘‘bfreelist’’ (also a buf structure). Buffers associated with a device are chained off of a bdevtab entry. Buffers explicitly associated with no device are chained off of ‘‘bfreelist’’ (another chain).

Use: Each buf structure (2*NTASKS+4) controls an I/O buffer.

+0	*b_forw	buf pointer headed by bdevtab
+4	*b_back	buf pointer headed by bdevtab
+8	b_flags	buffer flags
		0x00000000 - B_WRITE - non-read pseudo flag
		0x00000001 - B_READ - read flag
		0x00000002 - B_DONE - I/O complete
		0x00000004 - B_ERROR - I/O error
		0x00000008 - B_BUSY - buffer in use (locked)
		0x00000030 - B_XMEM - memory extension (unused)
		0x00000040 - B_WANTED - buffer wanted (task asleep waiting)
		0x00000080 - B_AGE - delayed write for correct aging (controls placement on available queue)
		0x00000100 - B_ASYNC - no wait for completion
		0x00000200 - B_DELWRI - delayed write (holds buffer between uses)
		0x00000400 - B_IO - I/O outstanding on this buf
		0x00000800 - B_MOUNT - this buffer contains the superblock of a mounted file system
		0x00001000 - B_FSAUD - this buffer is being used by the file system audit
+c	*av_forw	buf pointer headed by bfreelist
+10	*av_back	buf pointer headed by bfreelist
+14	b_mdct	device mdct-rid
+18	b_dcn	device major number
+1a	b_part	device partition
+1c	b_un	union
+1c	b_addr	address of actual buffer
+1c	*b_words	pointer to words for clearing
+1c	*b_filsys	pointer to superblock
+1c	*b_dino	pointer to block in ilist
+1c	*b_daddr	pointer to indirect block
+20	b_blkno	block # on device
+24	*b_mptr	ptr to mount table entry (null unless B_MOUNT set)
+28	b_taskid	taskid information (null unless B_BUSY set)

+2c	b_tstamp	time when io started (2 minutes from this time results in automatic task teardown)
+30	b_extra[16]	structure padding

Structure: cap - length: 0x18

Source: os/fmgr/head/cap_tbl.h

Location: An externally declared array called ‘‘cap_tbl’’.

Use: One is maintained for each open or fork.

+0	c_cap	capability
+4	c_pid	client process number
+8	*c_fptr	ptr to corres. file table entry
+c	*c_cptr	ptr to next capability for this file (cap table entries are chained only on forks off of the same open, each points to the same file table entry)
+10	c_tstamp	time c_pid was first noted as invalid

Structure: cpmsghdr - length: 0x28

Source: head/cpmsghdr.h, see also head/fmgr/....

Location: Message buffer formats found in the message segment. Pointers to dequeued messages will be found in ‘‘tasktab’’ or one of the delayed_q’s.

Use: The means in which requests are made to the file manager. All message formats begin with a capability message header.

+0	cpm_mshd	standard message header (msghdr.h)
+0	ms_link	link to next message (delay queues only)
+4	ms_from	sending process
+8	ms_to	receiving process
+c	ms_nblks	msg size in blocks
+d	ms_flags	msg flags
+e	ms_type	message type
+f	ms_stat	message status
+10	ms_size	message size in bytes
+12	ms_otype	original message type
+13	ms_seqnu	unused
+14	ms_ident	message identifier
+18	cpm_mc	capability field
+18	mc_num	capability number
+1c	cp_owner	owner process
+20	cp_cap	capability bits 0 -> 7: i_use count bits 8 -> 15: permissions bits 16 -> 31: file table index
+24	cpm_guid	group/user id
	type 0x01	FM_READ - read file
		request:
+28	fm_iosid	segment id
+30	fm_iobyoff	byte offset into segment
+34	fm_iocnt	number of bytes for I/O
+38	fm_ioblk	block number for I/O
+40	fm_iores	remaining bytes to be transferred
		reply:
+28	ret0	not used
+2c	ret1	not used
+30	fm_aiocnt	no bytes transferred
+34	fm_iortry	memory manager used field
+38	fm_err0	not used

+3c	fm_err1	error return from driver
	type 0x02	FM_WRITE - write file
		see type 0x01 request and reply
	type 0x03	FM_OPEN - open file
		request:
+28	fm_ocfoff	offset to name
+2c	fm_ocmode	mode for open or create
		0x000 - OP_READ - open: read
		0x001 - OP_WRITE - open: write
		0x002 - OP_RW - open: read/write
		0x001 - CR_XBYOT - create: execution by others
		0x002 - CR_WBYOT - create: write by others
		0x004 - CR_RBYOT - create: read by others
		0x008 - CR_XBYG - create: execute by group
		0x010 - CR_WBYG - create: write by group
		0x020 - CR_RBYG - create: read by group
		0x040 - CR_XBYOW - create: execute by owner
		0x080 - CR_WBYOW - create: write by owner
		0x100 - CR_RBYOW - create: read by owner
		0x200 - CR_SVTXAX - create: save text after execute
		0x400 - CR_SGIDX - create: set group id on execute
		0x800 - CR_SUIDX - create: set user id on execute
+30	fm_nopcr[ ]	filename
		reply:
+28	fm_ocapno	capability number (-1 for kernel process opens) (if pipe - cap num of read end)
+2c	fm_otype	type of file (if pipe - cap num of write end)
+30	fm_procid	process number of device file
+34	fm_unused	unused
+38	fm_odevid	mdct and partition
+40	fm_ofilsiz	file size
	type 0x04	FM_CLOSE - close file
+28	fm_usecnt	# file descriptors using inode
+2c	fm_clmode	close mode
	type 0x05	FM_EXEC - open file for execution
		request:
+28	fm_off	filename offset
+2c	fm_name[ ]	file name
		reply:
+28	fm_ecapno	capability number
+2c	fm_segname	unique segment name
	type 0x06	FM_FORK - increment count for open file
+28	fm_fkcapc	number of capabilities
+2c	fm_fkchpid	child process number
+30	fm_fkchpid	fcount increment/decrement
+34	fm_fkmce[ ]	list of capnums and caps
	type 0x08	FM_CREAT - create file
		see type 0x03 request and reply

	type 0x09	FM_LINK - link to a file
+28	fm_loff	offset to existing pathname
+2c	fm_loff1	offset to link name
+30	fm_nlink[ ]	existing and link names
	type 0x0a	FM_UNLINK - remove link from file
		see type 0x05 request
	type 0x0b	FM_UTIME - modify date of file
+28	fm_utoff	filename offset
+2c	fm_uflag	flag
		>>>>>>>> what values <<<<<<<<
+30	fm_utime	utime structure
+30	f_atime	new access time
+34	f_mtime	new modify time
+38	f_ctime	new create time
+3c	fm_nutime[ ]	filename
	type 0x0c	FM_CHDIR - change directory
		request:
		see type 0x05 request
		reply:
+28	fm_chcapno	capability number
	type 0x0d	FMINIT - file manager initialization
+28	fm_idevid	mdct and partition of root device
+34	fm_idcnid	major device number root device
+38	fm_ipnum	root device process number
	type 0x0e	FM_MKNOD - make a node
+28	fm_mkoff	offset to name
+2c	fm_mkmode	permissions and file type
+30	fm_mkdvid	mdct and partition
+38	fm_mkdcn	major device number
+3a	fm_nmknod[ ]	name of the file
	type 0x0f	FM_CHMOD - change mode of file
+28	fm_chmoff	filename offset
+2c	fm_chmode	new mode of file
+30	fm_nchmod[ ]	filename
	type 0x10	FM_CHOWN - change owner of file
+28	fm_choff	filename offset
+2c	fm_uown	user id of new owner
+30	fm_gown	group id of new owner

+34	fm_nchown[ ]	filename
	type 0x11	FM_SYNC - update file systems on secondary
		capability header only
	type 0x12	FM_STAT - get file status
		request:
		see type 0x05 request
		reply:
+28	fm_stat	‘‘stat’’ structure
+28	st_dev	device number
+2c	st_ino	inode number
+30	st_mode	file mode (see mode field of inode)
+32	st_nlink	link count
+34	st_uid	user id
+36	st_gid	group id
+38	st_rdev	special file device name
+3c	st_size	length in bytes
+40	st_atime	time last accessed
+44	st_mtime	time last modified
+48	st_ctime	time created
	type 0x13	FM_SIZE - get file size
+28	fm_fsize	size of file (reply only)
+2c	fm_fssize	total amount allocated contiguous (reply only)
	type 0x14	FM_FSTAT - get status of open file
		request:
		capability header only
		reply:
		see type 0x12 reply
	type 0x15	FM_MOUNT - mount file system
+28	fm_moff	offset to device name
+2c	fm_moff1	offset to mount point name
+30	fm_mronly	action flags 0x00000001 - read only access 0x00000002 - audit the file system
+34	fm_nmount[ ]	device and mount point names
	type 0x16	FM_UMOUNT - unmount file system
		see type 0x05 request
	type 0x17	FM_MOVE - move file to contiguous area
		request:
		see type 0x05 request
		reply:
+28	fm_fmblk	number of blocks available

	type 0x18	FM_ALLOC - allocate contiguous space
		request:
+28	fm_faoff	filename offset
+2c	fm_famode	permissions and file type
+30	fm_fasize	file size
+34	fm_nfall[ ]	filename
		reply:
+28	fm_facapno	capability number
+2c	fm_fatype	type of file
+30	fm_faprocid	not used
+34	fm_fachan	not used
+38	fm_fancblk	number of blocks in file
	type 0x19	FM_MNTSTAT - mount status
	type 0x1a	FM_TASKAUD - high priority task audit
	type 0x1b	FM_NMCODE - get segment name
		not currently available
	type 0x1c	FM_ACCESS - check access permissions
+28	fm_acoff	filename offset
+2c	fm_acmode	access request
+30	fm_naccess[ ]	filename
	type 0x1d	FM_USTAT - pack label
		request:
+28	fm_mvdev	mdct and partition
		reply:
+28	fm_ustat	ustat structure
+28	f_tfree	total free
+2c	f_tinode	total inodes free
+30	f_fname[ ]	file system name
+36	f_fpack[ ]	file system pack name
	type 0x1e	FM_SEGCODE - return segment name
		request:
+28	fm_segcode	code byte
+29	fm_segflag	flag byte
		reply:
+28	fm_segname	unique name for segment
	type 0x1f	FM_TEMP - no disk writes on file
		see type 0x05 request
	type 0x20	FM_BACKOUT - restore (core copy of) file
		see type 0x05 request
	type 0x21	FM_PERM - untemp file (write to disk)
		see type 0x05 request
	type 0x22	FM_MV - windowless move
+28	fm_off	offset to ‘‘from’’ filename
+2c	fm_off1	offset to ‘‘to’’ filename
+30	fm_nmv[ ]	‘‘from’’ and ‘‘to’’ filenames
	type 0x23	FM_BUFRD - buffered read
		request:
+28	fm_bfsg1	1st segment id

+2c	fm_bfsg2	2nd segment id
+30	fm_bfoff	offset into 1st segment
+34	fm_bfcnt	number of bytes for I/O (max 128k)
		reply:
+28	ret0	not used
+2c	ret1	not used
+30	fm_aiocnt	# bytes transferred
+34	fm_iortry	memory manager used field
+38	fm_err0	no used
+3c	fm_err1	error return from driver
	type 0x24	FM_BUFWRT - buffered write
		see type 0x23 request and reply
	type 0x25	FM_LSEEK - lseek
		request:
+28	fm_lsoff	file offset
+2c	fm_lscmd	lseek command type
		reply:
+28	fm_lsaoff	resultant file offset
	type 0x26	FM_PIPE - open pipe
		see type 0x03 request and reply
	type 0x27	FM_ATOMSW - atomic switch
		see type 0x22
	type 0x28	FM_PERF - performance reporting
		request:
+28	fm_prtyp	performance request type
		0x00000000 - FMP_REQ - report request frequency 0x00000001 - FMP_ERR - report error frequency 0x00000002 - FMP_MISC - report misc. info
		misc. info reply:
+28	fm_fault	fault count
+2c	fm_xfault	external faults
+30	fm_pfault	phase-1 faults
+34	fm_taskfault	task faults
+38	fm_init	initialization events
+3c	fm_util	utility events
+40	fm_retry	driver retry errors
+44	fm_unknown	unknown acknowledgments
+48	fm_bfhit	buffer hits
+4c	fm_bfmiss	buffer misses
+50	fm_bfgbusy	times buffer was busy
+54	fm_bfgempty	
+58	fm_bfgdelw	delayed writes
+5c	fm_nam	namei calls
+60	fm_restraint	restrained requests
+64	fm_teardown	task torn down
+68	fm_mntdown	mount table entries restored
+6c	fm_bufdown	buffers freed by teardown
+70	fm_inodown	inodes freed by teardown
+74	fm_fildown	file entries freed by teardown
+78	fm_sbidown	unlocks of SUPERB ilock

+7c	fm_sbfdown	unlocks of SUPERB flock
+80	fm_inocnt	active inode slots
+84	fm_inomax	high water active inode slots
+88	fm_filecnt	active file table slots
+8c	fm_filemax	highwater active file slots
+90	fm_extra	
	type 0x2f	FM_FSLAUD - file system link audit
	type 0x30	FM_FSBAUD - file system block audit
	type 0x31	FM_AUD - audit request from sim
	type 0x33	FM_FUAUD -msg from field update audit

Structure: delayed_q - length: 0x10

Source: head/fmgr/fmgr.h

Location: One externally declared array for each restraint queue, currently ‘‘open_q’’ and ’’mount_q’’.

Use: Certain file manager requests are throttled. This structure is used to chain requests which must wait for completion of earlier requests.

+0	q_count	# of active requests

+4	q_max	max # of active requests allowed
		for ‘‘open_q’’: 4 (NSOPEN)
		for ‘‘mount_q’’: 1 (NSMOUNT)

+8	*q_firstp	pointer to 1st message on queue
+c	*q_lastp	pointer to last message on queue

Structure: file - length: 0x20

Source: head/fmgr/file.h

Location: An externally declared array called ‘‘file’’.

Use: Contains file specific information with one entry for each original open (subsequent opens are chained).

+0	f_flags	file flags 0x00000001 - FLOCK - file table entry locked 0x00000002 - FWANT - another task wants (is asleep waiting)
+4	*f_iptr	pointer to incore inode
+8	*f_cptr	pointer to capability (chain)
+c	*f_fptr	pointer to next file entry on chain (there is one file table entry for each open, all point to the same inode)
+10	f_taskid	taskid information (valid only if FLOCK set)
+14	f_count	use count, > 1 indicates fork has occurred and is the number of chained cap_tbl entries
+18	f_ocount	previous (to this task) count (used for task teardown)
+1c	f_offset	read/write character pointer

Structure: filsys - length: 0x1dc

Source: head/sys/filsys.h

Location: The superblock of a file system. Mounted file systems will have the superblock in a buffer pointed to by a buf structure which is in turn pointed to by an entry in the mount table.

Use: Controls the access to the file system.

+0	s_isize	size in blocks of I list (plus 2)
+2		unused
+4	s_fsize	size in blocks of the file system
+8	s_nfree	number of incore free blocks (index+1 into incore free chain for the next free block)
+a		unused
+c	s_free[ ]	incore free chain control
	s_free[0]	ptr to blk containing 2nd free block array
	s_free[1]	ptr to blk containing 1st free block array
	s_free[2]-[49]	contains fsysdiag structure
+d4	s_ninode	number of incore free inodes (index +1 into s_inode for next free inode)
+d6	s_inode[100]	incore free inodes
+19e	s_flock	lock during free list manipulation
+19f	s_ilock	lock during I list manipulation
+1a0	s_fmod:1 s_ronly:1 s_ifull:1 s_bfull:1	super block modified flag mounted read-only flag ilist full flag blocks full flag
+1a4	s_time	current date of last update
+1a8	s_dinfo[4]	device information
+1b0	s_tfree	total free, for subsystem examination
+1b4	s_tinode	free inodes, for subsystem examination
+1b6	s_fname[6]	file system name
+1bc	s_fpack[6]	file system pack name
+1c4	s_ftaski	(struct) taskid information (flock)
+1c8	s_itaski	(struct) taskid information (ilock)
+1cc	s_cfree	free blocks on chain
+1d0	s_nxtblk	next free block not on chain (first zero in file system bit map)
+1d4	s_nxtcon	next available contiguous area
+1d8	s_label	file system label (0xdead3bcc)

Structure: inode - length: 0x60

Source: head/sys/inode.h

Location: The incore inode structure maintained in the file manager's inode table in a segment by itself under the external name ‘‘inode" (currently segment address 0x2c0000).

Use: One for each active file, each current directory, each mounted-on file, and root.

+0	i_hchain	inode hash chain pointer
+4	i_flag	inode flags x’0001 - ILOCK - locked x’0002 - IUPD - has been modified x’0004 - IACC - update access time x’0008 - IMOUNT - mounted-on x’0010 - IWANT - process is waiting on lock x’0020 - ITRUNC - to be truncated x’0040 - ICRT - has been created x’0080 - IXSYNC - has been temped (do not write inode to disk) x’0100 - ITRUNC2 - has been truncated without freeing blocks at least once. x’0200 - IFSAUD - being audited (fsaud) x’0400 - ITRSHD - fsaud says is trashed.
+8	i_count	total reference count (incremented each task use - inode slot not reused unless count goes to zero)
+9	i_wcount	reference count (writes)
+a	i_invoc	invocation count for inode on disk
+b	i_use	usage count for incore inode (incremented only when inode slot is assigned - put in capabilities)
+c	i_number	i number, 1-to-1 with device address
+e	i_mindx	mount table index at mount point
+f	i_unused	
+10	i_dev	mount table pointer of the file system containing this inode.
+14	i_taskid	(struct) taskid information (valid only if ILOCK set)
+18	i_tstamp	time stamp of last disk update
+1c	*i_fptr	pointer to file entry chain
+20	i_mode	mode of inode bit flags: x’0040 - IEXEC - execute permission x’0080 - IWRITE - write permission x’0100 - IREAD - read permission x’0400 - ISGID - set group id on execution

		x’0800 - ISUID - set user id on execution
		inode types:
		x’1000 - IFIFO - FIFO special
		x’2000 - IFCHR - character special
		x’3000 - IFMPC - multiplexed character
		x’4000 - IFDIR - directory
		x’5000 - IPIPE - *UNIX* system pipe
		x’6000 - IFBLK - block special
		x’7000 - IFMBP - multiplexed block
		x’8000 - IFREG - regular
		x’a000 - IFEXT - contiguous extents
		x’b000 - IF1EXT - one contiguous extent
		x’c000 - IFIOP - IOP special
		x’e000 - IFREC - record
		x’f000 - IFMT - inode file type mask
+22	i_nlink	directory entries (# of links)
+24	i_uid	owner
+26	i_gid	group of owner
+28	i_size	size of file - end of actual data
+2c	i_addr[ ]	disk addresses
+2c	i_addr[0]	direct block address (normal file)
		mdct-rid (special device file)
+30	i_addr[1]	direct block address (normal file)
		dcn (special device file)
+34	i_addr[2]	direct block address (normal file)
		partition (special device file)
		name (FIFO or pipe special device file)
+38	i_addr[3]->[9]	direct block addresses (normal file)
+54	i_addr[10]	single indirect block address (normal file)
+58	i_addr[11]	double indirect block address (normal file)
+5c	i_addr[12]	triple indirect block address (normal file)

Structure: measf - length: 0x1e4

Source: os/fmgr/head/meas.h

Location: An externally declared structure named ‘‘mf’’.

Use: A collection of counters which characterize the activity of the file manager since the last boot.

+0	m_fill1	x’dead3b indicates start
+4	m_fhist[FM_LAST+1]	frequency histogram for requests
+d4	m_fill2	x’dead3b indicates end m_fhist
+d8	m_err[NERR]	frequency histogram for errors
+1a0	m_fill3	x’dead3b indicates end of m_err
+1a4	m_fault	# of faults
+1a8	m_xfault	# of external faults
+1ac	m_pfault	# of phase 1 faults
+1b0	m_taskfa	# of task faults
+1b4	m_retry	# of retry errors from drivers
+1b8	m_unknow	# of unknown acks received
+1bc	m_bfhit	# of buffer hits
+1c0	m_bfmiss	# of buffer misses
+1c4	m_bfgbus	# of times buffer was busy
+1c8	m_bfgemp	unknown
+1cc	m_bfgdel	# of delayed writes
+1d0	m_restra	# of restrained requests
+1d4	extra[16]	

Structure: mnttab - length: 0x58

Source: head/mnttab.h

Location: An externally declared array called ‘‘mnttab’’.

Use: One entry for each mounted file system telling the pathname and file system name.

+0	mt_dev	device file name
+20	mt_filsys	file system name
+40	mt_ro_fl	read only flag
+44	mt_time	mount time ?????
+48	mt_devid	new device id
+48	dev_mdct	
+4c	dev_part:16	
+50	mt_dcn	major device

Structure: mount - length: 0x48

Source: head/fmgr/fmgr.h

Location: An externally declared array called ‘‘mount’’.

Use: One entry is maintained for each file system mount. Primarily used for file access across file systems. Points to the inode of the root directory of the file system as well as the inode of the mount point (both of which have their use counts incremented).

+0	m_mdct	device mdct-rid
+4	m_dcn	device major number
+6	m_part	device partition
+8	*m_bptr	pointer to superblock bfr header
+c	*m_fcbptr	pointer to free chain bfr header
+10	*m_iptr	pointer to mounted inode (inode of directory mounted upon)
+14	*m_rootp	pointer to root inode of filesys
+18	m_use	use count
+19	m_audited	file system has been audited bits 0 -> 4: has been audited flag bits 5 -> 7: audit flags
+1c	m_taskid	taskid information
+20	m_prevmn	flag for previous mount (used for task teardown)
+21	m_rdonly	flag for previous read only (used for task teardown)
+22	m_syncmax	number of syncs since last SB write
+23	m_spare1	unused
+24	m_fsinfo	fsysdiag structure

+24	fs_mts	mount time stamp
+28	fs_opens	opens since mount
+2c	fs_close	closes since mount
+30	fs_reads	total direct reads
+34	fs_writes	total direct writes
+38	fs_seio	buffered block I/O errors
+3a	fs_bdeio	buffered data block I/O errors
+3c	fs_deio	not-buffered data block I/O errors
+40	fs_aflag	file system audit flags
+42	fs_audcnt	audits since mount
+44	fs_blker	block audit error count
+46	fs_lnker	link audit error count

Structure: msgbufe - length: 0x10

Source: head/msgbufe.h

Location: In the kernel’s address space at 0x360000 (may vary per head/va.h). The index into the segment for each element is the same as the index into the message segment for it’s corresponding message buffer.

Use: A kernel private segment used for queuing and auditing the message buffers.

+0	*me_link	message queue link field
+4	me_owner	current owner of the associated message buffer
+8	me_tstamp	the time this message buffer last changed status (allocated buffers only)
+c	me_blks	total blocks allocated
+d	me_flags	message extender flags x’08’ - allocated x’10’ - old message (audit flag) x’20’ - old queued message x’40’ - currently on a message queue x’80’ - has been dequeued at least once
+e	me_type	message type of the associated message buffer
+f	me_unused	unused at this time

Structure: tasktab - length: 0x800

Source: os/fmgr/task.h

Location: One per task segment as declared in task0.c through task15.c (currently segment addresses 0x40000 through 0x220000). The addresses of each of the tasks are located in an externally declared array called ‘‘tasktab’’. The declaration ‘‘taskptr’’ points to the currently active task.

Use: Each defines the state of its associated task.

+0	t_status	task status
		x’00000000’ - TAVAIL - slot available
		x’00000001’ - TINUSE - slot in use
		x’00000002’ - TREADY - ready to run
		x’00000004’ - TACTIVE - currently running
		x’00000008’ - TNOACK - don’t ack message
		x’00000010’ - TFUNC - function completed
		x’00000020’ - TDECQU - restraint queue decremented
		x’00000040’ - TDOWN - tear task down when audit runs
+4	t_taskid	task identification
+8	*t_slpaddr	task sleep address
+c	*t_stack	pointer to task stack
+10	*t_msg	pointer to recvd message
+14	t_err	task error status
		x’00000001 - EPERM - Not super-user
		x’00000002 - ENOENT - No such file or directory
		x’00000003 - ESRCH - No such process
		x’00000004 - EINTR - Interrupted system call
		x’00000005 - EIO - I/O error
		x’00000006 - ENXIO - No such device or address
		x’00000007 - E2BIG - Arg list too long
		x’00000008 - ENOEXEC - Exec format error
		x’00000009 - EBADF - Bad file number
		x’0000000a - ECHILD - No children
		x’0000000b - EAGAIN - No more processes
		x’0000000c - ENOMEM - Not enough core
		x’0000000d - EACCES - Permission denied
		x’0000000e - EFAULT - Bad address
		x’0000000f - ENOTBLK - Block device required
		x’00000010 - EBUSY - Mount device busy
		x’00000011 - EEXIST - File exists
		x’00000012 - EXDEV - Cross-device link
		x’00000013 - ENODEV - No such device
		x’00000014 - ENOTDIR - Not a directory
		x’00000015 - EISDIR - Is a directory
		x’00000016 - EINVAL - Invalid argument
		x’00000017 - ENFILE - File table overflow
		x’00000018 - EMFILE - Too many open files
		x’00000019 - ENOTTY - Not a typewriter
		x’0000001a - ETXTBSY - Text file busy

		x’0000001b - EFBIG - File too large
		x’0000001c - ENOSPC - No space left on device
		x’0000001d - ESPIPE - Illegal seek
		x’0000001e - EROFS - Read only file system
		x’0000001f - EMLINK - Too many links
		x’00000020 - EPIPE - Broken pipe
		x’00000021 - ETEMP - Temped file
		x’00000022 - ENOTRAP
		x’00000023 - ENOMSG
		x’00000024 - ENOALOC
		x’00000026 - EFSAUD
		x’00000027 - EFIRST
		x’00000028 - ENOMOVE
		x’00000029 - ENOEXT
		x’0000002a - EPATH
		x’0000002b - ETABLE
		x’0000002c - EFUNC
		x’0000002d - EFMAUD
+18	t_tstamp	task start time stamp
+1c	t_patchcnt	system patch cnt at task start
+20	t_proc	temp storage of proc number when device driver created or opened
+24	t_ncblks	temp storage of #/contig blks
+28	t_offset	temp storage of directory offset
+2c	t_dent	temp storage of directory entry
+2c	d_ino	inode number
+2e	d_name[ ]	last component of pathname
+3c	*t_pdir	temp storage of ptr to incore directory inode
+40	t_start	used for performance task timing
+44	t_dbuf[DIRSIZ]	temp storage of pathname component
+52	t_sdfpath	special device file pathname for ECD access
+92	t_bufreq	buffered io flag
		x’00’ - no
		x’01’ - yes
+93	t_extra[16]	structure padding
+a4	t_stackarea	task private stack (remainder of page)

Release 6 Hexadecimal Offset Charts

8

Contents

Release 6 Hexadecimal Offset Charts

8

Kernel Address Space

This section lists the control block structures in the kernel address space. The lists name each field and give the hexadecimal offset to the field from the beginning of the structure.

Structure: ative - length: 0x1c

Source: os/kern/ative.h

Location: Found in the kernel’s data segment in an array called ‘‘atchtbl[ ]’’. Address is variable from load to load.

Use: Maintains the necessary information for the kernel to dispatch processes attached to interrupts.

+0	at_prc	attached process number
+4	*at_ent()	attached process entry point
+8	at_psw	psw value for interrupt
+c	at_ident	interrupt id
+10	at_sbr	segment base register value
+14	at_is	interrupt source
+15	at_ch	I/O channel
+16	at_dv	I/O device
+17	at_dummy	unused at this time
+18	at_status	status of the attach point x’00000001’ - AT_FREE x’00000010’ - AT_BUSY

Structure: dcte - length: 0x80
Source: head/dcte.h
Location: In the kernel’s address space at 0x140000 (may vary per head/va.h).
Use: Is the central source of process related information in the kernel and special processes.

+0	d_flag	flag word	
		0x00000001 - DF_LFAIL	
		0x00000002 - DF_LBOLT	
		0x00000004 - DF_PROFIL	
		0x00000008 - DF_RB	
		0x00000010 - DF_JC	
		0x00000020 - DF_TOUT	
		0x00000040 - DF_LOAD	
		0x00000080 - DF_READY	
		0x00000100 - DF_RLIST	
		0x00000200 - DF_SLEEP	
		0x00000400 - DF_REMOV	
		0x00000800 - DF_SWAP/DF_SUP	
		0x00001000 - DF_NOSWAP	
		0x00002000 - DF_KPRC	
		0x00004000 - DF_SYS	
		0x00008000 - DF_RTOR	
		0x00010000 - DF_STOR	
		0x00020000 - DF_STATIC	
		0x00040000 - DF_TMPNR	
		0x00080000 - DF_NOTERM	
		0x00100000 - DF_RUN	
		0x00200000 - DF_NOFIT	
		0x00400000 - DF_MSGHOG	
		0x00800000 - DF_DLMESSNTL	
		0x01000000 - DF_DLMNONSWP	
		0x02000000 - DF_UNXTERM	<R6.2 Only>
		0x02000000 - DF_PREEMPTED	<R6.3 through R6.6>
		0x02000000 - DF_RRTOUT	<R6.7 and Later>
		0x04000000 - DF_FLTMSG	
		0x08000000 - DF_MAXINTVL	
		0x10000000 - DF_DIOCHG	
		0x20000000 - DF_SUSPEND	
		0x40000000 - DF_DISP	
		0x80000000 - DF_DEAGED	<R6.7 and Later>
+4	*d_link	link to the next dcte on the dispatcher chain of the same execution level	

+8	*d_lblk	link for l_bolt chain	
+c	*d_stmlk	link for single time-out chain	
+10	*d_rtmlk	link for repetitive time-out chain	
+14	d_rtime	time interval for repetitive time-out request	
+18	*d_msg	pointer to the first message to this process	
+1c	*d_msgend	pointer to last message	
+20	d_stout	real-time value (msec) for single time-out request	
+24	d_rtout	real-time value (msec) for repetitive time-out request	
+28	d_evflag	event flags 0x00000001 - E_USR16 0x00000002 - E_USR15 0x00000004 - E_USR14 0x00000008 - E_USR13 0x00000010 - E_USR12 0x00000020 - E_USR11 0x00000040 - E_USR10 0x00000080 - E_USR9 0x00000100 - E_USR8 0x00000200 - E_USR7 0x00000400 - E_USR6 0x00000800 - E_USR5 0x00001000 - E_USR4 0x00002000 - E_USR3 0x00004000 - E_USR2 0x00008000 - E_USR1 0x00010000 - E_SYS16/E_SIGABN 0x00020000 - E_SIGCHAR 0x00040000 - E_SIGACK 0x00080000 - E_CHILD 0x00100000 - E_SYS12/E_CHACT 0x00200000 - E_AUD 0x00400000 - E_UTIL 0x01000000 - E_INIT 0x02000000 - E_ABORT 0x04000000 - E_QIT 0x08000000 - E_INT 0x10000000 - E_HUP 0x20000000 - E_MSG 0x40000000 - E_TIMEOUT 0x80000000 - E_WAKEUP	*
+2c	d_pn	process number	*
+30	d_pcbent		
+30	*d_pcbid	pcb segment id (kern or sup processes)	
+30	(*d_sproc)()	entry point to special process	
+34	d_sleep	sleep bit pattern	

+38	d_ucnt	user count
+3a	d_fcode	fault code
		0x00 - NOFLT
		0x10 - x80 reserved for *UNIX*® real-time reliable (RTR) operating system applications
		0x81 - OV_SOPOK
		0x82 - OV_SOPOVLD
		0x85 - OV_DLINCLR
		0x86 - OV_DLINOVLD
		0x87 - OV_DLOPCLR
		0x88 - OV_DLOPOVLD
		0x8b - OV_IOPOK
		0x8c - OV_IOPOVLD
		0x8d - OV_IOPBOOT
		0x8f - OV_DFCOK
		0x90 - OV_DFCOVLD
		0x91 - OV_FMOVCLR
		0x92 - OV_FMOVLD
		0x95 - OV_SUOVCLR
		0x96 - OV_SUOVLD
		0x97 - OV_TEOVCLR
		0x98 - OV_TEOVLD
		0x9e - OV_MEM1FULL *
		0x9f - OV_DCTOK
		0xa0 - OV_DCTOVLD
		0xa1 - OV_DCTCRIT
		0xa2 - OV_SDECLR
		0x99 - OV_KLCLR
		0x9a - OV_KLOCK
		0x9c - OV_MEM1CLR
		0x9d - OV_MEM1LOW
		0xa3 - OV_SDELOW
		0xa4 - OV_SDEPRFL
		0xa5 - OV_SWAPCLR
		0xa6 - OV_SWAPLOW
		0xa7 - OV_MEMCLR
		0xa8 - OV_MEMLOW
		0xa9 - OV_MEMKPFL
		0xac - OV_MSGOK
		0xad - OV_MSGLOW
		0xae - OV_MSGCRIT
		0xaf - OV_MSGOUT
		0xb0 - FLT_FULL_DIO
		0xb1 - FLT_DREP
		0xb2 - FLT_PICP

* UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

+3a	d_fcode	fault code 0xb3 - FLT_CCP 0xb4 - FLT_DRED 0xb5 - FLT_ADRD 0xb6 - FLT_PICD 0xb7 - FLT_CCD 0xb8 - FLT_NDREP 0xb9 - FLT_NPICP 0xba - FLT_NADRP 0xbb - FLT_MYC 0xbc - FLT_QMSAUD 0xbd - FLT_NONQMSAUD 0xbe - FLT_DUPLEX 0xbf - FLT_SIMPLEX 0xc0 - FLT_CFT 0xc1 - FLT_RSCOMP 0xc2 - FLT_SSCOMP 0xc3 - FLT_CMI 0xc4 - FLT_CMA 0xc5 - FLT_CMB 0xc6 - FLT_CMC 0xc7 - FLT_CMD 0xc8 - FLT_CMAN 0xc9 - FLT_UCLRMV 0xca - FLT_SSREQ 0xcb - FLT_SOFTSW 0xcc - FLT_CRMV 0xcd - FLT_TMOUT 0xd1 - FLT_PINV 0xd2 - FLT_PIND 0xd3 - FLT_SINV 0xd4 - FLT_SIND 0xd5 - FLT_BADOST 0xd6 - FLT_PROT 0xd7 - FLT_ADDR 0xd8 - FLT_PRIV 0xd9 - FLT_OPCD 0xda - FLT_STACK 0xe0 - FLT_PHASE0 0xe1 - FLT_SINIT 0xe2 - FLT_SCRIT 0xf0 - xff reserved for *UNIX* RTR operating system applications
+3b	d_chan	control channel
+3c	d_cprior	current priority for supervisor process process id for kernel process
+3d	d_iprior	initial priority for supervisor process execution level for kernel process

+3e	d_age	time in 1/2 sec units that the	<R6.2 through R6.3>
		process is waiting to be	<R6.2 through R6.3>
		scheduled (type: short)	<R6.2 through R6.3>
+3e	d_age	time in 1/2 sec units that the	<R6.4 and Later>
		process is waiting to be	<R6.4 and Later>
		scheduled (type: char)	<R6.4 and Later>
+3f	d_unused		<R6.4 and Later>
+40	d_pcode:11	pcode portion of the sup/kp utilid	
	d_ucode:11	pcode portion of the user utilid	
	d_spare:10	sure kill flags	
		0x000 - DS_TERMPID	<R6.2 through R6.5>
		0x000 - DS_TRMPID	<R6.6 and Later>
		0x001 - DS_UTERMPID	<R6.2 through R6.5>
		0x001 - DS_UTRMPID	<R6.6 and Later>
		0x002 - DS_TERMUID	<R6.2 through R6.5>
		0x002 - DS_TRMUID	<R6.6 and Later>
		0x004 - DS_UTERMUID	<R6.2 through R6.5>
		0x004 - DS_UTRMUID	<R6.6 and Later>
		0x008 - DS_TERMCLASS	<R6.2 through R6.5>
		0x008 - DS_TRMCLASS	<R6.6 and Later>
		0x010 - DS_UTERMCLASS	<R6.2 through R6.5>
		0x010 - DS_UTRMCLASS	<R6.6 and Later>
		0x020 - DS_UNIXTERM	
+44	d_class	process class flag	
		0x00000001 - DC_DLG	
		0x00000002 - DC_ESSEN	
		0x00000004 - DC_ULARP	
		0x00000008 - DC_CFT_T	
		0x00000010 - DC_CFT_D	
		0x00000020 - DC_CFT_S	
		0x00000040 - DC_CINIT	
		0x00000080 - DC_CFT_P	
		0x000000b8 - DC_CFT	
		0x00000100 - DC_ODIN	
		0x00000200 - DC_OMDB	<R6.2 through R6.4>
		0x00000400 - DC_SPARE1	<R6.2 Only>
		0x00000400 - DC_DAP	<R6.3 and Later>
		0x00000800 - DC_SPARE2	<R6.2 Only>
		0x00000800 - DC_SPARE	<R6.3 and Later>
		bits 12-31 reserved for applications	
+48	d_eent	event entry (psw and function)	
+48	pe_psw	processor status word	
+4c	pe_pa	entry point for program address	
+50	d_oent	ost entry vector (psw and function)	
+50	pe_psw	processor status word	
+54	pe_pa	entry point for program address	

+58	d_fent	fault entry vector (psw and function)	
+58	pe_psw	processor status word	
+5c	pe_pa	entry point for program address	
+60	d_sbr	segment base value	
+64	d_enable	enable flag for event entry	
+68	d_slptr	pointer to slist entry for supervisor	<R6.2 through R6.6>
		process, used by fltint operating	
		system trap (OST) for kernel	<R6.3 and Later>
		process: bits 0-25 reserved for PA	<R6.3 and Later>
		0x04000000 - DELFLTSI	<R6.3 and Later>
		0x08000000 - SEGMOD	<R6.3 and Later>
		0x10000000 - SYS_PHASE	<R6.3 and Later>
+6c	d_psize	current size of this process	
+70	d_audmap	audit flag for dcte audit	<R6.2 through R6.3>
+70	d_rdblk	# s_ticks process remains RB’d	<R6.4 and Later>
+71	d_aud1	audit spare	<R6.2 through R6.3>
+71	d_inmem	# s_ticks process remains in memory	<R6.4 and Later>
+72	d_msgcnt	message buffer usage count	
+74	d_start	process start time	
+78	d_otime	process time in ost code	
+7c	d_ptime	process time in process code	

Structure: dctext - length: 0x10 <R6.2 through R6.6>
0x20 <R6.7 and Later>

Source: head/dcte.h

Location: In the kernel's address space at 0x340000 (may vary per head/va.h).

Use: An extension of the dct entry which, like the dcte, contains process related information.

+0	de_dctndx	index of the associated dcte
+2	de_state	creation/termination/suspend states 0x0000 - No creation/termination/suspend in progress Termination: 0x0064 - use count decremented 0x006e - term_dct() called 0x0078 - GRASP informed 0x0082 - unlinked from dispatch 0x008c - atb flushed 0x0096 - ack msg created 0x00a0 - unlinked from slist 0x00aa - incarnation cnt bumped 0x00b4 - forwarded to CMGR 0x00be - forwarded to PMGR 0x00c8 - caps removed 0x00d2 - sup segs removed 0x00dc - kp segs removed 0x012c - core dump started 0x0136 - forwarded from PMGR to CMGR Creation: 0x01f4 - kp pcreat started 0x01fe - kp dcte linked 0x0208 - E_INIT sent 0x0258 - sp pcreat started 0x0262 - sp dcte linked 0x026c - sp pstart completed 0x0276 - sp execute started 0x02bc - fork started 0x02c6 - dupcaps sent to FMGR 0x02d0 - pfork2 started 0x02da - wakeup to child Suspend: 0x0320 - process suspended from execution
+4	de_tstamp	last major change in creation/termination also used for (s) maxintvl OST
+8	de_pmap1	public library bit map 1
+c	de_pmap2	public library bit map 2 0x00000001 - ECD 0x00000002 - PLM 0x00000004 - KCONFIG 0x00000008 - *UNIX* system 0x00000010 - CRAFT

+c	de_pmap2	public library bit map 2 0x00000020 - LLA incore 0x00000040 - LLA general	
+10	*de_s_fp	scheduler list forward pointer for supervisor process	<R6.7 and Later>
+14	*de_s_bw	scheduler list backward pointer for supervisor process	<R6.7 and Later>
+18	de_xtra1	spare field	<R6.7 and Later>
+1c	de_xtra2	spare field	<R6.7 and Later>

Structure: instat - length: 0x50

Source: head/instat.h

Location: Found on the interrupt stack for preempted processes and occasionally on a process's stack depending upon the activity of the process.

Use: Contains all system register values required to resume execution of a preempted process.

+0	i_pa	program address at interrupt
+4	i_psw	psw
+8	i_psbr	primary segment base register
+c	i_ssbr	secondary segment base register
+10	i_reg[0]	general purpose reg 0
+14	i_reg[1]	general purpose reg 1
+18	i_reg[2]	general purpose reg 2
+1c	i_reg[3]	general purpose reg 3
+20	i_reg[4]	general purpose reg 4
+24	i_reg[5]	general purpose reg 5
+28	i_reg[6]	general purpose reg 6
+2c	i_reg[7]	general purpose reg 7
+30	i_reg[8]	general purpose reg 8
+34	i_reg[9]	general purpose reg 9 (argument pointer)
+38	i_reg[10]	general purpose reg 10 (frame pointer)
+3c	i_reg[11]	general purpose reg 11 (stack pointer)
+40	i_reg[12]	general purpose reg 12
+44	i_reg[13]	general purpose reg 13
+48	i_reg[14]	general purpose reg 14
+4c	i_reg[15]	general purpose reg 15

Structure: kpcb - length: 0x800

Source: head/kpcb.h

Location: One segment in each kernel process address space. All kpcb segments can be located via the kernel's dispatcher control table (DCT) entries.

Use: Contains all process specific information not needed directly by the kernel. Of most significance is the segment table of the process which is used by the microcode to define the virtual address space of the process.

+ 0	k_utilid	utility id
+ 4	k-sgt	segment table
+800	k_sbr	segment base register value
+804	k_eent	event entry vector
+804	pe_psw	processor status word
+808	pe_pa	entry point for program address
+80c	k_fent	ost entry vector
+80c	pe_psw	processor status word
+810	pe_pa	entry point for program address
+814	k_oent	ost entry vector
+814	pe_psw	processor status word
+818	pe_pa	entry point for program address
+81c	k_profaddr	profiling address
+820	k_pn	process number
+824	k_parpn	parent process number
+828	k_name	ASCII name of process
+838	k_tident	message ident to send on process death
+83c	k_tflag	if !=0,send k_ttype msg on process death
+83d	k_ttype	message type to send on process death
+83e	k_chan	control channel number
+83f	k_pcbpg	number of pages in this kpcb
		* The following declarations indicate spare fields to * allow field update of new process related information
+840	k_c0	spare characters
+848	k_s0	spare shorts
+850	k_i0	spare ints
+860	k_u0	spare unsigned for flag Kernel Process Segment List
+868	k_size	largest active segment number
+86a	k_sglsz	maximum segment configuration
+86c	k_seglist	segment list
+86c	k_segflg	
	kf_segndx:9	
	kf_segflg:23	segment flag word (struct ksegf)
		x'00000000' - KF_FREE x'..000001' - KF_EXEC

		x'..000002' - KF_WRT x'..000004' - KF_RD x'..000008' - KF_STK x'..000010' - KF_PWRT x'..000020' - KF_SHARE x'..000040' - KF_IOMAP
+870	k_segid	segment id

Structure:	kvt - length: 0x234 <R6.2 and R6.3 Only> 0x23C <R6.4 Only> 0x254 <R6.5 and R6.6 Only> 0x250 <R6.7 and Later>	* *
Source:	head/kvt.h	*
Location:	Found in the kernel's data segment in an external declaration called ''Kvt''. The exact address will vary from load to load.	| *
Use:	Contains all spy package metering data.	

For <R6.2 and R6.3 Only>:

+0	a_dctmem	virtual address of the 1st dcte
+4	a_dctpa	physical address of the 1st dcte
+8	a_depa	physical address of the 1st dctext
+c	dctcnt	total number of dcte's
+10	a_dctfree	address of the free dct count
+14	a_usrdct	address of pointer to current runner
+18	a_nxtdct	address of pointer to next loading proc
+1c	a_dispq	address of the dispatch queues
+20	a_portmem	virtual address of the 1st port
+24	a_portpa	physical address of the 1st port
+28	portcnt	total number of ports
+2c	a_stckpa	physical address of the kernel stack
+30	stacksize	size of the kernel stack in bytes
+34	a_msgpa	physical address of the 1st message
+38	a_mepa	physical address of 1st msg extender
+3c	msgcnt	total number of message buffers
+40	a_msgfree	address of free message block count
+44	a_Ovldfg	address of the message buffer overload flag
+48	pdecnt	total number of physical pages
+4c	a_pdefree	address of number of free pages
+50	pgecnt	total number of page tables
+54	a_pgefree	address of number of free page tables
+58	a_sdemem	virtual address of the 1st sde
+5c	a_sdepa	physical address of the 1st sde

+60	sdecnt	total number of sde’s
+64	a_sdefree	address of free sde count
+68	a_disksize	address of the disk swap size
+6c	a_diskfree	address of the free disk swap blks
+70	a_swapsize	address of the swap size
+74	a_swapmin	address of the swap size minimum
+78	a_Swapis	address of the segments swapped in count
+7c	a_Swapos	address of the segments swapped out count
+80	a_Swapib	address of the bytes swapped in count
+84	a_Swapob	address of the bytes swapped out count
+88	Sktime	time in kernel
+c8	Skptime	time in kernel processes
+108	Sstime	time in supervisor processes
+110	Sutime	time in user processes
+114	a_Tidle	address of time in idle loop
+118	a_prevtod	address of last tod clock tick
+11c	sdis_lev	supervisor and user processes
+15c	sdis_dif	dispatching statistics
+19c	pcra_cnt	number of processes created
+1a0	pkil_cnt	number of processes killed
+1a4	nkost	address of kernel ost counts
+1a8	nKPost	address of kernel processes activity array
+1ac	nsupost	address of supervisor ost counts
+1b0	nSUPost	address of level 2 activity indicator
+1b4	nuost	address of user ost data
+1b8	nint	address of interrupt usage data
+1bc	npir	address of pir usage data
+1c0	nkprocess	address of kernel process pid array
+1c4	a_clientdct	address of the kernel process trapping chain
+1c8	a_iptpnum	address of pnum of last kp trapped to by a sup
+1cc	ldfails	number of load fails
+1d0	ldreqs	number of load requests
+1d4	nofits	number of no fit processes
+1d8	mxfails	number of times S_MXFAIL conditions met
+1dc	rbcnt	number of roadblocked processes swapped

+1e0	actcnt	number of active processes swapped
+1e4	unused	reserved for field update additions

For <R6.4 and Later>:

+0	a_dctmem	virtual address of the 1st dcte
+4	a_dctpa	physical address of the 1st dcte
+8	a_depa	physical address of the 1st dctext
+c	dctcnt	total number of dcte's
+10	a_dctfree	address of the free dct count
+14	a_usrdct	address of pointer to current runner
+18	a_nxtdct	address of pointer to next loading proc
+1c	a_dispq	address of the dispatch queues
+20	a_portmem	virtual address of the 1st port
+24	a_portpa	physical address of the 1st port
+28	portcnt	total number of ports
+2c	a_stckpa	physical address of the kernel stack
+30	stacksize	size of the kernel stack in bytes
+34	a_msgpa	physical address of the 1st message
+38	a_mepa	physical address of 1st msg extender
+3c	msgcnt	total number of message buffers
+40	a_msgfree	address of free message blok count
+44	a_Ovldfg	address of the message buffer overload flag
+48	pdecnt	total number of physical pages
+4c	pde1cnt	total number of physical pages module 1
+50	a_pdefree	address of number of free pages
+54	a_pde1free	address of number of free pages module 1
+58	pgecnt	total number of page tables
+5c	a_pgefree	address of number of free page tables
+60	a_sdemem	virtual address of the 1st sde
+64	a_sdepa	physical address of the 1st sde
+68	sdecnt	total number of sde's
+6c	a_sdefree	address of free sde count
+70	a_disksize	address of the disk swap size
+74	a_diskfree	address of the free disk swap blocks

+78	a_swapsize	address of the swap size	
+7c	a_swapmin	address of the swap size minimum	
+80	a_Swapis	address of the segments swapped in count	
+84	a_Swapos	address of the segments swapped out count	
+88	a_Swapib	address of the bytes swapped in count	
+8c	a_Swapob	address of the bytes swapped out count	
+90	Sktime	time in kernel	
+d0	Skptime	time in kernel processes	
+110	Sstime	time in supervisor processes	
+118	Sutime	time in user processes	
+11c	a_Tidle	address of time in idle loop	
+120	a_prevtod	address of last tod clock tick	
+124	sdis_lev	supervisor and user processes	
+164	sdis_dif	dispatching statistics	
+1a4	pcra_cnt	number of processes created	
+1a8	pkil_cnt	number of processes killed	
+1ac	nkost	address of kernel ost counts	
+1b0	nKPost	address of kernel process activity array	
+1b4	nsupost	address of supervisor ost counts	
+1b8	nSUPost	address of level 2 activity indicator	
+1bc	nuost	address of user ost data	
+1c0	nint	address of interrupt usage data	
+1c4	npir	address of pir usage data	
+1c8	nkprocess	address of kernel process pid array	
+1cc	a_clientdct	address of the kernel process trapping chain	
+1d0	a_iptpnum	address of pnum of last kp trapped to by a sup	
+1d4	ldfails	number of load fails	
+1d8	ldreqs	number of load requests	
+1dc	nofits	number of no fit processes	
+1e0	mxfails	number of times S_MXFAIL conditions met	
+1e4	rbcnt	number of roadblocked processes swapped	
+1e8	actcnt	number of active processes swapped	
+1ec	unused	reserved for field update additions	<R6.4 Only>
+1f8	rt_overld	real time overload state	<R6.5 and <R6.6 Only>

+1fc	u_occ	*UNIX* system occupancy	<R6.5 and R6.6 Only>
+200	u_occ_toff	*UNIX* system occupancy turn off value	<R6.5 and R6.6 Only>
+204	unused	reserved for field update additions	<R6.5 and R6.6 Only>
+1ec	idle_cnt	number of idle loop entries	<R6.5 and Later>
+1f0	rr_timer	current value of Round Robin timer	<R6.5 and Later>
+1f4	hp_proc	number of high prior proc active and on swap dev	<R6.5 and Later>
+1f8	u_occ	*UNIX* system occupancy	<R6.7 and Later>
+1fc	u_occ_toff	*UNIX* system occupancy turn off value	<R6.7 and Later>
+200	unused	reserved for field update additions	<R6.7 and Later>

Structure: msghdr - length: 0x14

Source: head/msghdr.h

Location: Contained in each allocated message buffer in the kernel's message segment (shared with kernel processes). This segment (KMSG) is located at 0x620000 but may vary depending upon changes to head/va.h. Supervisors use local copys of this structure.

Use: Contains message control information used by the kernel as well as the sending and receiving processes.

+0	*ms_link	ptr to next msg on input queue (maybe)
+4	ms_from	sending process number
+8	ms_to	receiving process number
+c	ms_nblks	msg size in 64 byte blocks (max = 7)
+d	ms_flags	msg header flags
		x'01' - MS_CAP
		x'02' - MS_ULOCK
		x'04' - MS_NACK
		x'08' - MS_ALOC
+e	ms_type	message type
		x'00' FM_BADMSG MSMIN
		x'01' FM_READ P_CREAT IOREAD
		x'02' FM_WRITE IOWRITE
		x'03' FM_OPEN IOOPEN
		x'04' FM_CLOSE P_INIT IOCLOSE
		x'05' FM_EXEC P_WAIT
		x'06' FM_FORK P_INMEM
		x'07' DELCAP P_UNINMEM
		x'08' FM_CREAT ADDCAP
		x'09' FM_LINK MSTERM
		x'0a' FM_UNLINK MSGROW
		x'0b' FM_UTIME MSLOAD
		x'0c' FM_CHDIR MSPLOCK
		x'0d' FM_INIT MSKADD
		x'0e' FM_MKNOD MSKRMV
		x'0f' FM_CHMOD MSCMPCT
		x'10' FM_CHOWN
		x'11' FM_SYNC
		x'12' FM_STAT
		x'13' FM_SIZE
		x'14' FM_FSTAT
		x'15' FM_MOUNT
		x'16' FM_UMOUNT
		x'17' FM_MOVE
		x'18' FM_ALLOC
		x'19' FM_MNTSTAT
		x'1a' FM_TASKAUD
		x'1b' FM_UNFORK
		x'1c' FM_ACCESS
		x'1d' FM_USTAT

		x'1e' FM_SEGCODE
		x'1f' FM_TEMP
+e	ms_type	message type
		x'20' FM_BACKOUT
		x'21' FM_PERM
		x'22' FM_MV
		x'23' FM_BUFRD
		x'24' FM_BUFWRT
		x'25' FM_LSEEK
		x'26' FM_PIPE
		x'27' FM_ATOMSW
		x'28' FM_PERF
		x'29' FM_DUPCAP
		x'2a' FM_ATOMPERM
		x'2b' FM_ATOMBACK
		x'2e' FM_FSCAUD
		x'2f' FM_FSLAUD
		x'30' FM_FSBAUD
		x'31' FM_AUD
		x'32' IOCANCEL
		x'33' FM_FUAUD
		x'34' FM_SHRTRD IOSHRTRD
		x'35' FM_SHRTWRT IOSHRTWRT
		x'47' FM_LAST
		x'61' IOSYSDLM
		x'64' MSFAULT
		x'65' MSRCVMSG
		x'6c' T_LIBMSG
		x'7e' ECDCHNG
		x'fb' WAITMSG
		x'fc' TRCSND
		x'fd' TRCRCV
		x'fe' MSSIG
		x'ff' MSACK MSMAX
+f	ms_stat	message status
		x'00' MSNOERR
		x'3f' BADTYPE
		x'40' SYSERR
		x'e0' MSOLD
		x'e1' MBOLOAD
		x'ff' MSDEAD MSPFAIL
+10	ms_size	msg size in bytes
+12	ms_otype	original type before ack
+13	ms_seqnum	message sequence number
+14	ms_ident	message id used by sender
		x'fffffffc' - TRCMSG
		x'fffffffd' - USRMSG
		x'fffffffe' - SIGMSG
		x'ffffffff' - UNXMSG

Structure: pcb - length: 0x19f4

Source: head/pcb.h

Location: One segment in each supervisor process address space. All pcb segments can be located via the kernel's dispatcher control table (DCT). The most currently running supervisor will have its pcb segment in the kernel's address space at 0x420000 (may vary per header va.h).

Use: Contains all supervisor specific information not needed directly by the kernel.

Offset	Field	Description
+0	p_sutilid	supervisor utility id
+4	p_ssgt	supervisor segment table
+800	p_spsbr	suervisor process psbr
+804	p_upsbr	user process psbr
+808	p_svect	starting entry vector
+808	pe_psw	processor status word
+80c	pe_pa	entry point for program address
+810	p_evect	event entry vector
+810	pe_psw	processor status word
+814	pe_pa	entry point for program address
+818	p_fvect	fault entry vector
+818	pe_psw	processor status word
+81c	pe_pa	entry point for program address
+820	p_ovect	ost entry vector
+820	pe_psw	processor status word
+824	pe_pa	entry point for program address
+828	p_profaddr	profiling address
+82c	p_pn	process number
+830	p_parpn	parent process number
+834	p_name	ASCII name of the process
+844	p_tident	message identification to send to parent on * death of process
+848	p_tflag	if !=0, send p_ttype message to parent on * death of process
+849	p_ttype	message type to send on process death
+84a	p_fcode	fault code
+84b	p_static	the process has static scheduling priority

+84c	p_evopt	all or any option for event wait
+84d	p_crflag	time out occurred while in critical region
+84e	p_prior	initial priority
+84f	p_tocnt	number of elapsed time slices
+850	p_wait	scheduler flag x’00000000’ - P_DONTCARE x’00000001’ - P_INCORE x’FFFFFFFF’ - P_OUTCORE
+852	p_cwait	incremented when event is expected, * clear when event arrives, * and is intended for catching event before * the process is roadblocked
+854	p_chan	control channel number
+855	p_pcbpg	number of pages in this pcb. The following declaration indicates spare fields to * allow field update of new process related information * that will be used by the KERNEL.
+856	p_c0	spare chars
+85c	p_s0	spare shorts
+860	p_i0	spare ints
+868	p_u0	unsigned for flags
+86c	p_fup	field update patch count
+870	p_ttg	time to go on the process time slice
+874	p_slice	time slice of the process
+878	p_ktime	time spent in kernel
+87c	p_kptime	time spent in kernel process
+880	p_stime	time spent in supervisor mode
+884	p_runtime	accumulated runtime, cleared each 60ms
+888	p_evflg	process event flags x’00000001’ - E_USR16 x’00000002’ - E_USR15 x’00000004’ - E_USR14 x’00000008’ - E_USR13 x’00000010’ - E_USR12 x’00000020’ - E_USR11 x’00000040’ - E_USR10 x’00000080’ - E_USR9 x’00000100’ - E_USR8 x’00000200’ - E_USR7 x’00000400’ - E_USR6

		x’00001000’ - E_USR4
		x’00002000’ - E_USR3
		x’00004000’ - E_USR2
		x’00008000’ - E_USR1
		x’00010000’ - E_SYS16
		x’00020000’ - E_SYS15
		x’00040000’ - E_SYS14
		x’00080000’ - E_SYS13
		x’00100000’ - E_SYS12
		x’00200000’ - E_SYS11
		x’00400000’ - E_UTIL
		x’00800000’ - E_RTIMEOUT
		x’01000000’ - E_INIT
		x’02000000’ - E_ABORT
		x’04000000’ - E_QIT
		x’08000000’ - E_INT
		x’10000000’ - E_HUP
		x’20000000’ - E_MSG
		x’40000000’ - E_TIMEOUT
		x’80000000’ - E_WAKEUP
+88c	p_evwait	mask for event wait flags
		x’00000000’ - P_EWANY
+890	p_evmsk	process event mask
+894	p_evpsd	psd save area at event entry
+894	ps_psw	processor status word
+898	ps_pa	program address
+89c	p_fpsd	psd save area at fault interrupt
+89c	ps_psw	processor status word
+8a0	ps_pa	program address
+8a4	p_topsd	psd save area at preemption or time-out
+8a4	ps_psw	processor status word
+8a8	ps_pa	program address
+8ac	p_initsp	initial value of the stack pointer
+8b0	p_tosave	register save area at preemption/ time-out
+8f0	p_state	
+8f0	i_pa	
+8f4	i_psw	
+8f8	i_psbr	
+8fc	i_ssbr	
+900	i_reg	
+940	p_clist	capability list
+940	cp_owner	owner process
+944	cp_cap	capability
+9f0	p_disptch	number of dispatches since creation
+9f4	p_swapcnt	number of swap outs since creation

		Supervisor Process Segment List
+9f8	p_size	largest active segment number
+9fa	p_sglsz	maximum segment configuration
+9fc	p_seglist	segment list
+9fc	p_segflg	
+9fc	p_segndx:9	
	p_segflg:23	segment flag word (struct ssegf)
		x’00000000’ - SF_FREE
		x’..000001’ - SF_EXEC
		x’..000002’ - SF_WRT
		x’..000004’ - SF_RD
		x’..000008’ - SF_STK
		x’..000010’ - SF_PWRT
		x’..000020’ - SF_SHARE
		x’..000040’ - SF_NOLD
		x’..000080’ - SF_NONSW
		x’..000100’ - SF_SBIT
		x’..000200’ - SF_UBIT
		x’..000400’ - SF_NXT
		x’..000800’ - SF_NN
+a00	p_segid	segment id

Structure: psw - length: 0x04

Source: head/psw.h

Location: The current psw is located in the ‘‘PSW’’ special register, preempted psw values are found on the interrupt stack, entry point psws are found in the DCT for kernel processes and in PCB segments for supervisors, and attachable entry point psws are found in the atchtbl[ ] array (of ative structures) in the kernel’s data segment.

Use: Used by the microcode to determine the required system environment for the currently running process.

+0	w_mode:2	processor mode x’0.......’ - W_MKRN x’4.......’ - W_MKP x’8.......’ - W_MSUP x’c.......’ - W_MUSR
+0.5	w_exlev:6	execution level x’.0.......’ - level 0 x’.1.......’ - level 1 " " " " x’.f.......’ - level f
+1	w_prvlg:4	privilege bits x’..1.....’ - W_SETEX x’..2.....’ - W_NMIO x’..4.....’ - W_SYSIO x’..8.....’ - W_WPSW
+1.5	w_emcntl:4	emulation control
+2	w_ssbr:3	secondary sbr
	w_psbr:3	primary sbr
	w_flag:6	bit flags b’..00,0001....’ - W_KSTK b’..00,0010....’ - W_SPARE b’..00,0100....’ - W_ISTK b’..00,1000....’ - W_MMON b’..01,0000....’ - W_SRC b’..10,0000....’ - W_DEST
+3.5	w_cond:4	condition codes x’.......1’ - W_CBIT x’.......2’ - W_NBIT x’.......4’ - W_VBIT x’.......8’ - W_ZBIT

Structure: sde - length: 0x20

Source: head/sde.h

Location: Found in the kernel's address space starting at segment index 13 (0x1a0000). This address is dependent upon header va.h and may move from load to load.

Use: Memory management routines use the SDT to map all segments known to the system, either in memory or on the swap device.

+0	*s_pgtptr	virtual address of the page table
+4	*s_link	link for swappable segment sde's or free sde's
+8	s_plkcnt	process lock count
+9	s_lkcnt	I/O lock count
+a	s_nswcnt	nonswap count
+c	s_active	number of processes on the don't swap list, that have allocated the segment
+e	s_users	total number of processes that have allocated the segment
+10	s_lstpgsz	number of bytes used in the last page
+12	s_tlpg	total number of pages
+13	s_inmmpg	total number of pages in main memory
+14	s_stat	segment status word x'00000000' - SS_FREE x'..000002' - SS_BROKE x'..000004' - SS_WRT x'..000008' - SS_GBCLT x'..000010' - SS_PURGE x'..000020' - SS_REMOV x'..000040' - SS_UTLY x'..000080' - SS_IOFAIL x'..000100' - SS_IOIN x'..000200' - SS_IOOUT x'..000400' - SS_LOCK x'..000800' - SS_NONSW x'..001000' - SS_ALT x'..002000' - SS_NEXT x'..004000' - SS_ACT x'..008000' - SS_PLOCK x'..010000' - SS_NSWSP x'..020000' - SS_BRKDN x'..040000' - SS_KPCB x'..080000' - SS_SPCB x'..100000' - SS_BLOCK x'..200000' - SS_NEW x'..400000' - SS_PGPRT x'..800000' - SS_ALLOC x'10000000' - SS_ONFL

+18	s_swapaddr	block number of the starting point on the swap device
+1c	sde_name	segment name

Kboot Address Space

This section lists the control block structures in the kboot address space. The lists name each field and give the hexadecimal offset to the field from the beginning of the structure.

Structure: bootab - length: 0x1060

Source: head/sgenbt.h

Location: In the kernel's address space somewhere near the end of the text segment under the external name ‘‘kbootab’’.

Use: Contains the mapping information used by kboot to create the segments and processes making up the boot.

+0	bt_ecdversion	ECD version number
+4	bt_nseg	number of valid entries in the bt_seg[ ] array
+6	bt_nprc	number of valid entries in the bt_prc[ ] array
+8	bt_ksdx[ ]	indices of the kernel's bt_seg[ ] entries
+40	bt_seg[ ]	boot image segment descriptors - each is a bsegdes structure
+a40	bt_prc[ ]	boot image process descriptors - each is a bprcdes structure
+bc0	bt_kparm	kernel dynamic memory parameters - see the btkparm structure
+c04	bt_npaths	number of processes to be pcreated
+c06	bt_nlibs	number of public libraries to be loaded
+c08	bt_upath[ ]	pathnames of pcreated processes
+c60	bt_libpath[ ]	pathnames of boot public libraries

Structure: bprcdes - length: 0x18

Source: head/sgenbt.h

Location: In the kernel’s address space somewhere near the end of the text segment under the external name ‘‘kbootab’’ which contains an array of bprcdes structures.

Use: Contains the mapping information used by kboot to create the processes making up the boot.

+0	pd_pnum	fixed process number
+4	pd_class	process class
+8	pd_pcbsdx	index of the process’s (k)pcb segment in bt_seg[ ]
+a	pd_flags	process flags 0x00000001 - kernel process 0x00000002 - supervisor 0x00000200 - shares segment with child 0x00000400 - shares segment with parent 0x00000800 - process being notified 0x00001000 - dlm essential 0x00002000 - static 0x00004000 - noterm
+c	pd_prior	supervisor initial priority or kernel process execution level
+e	pd_spare	unused
+10	pd_nseg	number of boot image segments
+12	pd_nints	number of interrupts to attach
+14	pd_libflags	public library bit map

Structure: bsegdes - length: 0x14

Source: head/sgenbt.h

Location: In the kernel’s address space somewhere near the end of the text segment under the external name ‘‘kbootab’’ which contains an array of bsegdes structures.

Use: Contains the mapping information used by kboot to create the segments making up the boot.

+0	sd_kbva	virtual address in the kboot address space of the segment initial image
+4	sd_segsize	size of the segment in bytes
+8	sd_segndx	true segment index of the segment
+a	sd_users	number of processes using the segment
+c	sd_segflgs	segment flag word 0x00000007 - segment protection flags 0x00000200 - LDP ’shared’ segment 0x00000400 - LDP ’common’ option 0x00000800 - PAS segment 0x00002000 - ECD segment 0x00008000 - segment is part of the kernel
+10	*sd_segid	segment ID of the true segment. This field is filled in by kboot.

Structure: btkparm - length: 0x44

Source: head/sgenbt.h

Location: In the kernel’s address space somewhere near the end of the text segment under the external name ‘‘kbootab’’. This structure is contained in kbootab.

Use: Defines the kernel’s dynamic memory segments.

+0	km_nmsg	number of message blocks to be allocated
+2	km_nport	size of the port segment (in words)
+4	km_nprc	number of dct entries
+6	km_nsegecd	number of ecd segments
+8	km_nseg	number of SDT entries
+c	km_npgt	number of page tables
+10	km_npage	number of PDT entries
+14	km_istkb	size of the interrupt stack (in bytes)
+18	km_kstkb	size of the kernel stack (in bytes)
+1c	km_swstart	starting block of swap area
+20	km_swblks	size of swap area (in blocks)
+24	km_swmin	swap size of largest supervisor (in pages)
+28	km_intlen	initialization interval
+2c	km_maxlevs[ ]	application phase levels
+30	km_PASndx	segment index of low PAS
+34	km_1PASndx	segment index of high PAS
+38	km_PASdm	PAS dump/nodump flag
+3c	km_part	partition boundary between mod 0 and mod1
+40	km_sched	scheduler’s time-out value

File Manager Address Space

This section lists the control block structures in the file manager address space. The lists name each field and give the hexadecimal offset to the field from the beginning of the structure.

Structure: bdevtab - length: 0x4c

Source: head/fmgr/fmgr.h

Location: An externally declared array called ‘‘bdevtab’’.

Use: One entry for each block device driver (indexed by dcn). Internal file manager buffers are chained off an array of pointers using a hash selection of ’and’ing 0x7 to the block number.

+0	d_proc	process number of the driver
+4	boflag	if nonzero then the driver process is to get an open or close message with each open or close request. If zero then the driver only gets one open and one close.
+8	d_bchain[8]	buffer device chain pointer array
+8	b_forw	forward chain pointer
+c	b_back	backward chain pointer
+48	b_extra[4]	unused

Structure: buf - length: 0x40 *

Source: os/fmgr/head/buf.h

Location: An externally declared array called ‘‘buf’’. Free buffers are chained off of ‘‘bfreelist’’ (also a buf structure). Buffers associated with a device are chained off of a bdevtab entry. Buffers explicitly associated with no device are chained off of ‘‘bfreelist’’ (another chain).

Use: Each buf structure (2*NTASKS+4) controls an I/O buffer.

+0	*b_forw	buf pointer headed by bdevtab
+4	*b_back	buf pointer headed by bdevtab
+8	b_flags	buffer flags 0x00000000 - B_WRITE - non-read pseudo flag 0x00000001 - B_READ - read flag 0x00000002 - B_DONE - I/O complete 0x00000004 - B_ERROR - I/O error 0x00000008 - B_BUSY - buffer in use (locked) 0x00000030 - B_XMEM - memory extension (unused) 0x00000040 - B_WANTED - buffer wanted (task asleep waiting) 0x00000000 - B_AGE - delayed write for correct aging (controls placement on available queue) 0x00000100 - B_ASYNC - no wait for completion 0x00000200 - B_DELWRI - delayed write (holds buffer between uses) 0x00000400 - B_IO - I/O outstanding on this buf 0x00000800 - B_MOUNT - this buffer contains the superblock superblock of a mounted file system 0x00001000 - B_FSAUD - this buffer is being used by by the file system audit
+c	*av_forw	buf pointer headed by bfreelist
+10	*av_back	buf pointer headed by bfreelist
+14	b_mdct	device mdct-rid
+18	b_dcn	device major number
+1a	b_part	device partition
+1c	b_un	union
+1c	b_addr	address of actual buffer
+1c	*b_words	pointer to words for clearing
+1c	*b_filsys	pointer to superblock
+1c	*b_dino	pointer to block in ilist
+1c	*b_daddr	pointer to indirect block
+20	b_blkno	block # on device
+24	*b_mptr	ptr to mount table entry (null unless B_MOUNT set)
+28	b_taskid	taskid information (null unless B_BUSY set)

+2c	b_tstamp	time when io started (2 minutes from this time result in automatic task teardown)
+30	b_extra[16]	structure padding

Structure: cap - length: 0x18

Source: os/fmgr/head/cap_tbl.h

Location: An externally declared array called ‘‘cap_tbl’’.

Use: One is maintained for each open or fork.

+0	c_cap	capability
+4	c_pid	client process number
+8	*c_fptr	ptr to corres. file table entry
+c	*c_cptr	ptr to next capability for this file (cap table entries are chained only on forks off of the same open, each points to the same file table entry)
+10	c_tstamp	time c_pid was first noted as invalid

Structure: cpmsghdr - length: 0x28

Source: head/cpmsghdr.h, see also head/fmgr/....

Location: Message buffer formats found in the message segment. Pointers to dequeued messages will be found in ‘‘tasktab’’ or one of the delayed_q’s.

Use: The means in which requests are made to the file manager. All message formats begin with a capability message header.

+0	cpm_mshd	standard message header (msghdr.h)
+0	ms_link	link to next message (delay queues only)
+4	ms_from	sending process
+8	ms_to	receiving process
+c	ms_nblks	msg size in blocks
+d	ms_flags	msg flags
+e	ms_type	message type
+f	ms_stat	message status
+10	ms_size	message size in bytes
+12	ms_otype	original message type
+13	ms_seqnu	unused
+14	ms_ident	message identifier
+18	cpm_mc	capability field
+18	mc_num	capability number
+1c	cp_owner	owner process
+20	cp_cap	capability bits 0 -> 7: i_use count bits 8 -> 15: permissions bits 16 -> 31: file table index
+24	cpm_guid	group/user id
	type 0x01	FM_READ - read file
		request:
+28	fm_iosid	segment-id
+30	fm_iobyoff	byte offset into segment
+34	fm_iocnt	number of bytes for I/O
+38	fm_ioblk	block number for I/O
+40	fm_iores	remaining bytes to be transferred
		reply:
+28	ret0	not used
+2c	ret1	not used
+30	fm_aiocnt	no bytes transferred
+34	fm_iortry	memory manager used field
+38	fm_err0	not used
+3c	fm_err1	error return from driver

	type 0x02	FM_WRITE - write file
		see type 0x01 request and reply
	type 0x03	FM_OPEN - open file
		request:
+28	fm_ocfoff	offset to name
+2c	fm_ocmode	mode for open or create
		0x000 - OP_READ - open: read 0x001 - OP_WRITE - open: write 0x002 - OP_RW - open: read/write 0x001 - CR_XBYOT - create: execution by others 0x002 - CR_WBYOT - create: write by others 0x004 - CR_RBYOT - create: read by others 0x008 - CR_XBYG - create: execute by group 0x010 - CR_WBYG - create: write by group 0x020 - CR_RBYG - create: read by group 0x040 - CR_XBYOW - create: execute by owner 0x080 - CR_WBYOW - create: write by owner 0x100 - CR_RBYOW - create: read by owner 0x200 - CR_SVTXAX - create: save text after execute 0x400 - CR_SGIDX - create: set group id on execute 0x800 - CR_SUIDX - create: set user id on execute
+30	fm_nopcr[ ]	filename
		reply:
+28	fm_ocapno	capability number (-1 for kernel process opens) (if pipe - cap num of read end)
+2c	fm_otype	type of file (if pipe - cap num of write end)
+30	fm_procid	process number of device file
+34	fm_unused	unused
+38	fm_odevid	mdct and partition
+40	fm_ofilsiz	file size
	type 0x04	FM_CLOSE - close file
+28	fm_usecnt	number of file descriptors using inode
+2c	fm_clmode	close mode
	type 0x05	FM_EXEC - open file for execution
		request:
+28	fm_off	filename offset
+2c	fm_name[ ]	filename
		reply:
+28	fm_ecapno	capability number
+2c	fm_segname	unique segment name

	type 0x06	FM_FORK - increment count for open file
+28	fm_fkcapc	number of capabilities
+2c	fm_fkchpid	child process number
+30	fm_fkchpid	fcount increment/decrement
+34	fm_fkmce[ ]	list of capnums and caps
	type 0x08	FM_CREAT - create file
		see type 0x03 request and reply
	type 0x09	FM_LINK - link to a file
+28	fm_loff	offset to existing pathname
+2c	fm_loff1	offset to link name
+30	fm_nlink[ ]	existing and link names
	type 0x0a	FM_UNLINK - remove link from file
		see type 0x05 request
	type 0x0b	FM_UTIME - modify date of file
+28	fm_utoff	filename offset
+2c	fm_uflag	flag
		>>>>>>>> what values <<<<<<<<
+30	fm_utime	utime structure
+30	f_atime	new access time
+34	f_mtime	new modify time
+38	f_ctime	new create time
+3c	fm_nutime[ ]	filename
	type 0x0c	FM_CHDIR - change directory
		request:
		see type 0x05 request
		reply:
+28	fm_chcapno	capability number
	type 0x0d	FMINIT - file manager initialization
+28	fm_idevid	mdct and partition of root device
+34	fm_idcnid	major device number root device
+38	fm_ipnum	root device process number
	type 0x0e	FM_MKNOD - make a node
+28	fm_mkoff	offset to name
+2c	fm_mkmode	permissions and file type
+30	fm_mkdvid	mdct and partition
+38	fm_mkdcn	major device number
+3a	fm_nmknod[ ]	name of the file
	type 0x0f	FM_CHMOD - change mode of file
+28	fm_chmoff	filename offset
+2c	fm_chmode	new mode of file
+30	fm_nchmod[ ]	filename
	type 0x10	FM_CHOWN - change owner of file
+28	fm_choff	filename offset
+2c	fm_uown	user id of new owner
+30	fm_gown	group id of new owner

+34	fm_nchown[ ]	filename
	type 0x11	FM_SYNC - update file systems on secondary
		capability header only
	type 0x12	FM_STAT - get file status
		request:
		see type 0x05 request
		reply:
+28	fm_stat	‘‘stat’’ structure
+28	st_dev	device number
+2c	st_ino	inode number
+30	st_mode	file mode (see mode field of inode)
+32	st_nlink	link count
+34	st_uid	user id
+36	st_gid	group id
+38	st_rdev	special file device name
+3c	st_size	length in bytes
+40	st_atime	time last accessed
+44	st_mtime	time last modified
+48	st_ctime	time created
	type 0x13	FM_SIZE - get file size
+28	fm_fsize	size of file (reply only)
+2c	fm_fssize	total amount allocated contiguous (reply only)
	type 0x14	FM_FSTAT - get status of open file
		request:
		capability header only
		reply:
		see type 0x12 reply
	type 0x15	FM_MOUNT - mount file system
+28	fm_moff	offset to device name
+2c	fm_moff1	offset to mount point name
+30	fm_mronly	action flags
		0x00000001 - read only access
		0x00000002 - audit the file system
+34	fm_nmount[ ]	device and mount point names
	type 0x16	FM_UMOUNT - unmount file system
		see type 0x05 request

	type 0x17	FM_MOVE - move file to contiguous area
		request:
		see type 0x05 request
		reply:
+28	fm_fmblk	number of blocks available
	type 0x18	FM_ALLOC - allocate contiguous space
		request:
+28	fm_faoff	filename offset
+2c	fm_famode	permissions and file type
+30	fm_fasize	file size
+34	fm_nfall[ ]	filename
		reply:
+28	fm_facapno	capability number
+2c	fm_fatype	type of file
+30	fm_faprocid	not used
+34	fm_fachan	not used
+38	fm_fancblk	number of blocks in file
	type 0x19	FM_MNTSTAT - mount status
	type 0x1a	FM_TASKAUD - high priority task audit
	type 0x1b	FM_NMCODE - get segment name
		not currently available
	type 0x1c	FM_ACCESS - check access permissions
+28	fm_acoff	filename offset
+2c	fm_acmode	access request
+30	fm_naccess[ ]	filename
	type 0x1d	FM_USTAT - pack label
		request:
+28	fm_mvdev	mdct and partition
		reply:
+28	fm_ustat	ustat structure
+28	f_tfree	total free
+2c	f_tinode	total inodes free
+30	f_fname[ ]	file system name
+36	f_fpack[ ]	file system pack name
	type 0x1e	FM_SEGCODE - return segment name
		request:
+28	fm_segcode	code byte
+29	fm_segflag	flag byte
		reply:
+28	fm_segname	unique name for segment
	type 0x1f	FM_TEMP - no disk writes on file
		see type 0x05 request

	type 0x20	FM_BACKOUT - restore (core copy of) file
		see type 0x05 request
	type 0x21	FM_PERM - untemp file (write to disk)
		see type 0x05 request
	type 0x22	FM_MV - windowless move
+28	fm_off	offset to ‘‘from’’ filename
+2c	fm_off1	offset to ‘‘to’’ filename
+30	fm_nmv[ ]	‘‘from’’ and ‘‘to’’ filenames
	type 0x23	FM_BUFRD - buffered read
		request:
+28	fm_bfsg1	1st segment id
+2c	fm_bfsg2	2nd segment id
+30	fm_bfoff	offset into 1st segment
+34	fm_bfcnt	number of bytes for input/output (I/O) (max 128k)
		reply:
+28	ret0	not used
+2c	ret1	not used
+30	fm_aiocnt	# bytes transferred
+34	fm_iortry	memory manager used field
+38	fm_err0	no used
+3c	fm_err1	error return from driver
	type 0x24	FM_BUFWRT - buffered write
		see type 0x23 request and reply
	type 0x25	FM_LSEEK - lseek
		request:
+28	fm_lsoff	file offset
+2c	fm_lscmd	lseek command type
		reply:
+28	fm_lsaoff	resultant file offset
	type 0x26	FM_PIPE - open pipe
		see type 0x03 request and reply
	type 0x27	FM_ATOMSW - atomic switch
		see type 0x22
	type 0x28	FM_PERF - performance reporting
		request:
+28	fm_prtyp	performance request type
		0x00000000 - FMP_REQ - report request frequency
		0x00000001 - FMP_ERR - report error frequency
		0x00000002 - FMP_MISC - report misc. info
		misc. info reply:
+28	fm_fault	fault count
+2c	fm_xfault	external faults
+30	fm_pfault	phase-1 faults
+34	fm_taskfault	task faults
+38	fm_init	initialization events
+3c	fm_util	utility events
+40	fm_retry	driver retry errors
+44	fm_unknown	unknown acknowledgements
+48	fm_bfhit	buffer hits

+4c	fm_bfmiss	buffer misses	
+50	fm_bfgbusy	times buffer was busy	
+54	fm_bfgempty		
+58	fm_bfgdelw	delayed writes	
+5c	fm_nam	namei calls	
+60	fm_restraint	restrained requests	
+64	fm_teardown	task torn down	
+68	fm_mntdown	mount table entries restored	
+6c	fm_bufdown	buffers freed by teardown	
+70	fm_inodown	inodes freed by teardown	
+74	fm_fildown	file entries freed by teardown	
+78	fm_sbidown	unlocks of SUPERB ilock	
+7c	fm_sbfdown	unlocks of SUPERB flock	
+80	fm_inocnt	active inode slots	
+84	fm_inomax	high water active inode slots	
+88	fm_filecnt	active file table slots	
+8c	fm_filemax	highwater active file slots	
+90	fm_extra		
	type 0x2f	FM_FSLAUD - file system link audit	
	type 0x30	FM_FSBAUD - file system block audit	
	type 0x31	FM_AUD - audit request from sim	
	type 0x33	FM_FUAUD -msg from field update audit	
	type 0x34	FM_DIRSW - atomic directory switch see type 0x22	<Release 6.8 and Later>

Structure: delayed_q - length: 0x10

Source: head/fmgr/fmgr.h

Location: One externally declared array for each restraint queue, currently ‘‘open_q’’ and ‘‘mount_q’’.

Use: Certain file manager requests are throttled. This structure is used to chain requests which must wait for completion of earlier requests.

+0	q_count	# of active requests

+4	q_max	max # of active requests allowed
		for ‘‘open_q’’: 4 (NSOPEN)
		for ‘‘mount_q’’: 1 (NSMOUNT)

+8	*q_firstp	pointer to 1st message on queue
+c	*q_lastp	pointer to last message on queue

Structure: file - length: 0x20

Source: head/fmgr/file.h

Location: An externally declared array called ‘‘file’’.

Use: Contains file specific information with one entry for each original open (subsequent opens are chained).

+0	f_flags	file flags 0x00000001 - FLOCK - file table entry locked 0x00000002 - FWANT - another task wants (is asleep waiting)
+4	*f_iptr	pointer to incore inode
+8	*f_cptr	pointer to capability (chain)
+c	*f_fptr	pointer to next file entry on chain (there is one file table entry for each open, all point to the same inode)
+10	f_taskid	taskid information (valid only if FLOCK set)
+14	f_count	use count, > 1 indicates fork has occurred and is the number of chained cap_tbl entries
+18	f_ocount	previous (to this task) count (used for task teardown)
+1c	f_offset	read/write character pointer

Structure: filsys - length: 0x1dc

Source: head/sys/filsys.h

Location: The superblock of a file system. Mounted file systems will have the superblock in a buffer pointed to by a buf structure which is in turn pointed to by an entry in the mount table.

Use: Controls the access to the file system.

+0	s_isize	size in blocks of I list
+2	sent1	
+4	s_fsize	size in blocks of entire volume
+8	s_nfree	number of incore free blocks
+a	sent2	
+c	s_free[ ]	incore free blocks
	s_free[0] s_free[1] s_free[2]-[49]	ptr to blk containing 2nd free block array ptr to blk containing 1st free block array contains fsysdiag structure
+d4	s_ninode	number of incore free inodes (index +1 into s_inode for next free inode)
+d6	s_inode[100]	incore free inodes
+19e	s_flock	lock during free list manipulation
+19f	s_ilock	lock during I list manipulation
+1a0	s_fmod:1 s_ronly:1 s_ifull:1 s_bfull:1	super block modified flag mounted read-only flag ilist full flag blocks full flag
+1a4	s_time	current date of last update
+1a8	s_dinfo[4]	device information
+1b0	s_tfree	total free, for subsystem examination
+1b4	s_tinode	free inodes, for subsystem examination
+1b6	s_fname[6]	file system name
+1bc	s_fpack[6]	file system pack name
+1c4	s_ftaski	(struct) taskid information (flock)
+1c8	s_itaski	(struct) taskid information (ilock)
+1cc	s_cfree	free blocks on chain
+1d0	s_nxtblk	next free block not on chain (first zero in file system bit map) *
+1d4	s_nxtcon	next available contiguous area
+1d8	s_label	file system label (0xdead3bcc)

Structure: inode - length: 0x60

Source: head/sys/inode.h

Location: The incore inode structure maintained in the file manager's inode table in a segment by itself under the external name ''inode'' (currently segment address 0x2c0000).

Use: One for each active file, each current directory, each mounted-on file, and root.

+0	i_hchain	inode hash chain pointer
+4	i_flag	inode flags x'0001 - ILOCK - locked x'0002 - IUPD - has been modified x'0004 - IACC - update access time x'0008 - IMOUNT - mounted-on x'0010 - IWANT - process is waiting on lock x'0020 - ITRUNC - to be truncated x'0040 - ICRT - has been created x'0080 - IXSYNC - has been temped (do not write inode to disk) x'0100 - ITRUNC2 - has been truncated without freeing blocks at least once. x'0200 - IFSAUD - being audited (fsaud) x'0400 - ITRSHD - fsaud says is trashed.
+8	i_count	total reference count (incremented each task use - inode slot not reused unless count goes to zero)
+9	i_wcount	reference count (writes)
+a	i_invoc	invocation count for inode on disk
+b	i_use	usage count for incore inode (incremented only when inode slot is assigned - put in capabilities)
+c	i_number	i number, 1-to-1 with device address
+e	i_mindx	mount table index at mount point
+f	i_unused	
+10	i_dev	mount table pointer of the file system containing this inode.
+14	i_taskid	(struct) taskid information (valid only if ILOCK set)

+18	i_tstamp	time stamp of last disk update
+1c	*i_fptr	pointer to file entry chain
+20	i_mode	mode of inode
		bit flags:
		x’0040 - IEXEC - execute permission
		x’0080 - IWRITE - write permission
		x’0100 - IREAD - read permission
		x’0400 - ISGID - set group id on execution
		x’0800 - ISUID - set user id on execution
		inode types:
		x’1000 - IFIFO - FIFO special
		x’2000 - IFCHR - character special
		x’3000 - IFMPC - multiplexed character
		x’4000 - IFDIR - directory
		x’5000 - IPIPE - *UNIX* system pipe
		x’6000 - IFBLK - block special
		x’7000 - IFMBP - multiplexed block
		x’8000 - IFREG - regular
		x’a000 - IFEXT - contiguous extents
		x’b000 - IF1EXT - one contiguous extent
		x’c000 - IFIOP - IOP special
		x’e000 - IFREC - record
		x’f000 - IFMT - inode file type mask
+22	i_nlink	directory entries (# of links)
+24	i_uid	owner
+26	i_gid	group of owner
+28	i_size	size of file - end of actual data
+2c	i_addr[ ]	disk addresses
+2c	i_addr[0]	direct block address (normal file)
		mdct-rid (special device file)
+30	i_addr[1]	direct block address (normal file)
		dcn (special device file)
+34	i_addr[2]	direct block address (normal file)
		partition (special device file)
		name (FIFO or pipe special device file)
+38	i_addr[3]->[9]	direct block addresses (normal file)
+54	i_addr[10]	single indirect block address (normal file)
+58	i_addr[11]	double indirect block address (normal file)
+5c	i_addr[12]	triple indirect block address (normal file)

Structure: measf - length: 0x1e4

Source: os/fmgr/head/meas.h

Location: An externally declared structure named ‘‘mf’’.

Use: A collection of counters which characterize the activity of the file manager since the last boot.

+0	m_fill1	x’dead3b indicates start
+4	m_fhist[FM_LAST+1]	frequency histogram for requests
+d4	m_fill2	x’dead3b indicates end m_fhist
+d8	m_err[NERR]	frequency histogram for errors
+1a0	m_fill3	x’dead3b indicates end of m_err
+1a4	m_fault	# of faults
+1a8	m_xfault	# of external faults
+1ac	m_pfault	# of phase 1 faults
+1b0	m_taskfa	# of task faults
+1b4	m_retry	# of retry errors from drivers
+1b8	m_unknow	# of unknown acks received
+1bc	m_bfhit	# of buffer hits
+1c0	m_bfmiss	# of buffer misses
+1c4	m_bfgbus	# of times buffer was busy
+1c8	m_bfgemp	unknown
+1cc	m_bfgdel	# of delayed writes
+1d0	m_restra	# of restrained requests
+1d4	extra[16]	

Structure: mnttab - length: 0x58

Source: head/mnttab.h

Location: An externally declared array called ‘‘mnttab’’.

Use: One entry for each mounted file system telling the pathname and file system name.

+0	mt_dev	device filename
+20	mt_filsys	file system name
+40	mt_ro_fl	read only flag
+44	mt_time	mount time ?????
+48	mt_devid	new device id
+48	dev_mdct	
+4c	dev_part:16	
+50	mt_dcn	major device

Structure: mount - length: 0x48

Source: head/fmgr/fmgr.h

Location: An externally declared array called ‘‘mount’’.
Use: One entry is maintained for each file system mount. Primarily used for file access across file systems. Points to the inode of the root directory of the file system as well as the inode of the mount point (both of which have their use counts incremented).

+0	m_mdct	device mdct-rid
+4	m_dcn	device major number
+6	m_part	device partition
+8	*m_bptr	pointer to superblock bfr header
+c	*m_fcbptr	pointer to free chain bfr header
+10	*m_iptr	pointer to mounted inode (inode of directory mounted upon)
+14	*m_rootp	pointer to root inode of filesys
+18	m_use	use count
+19	m_audited	file system has been audited bits 0 -> 4: has been audited flag bits 5 -> 7: audit flags
+1c	m_taskid	taskid information
+20	m_prevmn	flag for previous mount (used for task teardown)
+21	m_rdonly	flag for previous read only (used for task teardown)
+22	m_syncmax	number of syncs since last SB write
+23	m_spare1	unused
+24	m_fsinfo	fsysdiag structure
+24	fs_mts	mount time stamp
+28	fs_opens	opens since mount
+2c	fs_close	closes since mount
+30	fs_reads	total direct reads
+34	fs_writes	total direct writes
+38	fs_seio	buffered block I/O errors
+3a	fs_bdeio	buffered data block I/O errors
+3c	fs_deio	not-buffered data block I/O errors
+40	fs_aflag	file system audit flags
+42	fs_audcnt	audits since mount
+44	fs_blker	block audit error count
+46	fs_lnker	link audit error count

Structure: msgbufe - length: 0x10

Source: head/msgbufe.h

Location: In the kernel’s address space at 0x360000 (may vary per head/va.h). The index into the segment for each element is the same as the index into the message segment for its corresponding message buffer.

Use: A kernel private segment used for queuing and auditing the message buffers.

+0	*me_link	message queue link field
+4	me_owner	current owner of the associated message buffer
+8	me_tstamp	the time this message buffer last changed status (allocated buffers only)
+c	me_blks	total blocks allocated
+d	me_flags	message extender flags x’08’ - allocated x’10’ - old message (audit flag) x’20’ - old queued message x’40’ - currently on a message queue x’80’ - has been dequeued at least once
+e	me_type	message type of the associated message buffer
+f	me_unused	unused at this time

Structure: tasktab - length: 0x800

Source: os/fmgr/task.h

Location: One per task segment as declared in task0.c through task15.c (currently segment addresses 0x40000 through 0x220000). The addresses of each of the tasks are located in an externally declared array called ‘‘tasktab’’. The declaration ‘‘taskptr’’ points to the currently active task.

Use: Each defines the state of its associated task.

+0	t_status	task status x’00000000’ - TAVAIL - slot available x’00000001’ - TINUSE - slot in use x’00000002’ - TREADY - ready to run x’00000004’ - TACTIVE - currently running x’00000008’ - TNOACK - don’t ack message x’00000010’ - TFUNC - function completed x’00000020’ - TDECQU - restraint queue decremented x’00000040’ - TDOWN - tear task down when audit runs
+4	t_taskid	task identification
+8	*t_slpaddr	task sleep address
+c	*t_stack	pointer to task stack
+10	*t_msg	pointer to recvd message
+14	t_err	task error status x’00000001 - EPERM - Not super-user x’00000002 - ENOENT - No such file or directory x’00000003 - ESRCH - No such process x’00000004 - EINTR - Interrupted system call x’00000005 - EIO - I/O error x’00000006 - ENXIO - No such device or address x’00000007 - E2BIG - Arg list too long x’00000008 - ENOEXEC - Exec format error x’00000009 - EBADF - Bad file number x’0000000a - ECHILD - No children x’0000000b - EAGAIN - No more processes x’0000000c - ENOMEM - Not enough core x’0000000d - EACCES - Permission denied x’0000000e - EFAULT - Bad address x’0000000f - ENOTBLK - Block device required x’00000010 - EBUSY - Mount device busy x’00000011 - EEXIST - File exists x’00000012 - EXDEV - Cross-device link x’00000013 - ENODEV - No such device x’00000014 - ENOTDIR - Not a directory x’00000015 - EISDIR - Is a directory x’00000016 - EINVAL - Invalid argument x’00000017 - ENFILE - File table overflow x’00000018 - EMFILE - Too many open files x’00000019 - ENOTTY - Not a typewriter

		x’0000001a - ETXTBSY - Text file busy x’0000001b - EFBIG - File too large x’0000001c - ENOSPC - No space left on device x’0000001d - ESPIPE - Illegal seek x’0000001e - EROFS - Read only file system x’0000001f - EMLINK - Too many links x’00000020 - EPIPE - Broken pipe x’00000021 - ETEMP - Temped file x’00000022 - ENOTRAP x’00000023 - ENOMSG x’00000024 - ENOALOC x’00000026 - EFSAUD x’00000027 - EFIRST x’00000028 - ENOMOVE x’00000029 - ENOEXT x’0000002a - EPATH x’0000002b - ETABLE x’0000002c - EFUNC x’0000002d - EFMAUD
+18	t_tstamp	task start time stamp
+1c	t_patchcnt	system patch cnt at task start
+20	t_proc	temp storage of proc number when device driver created or opened
+24	t_ncblks	temp storage of #/contig blks
+28	t_offset	temp storage of directory offset
+2c	t_dent	temp storage of directory entry
+2c	d_ino	inode number
+2e	d_name[ ]	last component of pathname
+3c	*t_pdir	temp storage of ptr to incore directory inode
+40	t_start	used for performance task timing
+44	t_dbuf[DIRSIZ]	temp storage of pathname component
+52	t_sdfpath	special device file pathname for ECD access *
+92	t_bufreq	buffered io flag x’00’ - no x’01’ - yes
+93	t_extra[16]	structure padding
+a4	t_stackarea	task private stack (remainder of page)

Release 21 Hexadecimal Offset Charts

9

Contents

Release 21 Hexadecimal Offset Charts

9

Kernel Address Space

This section lists the control block structures in the kernel address space. The lists name each field and give the hexadecimal offset to the field from the beginning of the structure.

Structure: ative - length: 0x1c

Source: os/kern/ative.h

Location: Found in the kernel’s data segment in an array called ‘‘atchtbl[ ]’’. Address is variable from load to load.

Use: Maintains the necessary information for the kernel to dispatch processes attached to interrupts.

+0	at_prc	attached process number
+4	*at_ent()	attached process entry point
+8	at_psw	psw value for interrupt
+c	at_ident	interrupt id
+10	at_sbr	segment base register value
+14	at_is	interrupt source
+15	at_ch	I/O channel
+16	at_dv	I/O device
+17	at_dummy	unused at this time
+18	at_status	status of the attach point x’00000001’ - AT_FREE x’00000010’ - AT_BUSY

Structure: dcte - length: 0x80
Source: head/dcte.h
Location: In the kernel’s address space at 0x140000 (may vary per head/va.h).
Use: Is the central source of process related information in the kernel and special processes.

+0	d_flag	flag word 0x00000001 - DF_LFAIL - memory manager failed to load 0x00000002 - DF_LBOLT - on lighting bolt list 0x00000004 - DF_PROFIL - profiler is invoked 0x00000008 - DF_RB - process is roadblocked 0x00000010 - DF_JC - process giving up processor 0x00000020 - DF_TOUT - process has exhausted time slice 0x00000040 - DF_LOAD - process being loaded 0x00000080 - DF_READY - ready to run (not roadblocked) 0x00000100 - DF_RLIST - on ready list 0x00000200 - DF_SLEEP - asleep on a bit pattern 0x00000400 - DF_REMOV - this process being terminated 0x00000800 - DF_SWAP/DF_SUP - supervisor process 0x00001000 - DF_NOSWAP - non-swappable supervisor process 0x00002000 - DF_KPRC - kernel process 0x00004000 - DF_SYS - special process 0x00008000 - DF_RTOR - on repetative timeout queue 0x00010000 - DF_STOR - on single timeout queue 0x00020000 - DF_STATIC - static priority 0x00040000 - DF_TMPNR - temporarily non-resident 0x00080000 - DF_NOTERM - a non killable process 0x00100000 - DF_RUN - this process has run 0x00200000 - DF_NOFIT - not enough memory to load last try 0x00400000 - DF_MSGHOG - supervisor message hog 0x00800000 - DF_DLMESSNTL - disk limp mode essential 0x01000000 - DF_DLMNONSWP - made nonswap because of DLM 0x02000000 - DF_RRTOUT - process has exhausted Round Robin time 0x04000000 - DF_FLTMSG - owns an audit fault message 0x08000000 - DF_MAXINTVL - process defined maximum time-out interval 0x10000000 - DF_DIOCHG - informed of DIO state changes 0x20000000 - DF_SUSPEND - process suspended 0x40000000 - DF_DISP - process dispatched since last sched tick 0x80000000 - DF_DEAGED - process has de-aged
+4	*d_link	link to the next dcte on the dispatcher chain of the same execution level

+8	*d_lblk	link for l_bolt chain
+c	*d_stmlk	link for single time-out chain
+10	*d_rtmlk	link for repetitive time-out chain
+14	d_rtime	time interval for repetitive time-out request
+18	*d_msg	pointer to the first message to this process
+1c	*d_msgend	pointer to last message
+20	d_stout	real-time value (msec) for single time-out request
+24	d_rtout	real-time value (msec) for repetitive time-out request
+28	d_evflag	event flags 0x00000001 - E_USR16 - user event 0x00000002 - E_USR15 - user event 0x00000004 - E_USR14 - user event 0x00000008 - E_USR13 - user event 0x00000010 - E_USR12 - user event 0x00000020 - E_USR11 - user event 0x00000040 - E_USR10 - user event 0x00000080 - E_USR9 - user event 0x00000100 - E_USR8 - user event 0x00000200 - E_USR7 - user event 0x00000400 - E_USR6 - user event 0x00000800 - E_USR5 - user event 0x00001000 - E_USR4 - user event 0x00002000 - E_USR3 - user event 0x00004000 - E_USR2 - user event 0x00008000 - E_USR1 - user event 0x00010000 - E_SYS16 - reserved for system use 0x00020000 - E_SIGCHAR - inter-character timeout 0x00040000 - E_SIGACK - acknowledge timeout 0x00080000 - E_CHILD - reserved for system use 0x00100000 - E_SYS12 - reserved for system use 0x00200000 - E_AUD - initializing audits 0x00400000 - E_UTIL - utility event for planting break points 0x00800000 - E_RTIMEOUT - repetitive timeout 0x01000000 - E_INIT - initialize 0x02000000 - E_ABORT - abort 0x04000000 - E_QIT - quit 0x08000000 - E_INT - interrupt occurred 0x10000000 - E_HUP - hung up 0x20000000 - E_MSG - message received 0x40000000 - E_TIMEOUT - timeout 0x80000000 - E_WAKEUP - wakeup

+2c		d_pn	process number
+30		d_pcbent	
	+30	*d_pcbid	pcb segment id (kern or sup processes)
	+30	(*d_sproc)()	entry point to special process
+34		d_sleep	sleep bit pattern
+38		d_ucnt	user count
+3a		d_fcode	fault code 0x00 - NOFLT - no fault (normal) 0x10 - x80 - unused by RTR, reserved for applications 0x81 - OV_SOPOK - Spooler output process overload cleared 0x82 - OV_SOPOVLD - Spooler output process overload 0x85 - OV_DLINCLR - Data Link input buffer overload cleared 0x86 - OV_DLINOVLD - Data Link input buffer overload 0x87 - OV_DLOPCLR - Data Link output buffer overload cleared 0x88 - OV_DLOPOVLD - Data Link output buffer overload 0x8b - OV_IOPOK - IOP overload cleared 0x8c - OV_IOPOVLD - IOP command queue overload 0x8d - OV_IOPBOOT - An IOP was bootstrapped due to overload 0x8f - OV_DFCOK - DFC overload cleared 0x90 - OV_DFCOVLD - Disk File controller overload 0x91 - OV_FMOVCLR - File Manager overload cleared 0x92 - OV_FMOVLD - File Manager overload 0x95 - OV_SUOVCLR - User/Supervisor level program lockout cleared 0x96 - OV_SUOVLD - User/Supervisor level program lockout overload 0x97 - OV_TEOVCLR - timed event overload cleared 0x98 - OV_TEOVLD - timed event overload condition 0x99 - OV_KLCLR - Kernel level lockout cleared 0x9a - OV_KLOCK - Kernel level lockout overload condition 0x9c - OV_MEM1CLR - Module 1 memory allocation reduced to 60% 0x9d - OV_MEM1LOW - Module 1 memory at 80% allocated 0x9e - OV_MEM1FULL - Module 1 insufficient memory available 0x9f - OV_DCTOK - DCT entries allocation reduced to 50% 0xa0 - OV_DCTOVLD - DCT entries at 70% allocation 0xa1 - OV_DCTCRIT - DCT entries at 100% allocation 0xa2 - OV_SDECLR - SDE table overload recovered - 100 entries left 0xa3 - OV_SDELOW - SDE table overload - only 50 entries left 0xa4 - OV_SDEPRFL - Insufficient Segment Descriptor Entries 0xa5 - OV_SWAPCLR - Disk Swap Space reduced to 60% used 0xa6 - OV_SWAPLOW - Disk Swap Space 80% used 0xa7 - OV_MEMCLR - Non_swappable main memory allocation reduced to 60% 0xa8 - OV_MEMLOW - Non-swappable main memory 80% allocated 0xa9 - OV_MEMKPFL - Insufficient memory available to create a kernel process 0xac - OV_MSGOK - Message Buffers recovered to 50% allocation 0xad - OV_MSGLOW - Message Buffers at 70% allocation 0xae - OV_MSGCRIT - Message Buffers at 90% allocation 0xaf - OV_MSGOUT - Message Buffers completely full 0xb0 - FLT_FULL_DIO - DIO state is FULL DISK LIMP MODE 0xb1 - FLT_DREP - device reported error for programmed IO 0xb2 - FLT_PICP - PIC fault for programmed IO

		0xb3 - FLT_CCP - processor fault for programmed IO 0xb4 - FLT_DRED - device reported error for DMA IO 0xb5 - FLT_ADRD - DMA addressing fault for DMA IO 0xb6 - FLT_PICD - PIC fault for DMA IO 0xb7 - FLT_CCD - processor fault for DMA IO 0xb8 - FLT_NDRE - Device Reported error - PIO. No message 0xb9 - FLT_NPIC - PIC implicated - PIO in progress. No message 0xba - FLT_NADR - Addressing error - PIO in progress. No message 0xbb - FLT_MYC - process data has noncorrectible parity error 0xbc - FLT_QMSAUD - Message Buffer Audit fault (queued messages) 0xbd - FLT_NONQMSAUD - Message Buffer Audit fault (nonqueued messages) 0xbe - FLT_DUPLEX - DIO state is DUPLEX 0xbf - FLT_SIMPLEX - DIO state is SIMPLEX 0xc0 - FLT_CFT - Craft initialization fault 0xc1 - FLT_RSCOMP - recovery switch of processors is complete 0xc2 - FLT_SSCOMP - soft switch is complete 0xc3 - FLT_CMI - config. manager initialize request 0xc4 - FLT_CMA - config. manager remove request 0xc5 - FLT_CMB - config. manager remove request 0xc6 - FLT_CMC - config. manager limp mode 0xc7 - FLT_CMD - config. manager limp mode 0xc8 - FLT_CMAN - config. manager manual/ADP or routine remove 0xc9 - FLT_UCLRMV - unconditional remove 0xca - FLT_SSREQ - Request for a soft switch (from pcpmd to sim) 0xcb - FLT_SOFTSW - Routine soft switch - pcpeih only 0xcc - FLT_CRMV - config. manager remove under fault conditions 0xcd - FLT_TMOUT - non-segmented kernel level audit timed out 0xd1 - FLT_PINV - memory management page invalid or not in memory 0xd2 - FLT_PIND - memory management - page index too large 0xd3 - FLT_SINV - memory management - segment invalid 0xd4 - FLT_SIND - memory management - segment index too large 0xd5 - FLT_BADOST - illegal ost 0xd6 - FLT_PROT - protection violation 0xd7 - FLT_ADDR - byte or halfword addressing violation 0xd8 - FLT_PRIV - instruction privilege violation 0xd9 - FLT_OPCD - illegal op code 0xda - FLT_STACK - illegal switch between kernel and private stack 0xe0 - FLT_PHASE0 - phase level 0 initialization 0xe1 - FLT_SINIT - system initialization 0xe2 - FLT_SCRIT - system initialization - critical 0xf0 - xff - reserved for *UNIX*® RTR operating system applications
+3b	d_chan	control channel
+3c	d_cprior	current priority for supervisor process process id for kernel process
+3d	d_iprior	initial priority for supervisor process execution level for kernel process
+3e	d_age	time in 1/2 sec units that the process is waiting to be scheduled (type: char)
+3f	d_unused	

+40	d_pcode:11 d_ucode:11 d_spare:10	pcode portion of the sup/kp utilid pcode portion of the user utilid sure kill flags 0x000 - DS_TRMPID - norm. term by PID 0x001 - DS_UTRMPID - uncond. term by PID 0x002 - DS_TRMUID - norm. term by UID 0x004 - DS_UTRMUID - uncond. term by UID 0x008 - DS_TRMCLASS - norm. term by class 0x010 - DS_UTRMCLASS - uncond. term by class 0x020 - DS_UNIXTERM - uncond. *UNIX* system term
+44	d_class	process class flag 0x00000001 - DC_DLG - dialogue processes 0x00000002 - DC_ESSEN - essential process 0x00000004 - DC_ULARP - process monitored by ULARP user monitor 0x00000008 - DC_CFT_T - craft processes associated with a tty 0x00000010 - DC_CFT_D - craft daemon processes 0x00000020 - DC_CFT_S - processes spawned by spooler, (sops) 0x00000040 - DC_CINIT - poker processes 0x00000080 - DC_CFT_P - all craft processes 0x000000b8 - DC_CFT - processes notified of craft init

* UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

+44		d_class	process class flag 0x00000100 - DC_ODIN - processes that use odin 0x00000200 - DC_HPRI - high priority terminal processes 0x00000400 - DC_DAP - processes notified of a DAP restart 0x00000800 - DC_SPARE - spare class bits 12-31 - reserved for applications
+48		d_eent	event entry (psw and function)
	+48	pe_psw	processor status word
	+4c	pe_pa	entry point for program address
+50		d_oent	ost entry vector (psw and function)
	+50	pe_psw	processor status word
	+54	pe_pa	entry point for program address
+58		d_fent	fault entry vector (psw and function)
	+58	pe_psw	processor status word
	+5c	pe_pa	entry point for program address
+60		d_sbr	segment base value
+64		d_enable	enable flag for event entry
+68		d_slptr - pointer to slist entry for supervisor	process, used by fltint OST for kernel process: bits 0-25 reserved for PA 0x04000000 - DELFLTSI - deliver FLT_SINIT bit 0x08000000 - SEGMOD - temporarily invalidated user segment 0x10000000 - SYS_PHASE - FLT_SINIT set due to system phase
+6c		d_psize	current size of this process
+70		d_rdblk	# s_ticks process remains RB’d
+71		d_inmem	# s_ticks process remains in memory
+72		d_msgcnt	message buffer usage count
+74		d_start	process start time
+78		d_otime	process time in ost code
+7c		d_ptime	process time in process code

Structure: dctext - length: 0x20

Source: head/dcte.h

Location: In the kernel’s address space at 0x340000 (may vary per head/va.h).

Use: An extension of the dct entry which, like the dcte, contains process related information.
Address conversion equation:
*dctext= (0x340000 + (*dcte & 0xffff) >>2)

+0	de_dctndx	index of the associated dcte
+2	de_state	creation/termination/suspend states 0x0000 - No creation/termination/suspend in progress Termination: 0x0064 - use count decremented 0x006e - term_dct() called 0x0078 - GRASP informed 0x0082 - unlinked from dispatch 0x008c - atb flushed 0x0096 - ack msg created 0x00a0 - unlinked from slist 0x00aa - incarnation cnt bumped 0x00b4 - forwarded to CMGR 0x00be - forwarded to PMGR 0x00c8 - caps removed 0x00d2 - sup segs removed 0x00dc - kp segs removed 0x012c - core dump started 0x0136 - forwarded from PMGR to CMGR Creation: 0x01f4 - kp pcreat started 0x01fe - kp dcte linked 0x0208 - E_INIT sent 0x0258 - sp pcreat started 0x0262 - sp dcte linked 0x026c - sp pstart completed 0x0276 - sp execute started 0x02bc - fork started 0x02c6 - dupcaps sent to FMGR 0x02d0 - pfork2 started 0x02da - wakeup to child Suspend: 0x0320 - process suspended from execution
+4	de_tstamp	last major change in creation/termination also used for (s) maxintvl OST
+8	de_pmap1	public library bit map 1
+c	de_pmap2	public library bit map 2 0x00000001 - ECD 0x00000002 - PLM 0x00000004 - KCONFIG

		0x00000008 - *UNIX* system 0x00000010 - CRAFT 0x00000020 - LLA incore 0x00000040 - LLA general
+10	*de_s_fp	scheduler list forward pointer for supervisor process
+14	*de_s_bw	scheduler list backward pointer for supervisor process
+18	de_xtra1	spare field
+1c	de_xtra2	spare field

Structure: instat - length: 0x50

Source: head/instat.h

Location: Found on the interrupt stack for preempted processes and occasionally on a process's stack depending upon the activity of the process.

Use: Contains all system register values required to resume execution of a preempted process.

+0	i_pa	program address at interrupt
+4	i_psw	psw
+8	i_psbr	primary segment base register
+c	i_ssbr	secondary segment base register
+10	i_reg[0]	general purpose reg 0
+14	i_reg[1]	general purpose reg 1
+18	i_reg[2]	general purpose reg 2
+1c	i_reg[3]	general purpose reg 3
+20	i_reg[4]	general purpose reg 4
+24	i_reg[5]	general purpose reg 5
+28	i_reg[6]	general purpose reg 6
+2c	i_reg[7]	general purpose reg 7
+30	i_reg[8]	general purpose reg 8
+34	i_reg[9]	general purpose reg 9 (argument pointer)
+38	i_reg[10]	general purpose reg 10 (frame pointer)
+3c	i_reg[11]	general purpose reg 11 (stack pointer)
+40	i_reg[12]	general purpose reg 12
+44	i_reg[13]	general purpose reg 13
+48	i_reg[14]	general purpose reg 14
+4c	i_reg[15]	general purpose reg 15

Structure: iparm - length: 0x400

Source: head/fltrcv/phymem.h

Location: Found at physical address 0.

Use: This structure of data contains critical parameters that are used during a system initialization such as a bootstrap. This structure is initially set up by an early step in the bootstrap process (Little Boot).

+0	i_halt		halt ins. - protect against wild transfer
+4	i_initadr		INIT address for no-bootstrap init.
+8	i_bootadr		PINIT address for bootstrap init.
+c	i_pclr		power up flag word
+10	i_usav	struct-usav	set up by microcode on a mrf
	+10	us_pa	program address
	+14	us_sar	store address register
	+18	us_ssr	system status register
	+1c	us_psw	program status word
	+20	us_scr	store control register
	+24	us_err	3B error register
	+28	us_ib	instruction buffer
	+2c	us_sir	store instruction register
	+30	us_ppr	pulse point register
	+34	us_hsr	hardware status register
	+38	us_sdr	store data register
	+3c	us_im	interrupt mask
	+40	us_is	interrupt set register
			3B20D computer MAINSTORE
	+44	us_ser00	tore error register 0 - cont. 0
	+48	us_ser01	store error register 1 - cont. 0
	+4c	us_star0	tore trap address error reg - cont 0
	+50	us_ser10	store error register 0 - cont. 1
	+54	us_ser11	store error register 1 - cont. 1
	+58	us_star1	store trap address error reg - cont 1
			3B21D MAINSTORE
	+44	mc_cmd	Command Register
	+48	mc_stat	Status Register
	+4c	mc_erradd	Error Address Register
	+50	dp_cmd	DP Command Register
	+54	dp_errdata	DP Error Data Register
	+58	dp_stat	DP Status Register
	+5c	us_rtc	realtime clock
	+60	us_timers	timers
+64	i_minit		microcode on a manual initialization
+68	i_maothcu		maintenance activity in other CU (flag for PINIT)

+6c	i_dsp2			spare
+70	i_dsp3			spare
+74	i_dsp4			spare
+78	i_dsp5			spare
+7c	i_dsp6			spare
+80	i_vtoc	union-vtocent[10]		set up by little boot
	+80	lboot	struct	
		+80	v_flags	partition flags
		+81	v_lsize	last 4 bits - size of lboot
		+82	v_parno	partition number
		+84	v_startblk	start of partition
		+88	v_nblks	length of partition: includes writable CU ucode
		+8c	v_load	little-boot load address
	+80	vtoc	struct	
		+80	v_flags	partition flags
		+81	v_addlflgs	additional flags
		+82	v_parno	partition number
		+84	v_startblk	start of partition
		+88	v_nblks	length of partition
		+8c	v_packid	struct
		+8c	v_packname[2]	disk pack name
		+8e	v_packno	disk pack number
	+80	ventry	struct	
		+80	v_flags	partition flags
		+81	v_addlflgs	additional flags
		+82	v_parno	partition number
		+84	v_startblk	start of partition
		+88	v_nblks	length of partition
		+8c	v_bkparno	partition number of mate, if partition is one of a pair
+120	i_seqnum			used to identify PM or error slots
+124	i_cpspr1		spare	
+128	i_cpspr2		spare	
+12c	i_cpspr3		spare	
+130	i_cpspr4		spare	
+134	i_cpspr5		spare	
+138	i_cpspr6		spare	
+13c	i_btsftclk			sftclk value of the last bootstrap
+140	i_eaibuf	struct-eaipar		EAI interface buffer
	+140	ei_uc		dedicated to microcode
	+141	ei_ilvl		DMERT initialization level
	+142	ei_inh		inhibit options
	+143	ei_conf		configuration options
	+144	ei_appl		1st character of application parameter
	+145	ei_per		periodically scanned options
	+146	ei_appl2		2nd character of application parameter

	+147	ei_uc1	Dedicated for the micro-code to keep track of the boot device ID. Left nibble - DFC 0 Bus Device ID Right nibble - DFC 1 Bus Device ID
+148	i_pmort	*struct-pmort	pointer at first slot in postmort list
+14c	i_pmortl	*struct-pmort	pointer at last PM used
+150	i_erslot	*struct-erslot	pointer at first error slot
+154	i_erlast	*struct-erslot	pointer at last er slot used
+158	i_rparm	struct-rparm	process request parameters
	+158	r_source	source of request
	+159	r_code	panic code number
	+15a	r_dilvl	requested value of DMERT initlvl
	+15b	r_ailvl	requested valve of application initlvl
	+15c	r_uid	utility id of process requesting init
	+160	r_splong	spare
	+164	r_spc0	spare
	+165	r_spc1	spare
	+166	r_phopt	phase options
+168	i_cparm1		spare
+16c	i_cparm2		spare
+170	i_cparm3		spare
+174	i_cparm4		spare
+178	i_fltcod		fault code to use on this init.
+17c	i_dilvl		DMERT initialization level counter
+17d	i_ailvl		application initialization level counter
+17e	i_fill		fill so i_maxailvl is on word boundary
+17f	i_codecont		code flow control (see bit layout below)
+180	i_maxailvl	char[4]	maximum appl init levels for each DMERT init level
+184	i_dart		dart communication - must begin on fullword boundary
+185	i_usp2		spare
+186	i_usp3		spare
+187	i_usp4		spare
+188	i_sim	struct	CFT initialization level record
	+188	simident	SIM identifier
	+18c	cinit1	# of level 1 cinits
	+18e	cinit2	# of level 2 cinits
	+190	cinit3	# of level 3 cinits
	+192	cinitboot	# of boots soon after cinit
	+194	lastcinit	time of last cinit
	+198	time0	time structure was initialized
	+19c	cinitlvl	current cinit level
	+19e	cinitphase	current cinit phase
+1a0	i_prmdlvl		DMERT level output in PRMs
+1a1	i_prmalvl		application level output in PRMs
+1a2	i_usp5		spare
+1a3	i_usp6		spare
+1a4	i_uspc		spare
+1a8	i_padstat	char[8]	disk active status
+1b0	i_rpdd	char[8]	disk write status

+1b8	i_interval			length of initialization interval in sec.
+1bc	i_masindat			indicates which main store is up to date
+1c0	i_strlim			last addressable word of physical mas
+1c4	i_cs	struct		change state spec for passing control to kernel
	+1c4	cs_psw		kernel psw
	+1c8	cs_pa		kernel pa
	+1cc	cs_sbr		kernel sbr
+1d0	i_passize			low numbered PAS size in bytes
+1d4	i_hpassize			high numbered PAS size in bytes
+1d8	i_packid	struct		
	+1d8	i_packna	char[2]	disk pack name
	+1da	i_packno		disk pack number
+1dc	i_pdmp	struct-pdmp		panic dump structure
	+1dc	pi_dsize		number of bytes dumped to disk
	+1e0	pi_rtc		realtime clock when MRF occurred
	+1e4	pi_sp	char[2]	spare
	+1e6	pi_dstat		dump status
	+1e7	pi_dcont		dump control
	+1e8	i_pdarea	struct[10]	MAS areas to panic dump
		+1e8	pi_stadd	MAS start address
		+1ec	pi_nbytes	number of bytes to dump
+238	i_ECDadd			physical start address of Mod 0 ECD data
+23c	i_ECD1add			physical start address of Mod 1 ECD data
+240	i_ECDsiz			size of Module 0 ECD in bytes
+244	i_ECD1siz			size of Module 1 ECD in bytes
+248	i_sftclk			software clock
+24c	i_rtclk			value of rtc when sftclk last set
+250	i_patch			field update patch count
+254	i_pintmem			physical address of interrupt stack
+258	i_pkernmem			physical address of kernal interrupt stack
+25c	i_rdstk			pinit flag for saved state collection
+25d	i_bdisk			boot disk indicator for EIH
+25e	i_pbdisk			Previous boot disk indicator (for EIH)
+25f	i_csp4			spare
+260	olb	struct		offline boot structure
	+260	olb_vers		RTR offline boot version
	+264	mode		
		OLBIDLE	0x55555555	if not in offline boot
		ONLINE	0x012883dd	if in offline boot and on the online side
		OFFLINE	0x5	if in offline boot and not on the online side
		INVLID	0xfffffff6	for any other value found inside mode
	+268	original		original side, either 0 or 1
	+26c	modifiedecd		modified ecd? yes/no
	+270	sftclk		first stop time stamp
	+274	options1	struct	command line options word

		+274		bit field
		retrofit	0x00000	retrofit boot: yes/no
		ucl	0x00001	unconditional boot: yes/no
		oos	0x00002	dont abort when offline units are OOS: yes/no
		monitor	0x00003	redirect offline PRMs to online ROP: yes/no
		inh_sftc	0x00004	Software Inhibit: yes/no
		inh_hdwc	0x00005	Hardware Inhibit: yes/no
		inh_erri	0x00006	Interrupt Inhibit: yes/no
		broot	0x00007	use of backup root allowed: yes/no
		minconfi	0x00008	use MINCONFIG kernel image: yes/no
		trace	0x00009	spares
		dilvl	0x0000a	RTR init level: 2-4
		manual	0x0000d	if OLBYES, manual request
		dfcpair	0x0000e	for dfc pairs
		parm	0x00010	Application parameter
		tty	0x00018	id number: 0x00-0xfe, NOTTY (0xff)
	+278	options2	struct	command line options word
		+278		bit field
		iop_mask	0x00000	bit assert if iop to be switched
		iop_subu	0x00008	bit assert if switching all IOP subunits
		iop_move	0x00010	bit assert if iop has been moved
		spares	00018	spares
	+27c	status1	struct	status word
		+27c		bitfield
		can_rcv	0x00000	RCV allowed? yes/no
		boot_status	0x00001	
			0	Not completed
			1	Completed and Failed
			2	Completed and Successful
		nomrfs	0x00003	number of boot phases
		olback	0x00007	AIM acknowledge of MSGIP
		aim_progress	0x00008	application progress mark
		msgip	0x00010	message in progress
		retro_ucl	0x00018	chg fltrcv while switching SMs
		spare2	0x00019	spare
	+280	successp	char[8]	last success PRM seen
	+288	failingp	char[8]	first failing PRM seen
	+290	aim	Uint[45]	APP buffer, used also by CNI and RETRO
	+344	spares	Uint[5]	RTR spares
+358	i_csp5			spare
+359	i_csp6			spare
+35a	i_csp7			spare
+35b	i_csp8			spare
+35c	i_csp9			spare
+35d	i_cspa			spare
+35e	i_cspb			spare
+35f	i_cspc			spare
+360	i_cspd			spare
+361	i_cspe			spare
+362	i_cspf			spare

+36c	i_csp10		spare
+370	i_csp11		spare
+374	i_csp12		spare
+378	i_csp13		spare
+37c	i_crest[14]		Rest of the spare area
+3b4	i_usav2	struct	micro-code save area for certain mcert-registers
	mach_b_dep	union	
	mem_b_reg	struct-reg20_bset	
	+3b4	pm_dsr0	double store read 0
	+3b8	pm_dsr2	double store read 2
	+3bc	pm_dsr3	double store read 3
	+3d0	pm_hg	3A emulation hg register
	+3d4	pm_a_sar	ATB scratch register
	+3d8	pm_a_sdr	ATB scratch register
	+3dc	pm_a_scr	ATB scratch register
	+3e0	pm_a_q	ATB scratch register
	+3e4	pm_a_psw	ATB scratch register
	+3e8	pm_a_bgr	ATB scratch register
	+3ec	spare[9]	Spare registers
	mcert_b_reg	struct-reg21_bset	
	+3b4	mm_reg	Main memory register
	+3b8	m_error	Main Memory Error Register
	+3bc	pm_0bnkadd	MCERT bank address reg 0
	+3c0	pm_1bnkadd	MCERT bank address reg 1
	+3c4	pm_2bnkadd	MCERT bank address reg 2
	+3c8	pm_3bnkadd	MCERT bank address reg 3
	+3cc	pm_4bnkadd	MCERT bank address reg 4
	+3d0	pm_5bnkadd	MCERT bank address reg 5
	+3d4	pm_6bnkadd	MCERT bank address reg 6
	+3d8	pm_7bnkadd	MCERT bank address reg 7
	+3dc	us_acacerr	Cache Error Register a
	+3e0	us_bcacerr	Cache Error Register b
	+3e4	us_ccacerr	Cache Error Register c
	+3e8	us_dcacerr	Cache Error Register d
	+3ec	us_ecacerr	Cache Error Register e
	+3f0	us_fcacerr	Cache Error Register f
	+3f4	us_gcacerr	Cache Error Register g
	+3f8	us_spare	Spare word
	+3fc	pm_eaierr	EAI error register

Structure: kpcb - length: 0x800

Source: head/kpcb.h

Location: One segment in each kernel process address space. All kpcb segments can be located via the kernel’s dispatcher control table (DCT) entries.

Use: Contains all process specific information not needed directly by the kernel. Of most significance is the process’s segment table which is used to define the process’s virtual address space.

+ 0		k_utilid	utility id
+ 4		k-sgt	segment table
+800		k_sbr	segment base register value
+804		k_eent	event entry vector
	+804	pe_psw	processor status word
	+808	pe_pa	entry point for program address
+80c		k_fent	ost entry vector
	+80c	pe_psw	processor status word
	+810	pe_pa	entry point for program address
+814		k_oent	ost entry vector
	+814	pe_psw	processor status word
	+818	pe_pa	entry point for program address
+81c		k_profaddr	profiling address
+820		k_pn	process number
+824		k_parpn	parent process number
+828		k_name	ASCII name of process
+838		k_tident	message ident to send on process death
+83c		k_tflag	if !=0, send k_ttype msg on process death
+83d		k_ttype	message type to send on process death
+83e		k_chan	control channel number
+83f		k_pcbpg	number of pages in this kpcb
			The following declarations indicate spare fields to allow field update of new process related information
+840		k_c0	spare characters
+848		k_s0	spare shorts
+850		k_i0	spare ints
+860		k_u0	spare unsigned for flag Kernel Process Segment List
+868		k_size	largest active segment number

```
+86a        k_sglsz           maximum segment configuration
+86c        k_seglist         segment list
    +86c      k_segflg
                 kf_segndx:9
                 kf_segflg:23   segment flag word (struct ksegf)
                     x'00000000' - KF_FREE - free segment list entry
                     x'00000001' - KF_EXEC - segment is executable
                     x'00000002' - KF_WRT - segment is writable
                     x'00000004' - KF_RD - segment is readable
                     x'00000008' - KF_STK - segment is stack segment
                     x'00000010' - KF_PWRT - process can make segment writable
                     x'00000020' - KF_SHARE - segment is sharable
                     x'00000040' - KF_IOMAP - segment used by iomap primitive
     +870   k_segid  segment id
```

Structure: kvt - length: 0x254

Source: head/kvt.h

Location: Found in the kernel's data segment in an external declaration called ''Kvt''. The exact address will vary from load to load.

Use: Contains all spy package metering data.

+0	a_dctmem	virtual address of the 1st dcte
+4	a_dctpa	physical address of the 1st dcte
+8	a_depa	physical address of the 1st dctext
+c	dctcnt	total number of dcte's
+10	a_dctfree	address of the free dct count
+14	a_usrdct	address of pointer to current runner
+18	a_nxtdct	address of pointer to next loading proc
+1c	a_dispq	address of the dispatch queues
+20	a_portmem	virtual address of the 1st port
+24	a_portpa	physical address of the 1st port
+28	portcnt	total number of ports
+2c	a_stckpa	physical address of the kernel stack
+30	stacksize	size of the kernel stack in bytes
+34	a_msgpa	physical address of the 1st message
+38	a_mepa	physical address of 1st msg extender
+3c	msgcnt	total number of message buffers
+40	a_msgfree	address of free message block count
+44	a_Ovldfg	address of the message buffer overload flag
+48	pdecnt	total number of physical pages
+4c	pde1cnt	total number of physical pages module 1
+50	a_pdefree	address of number of free pages
+54	a_pde1free	address of number of free pages module 1
+58	pgecnt	total number of page tables
+5c	a_pgefree	address of number of free page tables
+60	a_sdemem	virtual address of the 1st sde
+64	a_sdepa	physical address of the 1st sde
+68	sdecnt	total number of sde's
+6c	a_sdefree	address of free sde count
+70	a_disksize	address of the disk swap size
+74	a_diskfree	address of the free disk swap blocks

+78	a_swapsize	address of the swap size
+7c	a_swapmin	address of the swap size minimum
+80	a_Swapis	address of the segments swapped in count
+84	a_Swapos	address of the segments swapped out count
+88	a_Swapib	address of the bytes swapped in count
+8c	a_Swapob	address of the bytes swapped out count
+90	Sktime	time in kernel
+d0	Skptime	time in kernel processes
+110	Sstime	time in supervisor processes
+118	Sutime	time in user processes
+11c	a_Tidle	address of time in idle loop
+120	a_prevtod	address of last tod clock tick
+124	sdis_lev	supervisor and user processes
+164	sdis_dif	dispatching statistics
+1a4	pcra_cnt	number of processes created
+1a8	pkil_cnt	number of processes killed
+1ac	nkost	address of kernel ost counts
+1b0	nKPost	address of kernel process activity array
+1b4	nsupost	address of supervisor ost counts
+1b8	nSUPost	address of level 2 activity indicator
+1bc	nuost	address of user ost data
+1c0	nint	address of interrupt usage data
+1c4	npir	address of pir usage data
+1c8	nkprocess	address of kernel process pid array
+1cc	a_clientdct	address of the kernel process trapping chain
+1d0	a_iptpnum	address of pnum of last kp trapped to by a sup
+1d4	ldfails	number of load fails
+1d8	ldreqs	number of load requests
+1dc	nofits	number of no fit processes
+1e0	mxfails	number of times S_MXFAIL conditions met
+1e4	rbcnt	number of roadblocked processes swapped
+1e8	actcnt	number of active processes swapped
+1ec	idle_cnt	number of idle routine entries
+1f0	rr_timer	current value of Round Robin timer

+1f4	hp_proc	number of high priority processes active and on swap dev
+1f8	lowrt_cnt	# of times realtime available is < 50ms
+1fc	u_occ	*UNIX* system occupancy
+200	u_occ_toff	*UNIX* system occupancy turn off value
+204	Tidlcnt	idle loop counter for 2STP
+208	unused[19]	reserved for field update additions

Structure: msghdr - length: 0x14

Source: head/msghdr.h

Location: Contained in each allocated message buffer in the kernel’s message segment (shared with kernel processes). This segment (KMSG) is located at 0x620000 but may vary depending upon changes to head/va.h. Supervisors use local copies of this structure.

Use: Contains message control information used by the kernel as well as the sending and receiving processes.

+0	*ms_link	ptr to next msg on input queue
+4	ms_from	sending process number
+8	ms_to	receiving process number
+c	ms_nblks	msg size in 64 byte blocks
+d	ms_flags	msg header flags x’01’ - MS_CAP - message contains capability x’02’ - MS_ULOCK - unlock segment on acknowledgement x’04’ - MS_NACK x’08’ - MS_ALOC x’20’ - MS_UTERM - msg type MSTERM - generated by a sure kill request x’40’ - MS_INCMPLT - sure kill generated over MAXUTERM MSTERM messages
+e	ms_type - message type (a partial list of values used by RTR)	x’00’ FM_BADMSG - bad message type x’00’ MSMIN x’01’ FM_READ - read from file x’02’ FM_WRITE - write to file x’02’ IOWRITE - I/O write message x’03’ FM_OPEN - open file x’03’ IOOPEN - I/O open message x’04’ FM_CLOSE - close file x’04’ P_INIT - initialize process manager x’04’ IOCLOSE - I/O close message x’05’ FM_EXEC - open file for execution x’05’ P_WAIT - returned on the death of a supervisor process x’06’ FM_FORK - increment count on open files x’06’ P_INMEM - message from field update that segments are in memory x’07’ DELCAP - delete capability x’07’ P_UNINMEM - message from field update to undo P_INMEM x’08’ FM_CREAT - create file x’08’ ADDCAP - add capability x’09’ FM_LINK - link to a file x’09’ MSTERM - terminate message x’0a’ FM_UNLINK - remove link from file x’0a’ MSGROW - grow segment message x’0a’ MSTERM2 - sure kill termination msg

x’0b’ FM_UTIME - modify date of file
x’0b’ MSLOAD - load a process
x’0c’ FM_CHDIR - change directory
x’0c’ MSPLOCK - load and process lock a segment
x’0d’ FM_INIT - initialization message
x’0d’ MSKADD - add a segment to kernel process
x’0e’ FM_MKNOD - make a node
x’0e’ MSKRMV - remove a segment from kernel process
x’0f’ FM_CHMOD - change mode of file
x’0f’ MSCMPCT - perform swap compaction
x’0f’ IOCDREAD - Control/Data buffer I/O read message
x’10’ FM_CHOWN - change owner of file
x’10’ IOCDWRITE - Control/Data buffer I/O write message
x’11’ FM_SYNC - update file systems on secondary
x’11’ IOCDCANCELT - I/O cancel message for IOCDREAD/IOCDWRITE
x’12’ FM_STAT - get status of file
x’13’ FM_SIZE - get size of file
x’13’ FM_FSIZE - FM_SIZE
x’14’ FM_FSTAT - get status of open file
x’15’ FM_MOUNT - mount file system
x’16’ FM_UMOUNT - unmount file system
x’17’ FM_MOVE - move file into contiguous area
x’18’ FM_ALLOC - allocate contiguous space for file
x’19’ FM_MNTSTAT - get a copy of the mount table mnttap
x’1a’ FM_TASKAUD - task and message queue audits
x’1b’ FM_UNFORK - remove a set of capabilities
x’1c’ FM_ACCESS - check access permissions
x’1d’ FM_USTAT - pack label
x’1e’ FM_SEGCODE - get segment name
x’1f’ FM_TEMP - temp (no disk write) a file
x’20’ FM_BACKOUT - restore (core copy of) file
x’21’ FM_PERM - untemp a file (write to disk)
x’22’ FM_MV - windowless move
x’23’ FM_BUFRD - buffered read
x’24’ FM_BUFWRT - buffered write
x’25’ FM_LSEEKl - seek
x’26’ FM_PIPE - open unnamed pipe
x’27’ FM_ATOMSW - atomic switch
x’28’ FM_PERF - file manager history
x’29’ FM_DUPCAP - duplicate (fork) capabilities
x’2a’ FM_ATOMPERM - perm 2 atomically switched files
x’2b’ FM_ATOMBACK - backout 2 atomically switched files
x’2c’ FM_DISKRM - disk removal notice from DKDRV
x’2d’ FM_WINDOW - open/close file system direct access window
x’2e’ FM_FSCAUD - file system compaction audit
x’2f’ FM_FSLAUD - file system link audit
x’30’ FM_FSBAUD - file system block audit
x’30’ IOSETPROT - set file system protection
x’31’ FM_AUD - aud msg from SIM
x’31’ IOCLRPROT - clear file system protection
x’32’ IOCANCEL - I/O cancel message (for multiple reader)

		x’33’ FM_FUAUD - msg from field update audit x’34’ FM_SHRTRD - short buffered read x’34’ IOSHRTRD - I/O short read x’35’ FM_SHRTWRT - short buffered write x’35’ IOSHRTWRT - I/O short write x’36’ FM_DIRSW - atomic directory switch x’47’ FM_LAST - including all badmsg slots in fmsegs x’61’ IOSYSDLM - I/O disk limp mode x’64’ MSFAULT - fault message x’65’ MSRCVMSG - message to drivers x’6c’ T_LIBMSG - get pids of processes using library x’7e’ ECDCHNG x’fb’ WAITMSG - child terminated x’fc’ TRCSNDp - trace message sent to child x’fd’ TRCRCV - return ptrace message sent by child x’fe’ MSSIG - signal; must be MSACK-1 x’ff’ MSACK - acknowledgement message x’ff’ MSMAX - maximum value of message types
+f	ms_stat	message status x’00’ MSNOERR - normal, non-error status x’3f’ BADTYPE x’40’ SYSERR x’e0’ MSOLD - old message returned to sender x’e1’ MBOLOAD - message buffer overload status x’ff’ MSDEAD - receiving process has died x’ff’ MSPFAIL - receiving process has died
+10	ms_size	msg size in bytes
+12	ms_otype	original type before ack
+13	ms_seqnum	message sequence number
+14	ms_ident	message id used by sender x’fffffffc’ - TRCMSG x’fffffffd’ - USRMSG x’fffffffe’ - SIGMSG x’ffffffff’ - UNXMSG

Structure: pcb - length: 0x19f4

Source: head/pcb.h

Location: One segment in each supervisor process address space. All pcb segments can be located via the kernel's dispatcher control table (DCT). The most currently running supervisor will have its pcb segment in the kernel's address space at 0x420000 (may vary per header va.h). Many supervisor's pcb start at their own virtual address of 0x600000.

Use: Contains all supervisor specific information not needed directly by the kernel.

+0	p_sutilid	supervisor utility id
+4	p_ssgt	supervisor segment table
+800	p_spsbr	supervisor process psbr
+804	p_upsbr	user process psbr
+808	p_svect	starting entry vector
+808	pe_psw	processor status word
+80c	pe_pa	entry point for program address
+810	p_evect	event entry vector
+810	pe_psw	processor status word
+814	pe_pa	entry point for program address
+818	p_fvect	fault entry vector
+818	pe_psw	processor status word
+81c	pe_pa	entry point for program address
+820	p_ovect	ost entry vector
+820	pe_psw	processor status word
+824	pe_pa	entry point for program address
+828	p_profaddr	profiling address
+82c	p_pn	process number
+830	p_parpn	parent process number
+834	p_name	ASCII name of the process
+844	p_tident	message identification to send to parent on death of process
+848	p_tflag	if !=0, send p_ttype message to parent on death of process
+849	p_ttype	message type to send on process death
+84a	p_fcode	fault code
+84b	p_static	the process has static scheduling priority
+84c	p_evopt	all or any option for event wait
+84d	p_crflag	time out occurred while in critical region

+84e	p_prior	initial priority
+84f	p_tocnt	number of elapsed time slices
+850	p_wait	scheduler flag x'00000000' - P_DONTCARE x'00000001' - P_INCORE x'FFFFFFFF' - P_OUTCORE
+852	p_cwait	incremented when event is expected, clear when event arrives, and is intended for catching event before the process is roadblocked
+854	p_chan	control channel number
+855	p_pcbpg	number of pages in this pcb. The following declaration indicates spare fields to allow field update of new process related information that will be used by the KERNEL
+856	p_c0	spare chars
+85c	p_s0	spare shorts
+860	p_schbuf	
+868	p_u0	unsigned for flags
+86c	p_fup	field update patch count
+870	p_ttg	time to go on the process time slice
+874	p_slice	time slice of the process
+878	p_ktime	time spent in kernel
+87c	p_kptime	time spent in kernel process
+880	p_stime	time spent in supervisor mode
+884	p_runtime	accumulated runtime, cleared each 60ms
+888	p_evflg	process event flags x'00000001' - E_USR16 x'00000002' - E_USR15 x'00000004' - E_USR14 x'00000008' - E_USR13 x'00000010' - E_USR12 x'00000020' - E_USR11 x'00000040' - E_USR10

		x’00000080’ - E_USR9
		x’00000100’ - E_USR8
		x’00000200’ - E_USR7
		x’00000400’ - E_USR6
		x’00001000’ - E_USR4
		x’00002000’ - E_USR3
		x’00004000’ - E_USR2
		x’00008000’ - E_USR1
		x’00010000’ - E_SYS16
		x’00020000’ - E_SIGCHAR
		x’00040000’ - E_SIGACK
		x’00080000’ - E_CHILD
		x’00100000’ - E_SYS12
		x’00200000’ - E_AUD
		x’00400000’ - E_UTIL
		x’00800000’ - E_RTIMEOUT
		x’01000000’ - E_INIT
		x’02000000’ - E_ABORT
		x’04000000’ - E_QIT
		x’08000000’ - E_INT
		x’10000000’ - E_HUP
		x’20000000’ - E_MSG
		x’40000000’ - E_TIMEOUT
		x’80000000’ - E_WAKEUP
+88c	p_evwait	mask for event wait flags x’00000000’ - P_EWANY
+890	p_evmsk	process event mask
+894	p_evpsd	psd save area at event entry
+894	ps_psw	processor status word
+898	ps_pa	program address
+89c	p_fpsd	psd save area at fault interrupt
+89c	ps_psw	processor status word
+8a0	ps_pa	program address
+8a4	p_topsd	psd save area at preemption or time-out
+8a4	ps_psw	processor status word
+8a8	ps_pa	program address
+8ac	p_initsp	initial value of the stack pointer
+8b0	p_tosave	register save area at preemption/time-out
+8f0	p_state	
+8f0	i_pa	
+8f4	i_psw	
+8f8	i_psbr	
+8fc	i_ssbr	
+900	i_reg	
+940	p_clist	capability list
+940	cp_owner	owner process

+944	cp_cap	capability
+9f0	p_disptch	number of dispatches since creation
+9f4	p_swapcnt	number of swap outs since creation Supervisor Process Segment List
+9f8	p_size	largest active segment number
+9fa	p_sglsz	maximum segment configuration
+9fc	p_seglist	segment list
+9fc	p_segflg	
+9fc	p_segndx:9	
	p_segflg:23	segment flag word (struct ssegf)

x’00000000’ - SF_FREE - free segment list entry
x’..000001’ - SF_EXEC - segment is executable
x’..000002’ - SF_WRT - segment is writable
x’..000004’ - SF_RD - segment is readable
x’..000008’ - SF_STK - segment is stack segment
x’..000010’ - SF_PWRT - process can make segment writable
x’..000020’ - SF_SHARE - segment is sharable
x’..000040’ - SF_NOLD - MMGR has failed to load segment
x’..000080’ - SF_NONSW - segment is nonswappable
x’..000100’ - SF_SBIT - segment belongs to the SUP process
x’..000200’ - SF_UBIT - belongs to a user process if the SF_SBIT is not set
x’..000400’ - SF_NXT - segment is needed next time the process is loaded
x’..000800’ - SF_NN - segment in current address space is needed now

+a00	p_segid	segment id

Structure: psw - length: 0x04

Source: head/psw.h

Location: The current psw is located in the ‘‘PSW’’ special register, preempted psw values are found on the interrupt stack, entry point psws are found in the DCT for kernel processes and in PCB segments for supervisors, and attachable entry point psws are found in the atchtbl[ ] array (of ative structures) in the kernel’s data segment.

Use: Used by the microcode to determine the required system environment for the currently running process.

+0	w_mode:2	processor mode x’0.......’ - W_MKRN x’4.......’ - W_MKP x’8.......’ - W_MSUP x’c.......’ - W_MUSR
+0.5	w_exlev:6	execution level x’.0.......’ - level 0 x’.1.......’ - level 1 " " " " x’.f.......’ - level f
+1	w_prvlg:4	privilege bits x’..1.....’ - W_SETEX x’..2.....’ - W_NMIO x’..4.....’ - W_SYSIO x’..8.....’ - W_WPSW
+1.5	w_emcntl:4	emulation control
+2	w_ssbr:3	secondary sbr
	w_psbr:3	primary sbr
	w_flag:6	bit flags b’..00,0001....’ - W_KSTK b’..00,0010....’ - W_SPARE b’..00,0100....’ - W_ISTK b’..00,1000....’ - W_MMON b’..01,0000....’ - W_SRC b’..10,0000....’ - W_DEST
+3.5	w_cond:4	condition codes x’.......1’ - W_CBIT x’.......2’ - W_NBIT x’.......4’ - W_VBIT x’.......8’ - W_ZBIT

Structure: sde - length: 0x20

Source: head/sde.h

Location: Found in the kernel’s address space starting at segment index 13 (0x1a0000). This address is dependent upon header va.h and may move from load to load.

Use: Memory management routines use the SDT to map all segments known to the system, either in memory or on the swap device.

+0	*s_pgtptr	virtual address of the page table
+4	*s_link	link for swappable segment sde’s or free sde’s
+8	s_plkcnt	process lock count
+9	s_lkcnt	I/O lock count
+a	s_nswcnt	nonswap count
+c	s_active	number of processes on the do not swap list, that have allocated the segment
+e	s_users	total number of processes that have allocated the segment
+10	s_lstpgsz	number of bytes used in the last page
+12	s_tlpg	total number of pages
+13	s_inmmpg	total number of pages in main memory
+14	s_stat	segment status word

x’00000000’ - SS_FREE - value of s_stat when the sde is free
x’00000001’ - SS_IOMSG - some i/o driver knows about the segment
x’00000002’ - SS_BROKE - segment is swapped out, and disk is bad
x’00000004’ - SS_WRT - segment is writable
x’00000008’ - SS_ONSL - the segment is on the swappable sde list
x’00000010’ - SS_PURGE - segment to be purged when it becomes unlocked
x’00000020’ - SS_REMOV - segment is being removed from memory
x’00000040’ - SS_UTLY - flag indicating segment contains breakpoints
x’00000080’ - SS_IOFAIL - i/o to read/write the segment has failed
x’00000100’ - SS_IOIN - segment is being read in
x’00000200’ - SS_IOOUT - segment is being written out
x’00000400’ - SS_LOCK - segment is locked
x’00000800’ - SS_NONSW - segment is nonswappable
x’00001000’ - SS_ALT - segment has been altered
x’00002000’ - SS_NEXT - segment belongs to a memory manager process
x’00004000’ - SS_ACT - segment belongs to a process on the nonswap list
x’00008000’ - SS_PLOCK - segment is process locked
x’00010000’ - SS_NSWSP - segment has no swap space
x’00020000’ - SS_BRKDN - segment is being broken down
x’00040000’ - SS_KPCB - kernel process pcb segment
x’00080000’ - SS_SPCB - supervisor process pcb segment
x’00100000’ - SS_BLOCK - segment is in the block state
x’00200000’ - SS_NEW - new segment and its swap space not initialized
x’00400000’ - SS_PGPRT - segment is protected on page basis
x’00800000’ - SS_ALLOC - the sdt entry is allocated
x’10000000’ - SS_ONFL - segment is on the free list

x'20000000' - SSMOD1 - segment is associated with module 1 of memory
x'40000000' - SSMOD0 - segment is associated with module 0 of memory

+18	s_swapaddr	block number of the starting point on the swap device
+1c	sde_name	segment name

Kboot Address Space

This section lists the control block structures in the kboot address space. The lists name each field and give the hexadecimal offset to the field from the beginning of the structure.

Structure: bootab - length: 0x1060

Source: head/sgenbt.h

Location: In the kernel’s address space somewhere near the end of the text segment under the external name ‘‘kbootab’’.

Use: Contains the mapping information used by kboot to create the segments and processes making up the boot.

+0	bt_ecdversion	ECD version number
+4	bt_nseg	number of valid entries in the bt_seg[ ] array
+6	bt_nprc	number of valid entries in the bt_prc[ ] array
+8	bt_ksdx[ ]	indices of the kernel’s bt_seg[ ] entries
+48	bt_seg[ ]	boot image segment descriptors - each is a bsegdes structure
+a48	bt_prc[ ]	boot image process descriptors - each is a bprcdes structure
+bc8	bt_kparm	kernel dynamic memory parameters - see the btkparm structure
+c0c	bt_npaths	number of processes to be pcreated
+c0e	bt_nlibs	number of public libraries to be loaded
+c10	bt_upath[ ]	pathnames of pcreated processes
+e68	bt_libpath[ ]	pathnames of boot public libraries

Structure: bprcdes - length: 0x18

Source: head/sgenbt.h

Location: In the kernel’s address space somewhere near the end of the text segment under the external name ‘‘kbootab’’ which contains an array of bprcdes structures.

Use: Contains the mapping information used by kboot to create the processes making up the boot.

+0	pd_pnum	fixed process number
+4	pd_class	process class
+8	pd_pcbsdx	index of the process’ (k)pcb segment in bt_seg[ ]
+a	pd_flags	process flags 0x00000001 - PD_KPRC - kernel process 0x00000002 - PD_SPRC - supervisor 0x00000200 - PD_CHILD - shares segment with child 0x00000400 - PD_PARENT - shares segment with parent 0x00000800 - PD_PROFIL - process being notified 0x00001000 - PD_DLMESSNTL - dlm essential 0x00002000 - PD_STATIC - static 0x00004000 - PD_NOTERM - noterm
+c	pd_prior	supervisor initial priority or kernel process execution level
+e	pd_spare	unused
+10	pd_nseg	number of boot image segments
+12	pd_nints	number of interrupts to attach
+14	pd_libflags	public library bit map

Structure: bsegdes - length: 0x14

Source: head/sgenbt.h

Location: In the kernel’s address space somewhere near the end of the text segment under the external name ‘‘kbootab’’ which contains an array of bsegdes structures.

Use: Contains the mapping information used by kboot to create the segments making up the boot.

+0	sd_kbva	virtual address in the kboot address space of the segment initial image
+4	sd_segsize	size of the segment in bytes
+8	sd_segndx	true segment index of the segment
+a	sd_users	number of processes using the segment
+c	sd_segflgs	segment flag word 0x00000007 - SD_ACCESS - segment protection flags 0x00000200 - SD_SHSHAR - LDP ’shared’ segment 0x00000400 - SD_SHCOM - LDP ’common’ option 0x00000800 - SD_PAS - PAS segment 0x00002000 - SD_ECD - ECD segment 0x00008000 - SD_KERNEL - segment is part of the kernel
+10	*sd_segid	segment id of the true segment. This field is filled in by kboot.

Structure: btkparm - length: 0x44

Source: head/sgenbt.h

Location: In the kernel’s address space somewhere near the end of the text segment under the external name ‘‘kbootab’’. This structure is contained in kbootab.

Use: Defines the kernel’s dynamic memory segments.

+0	km_nmsg	number of message blocks to be allocated
+2	km_nport	size of the port segment (in words)
+4	km_nprc	number of dct entries
+6	km_nsegecd	number of ecd segments
+8	km_nseg	number of SDT entries
+c	km_npgt	number of page tables
+10	km_npage	number of PDT entries
+14	km_istkb	size of the interrupt stack (in bytes)
+18	km_kstkb	size of the kernel stack (in bytes)
+1c	km_swstart	starting block of swap area
+20	km_swblks	size of swap area (in blocks)
+24	km_swmin	swap size of largest supervisor (in pages)
+28	km_intlen	initialization interval
+2c	km_maxlevs[ ]	application phase levels
+30	km_PASndx	segment index of low PAS
+34	km_1PASndx	segment index of high PAS
+38	km_PASdm	PAS dump/nodump flag
+3c	km_part_bound	partition boundary between mod 0 and mod1
+40	km_sched_tick	scheduler’s time-out value

File Manager Address Space

This section lists the control block structures in the file manager address space. The lists name each field and give the hexadecimal offset to the field from the beginning of the structure.

Structure:	bdevtab - length: 0x4c
Source:	head/fmgr/fmgr.h
Location:	An externally declared array called ‘‘bdevtab’’.
Use:	One entry for each block device driver (indexed by dcn). Internal file manager buffers are chained off an array of pointers using a hash selection of ’and’ing 0x7 to the block number.

+0	d_proc	process number of the driver
+4	boflag	if nonzero then the driver process is to get an open or close message with each open or close request. If zero then the driver only gets one open and one close.
+8	d_bchain[8]	buffer device chain pointer array
+8	b_forw	forward chain pointer
+c	b_back	backward chain pointer
+48	b_extra[4]	unused

Structure: buf - length: 0x40

Source: os/fmgr/head/buf.h

Location: An externally declared array called ‘‘buf’’. Free buffers are chained off of ‘‘bfreelist’’ (also a buf structure). Buffers associated with a device are chained off of a bdevtab entry. Buffers explicitly associated with no device are chained off of ‘‘bfreelist’’ (another chain).

Use: Each buf structure (2*NTASKS+4) controls an I/O buffer.

+0	*b_forw	buf pointer headed by bdevtab
+4	*b_back	buf pointer headed by bdevtab
+8	b_flags	buffer flags

0x00000002 - B_DONE - I/O complete
0x00000004 - B_ERROR - I/O error
0x00000008 - B_BUSY - buffer in use (locked)
0x00000030 - B_XMEM - memory extension (unused)
0x00000040 - B_WANTED - buffer wanted (task waiting)
0x00000080 - B_AGE - delayed write for correct aging
0x00000100 - B_ASYNC - no wait for completion
0x00000200 - B_DELWRI - delayed write (holds buffer between uses)
0x00000400 - B_IO I/O - outstanding on this buf
0x00000800 - B_MOUNT - superblock of a mounted filesys
0x00001000 - B_FSAUD - buffer being used by filesys audit
0x00002000 - B_DLM - disk limp mode indicator
0x00004000 - B_OVERLIM - io retry over limit
0x00008000 - B_BADMSG - bad data in io msg, and over retry limit
0x00010000 - B_BADBUF - crrpted data in io msg, and over retry limit
0x00020000 - 0B_IOFAIL - driver failure
0x000400000 - B_NOTRDY - device not active
0x000800000 - B_TRASHED - buffer trashed, do not reallocate for a while

+c	*av_forw	buf pointer headed by bfreelist
+10	*av_back	buf pointer headed by bfreelist
+14	b_mdct	device mdct-rid
+18	b_dcn	device major number
+1a	b_part	device partition
+1c	b_un	union
+1c	b_addr	address of actual buffer
+1c	*b_words	pointer to words for clearing
+1c	*b_filsys	pointer to superblock
+1c	*b_dino	pointer to block in ilist
+1c	*b_daddr	pointer to indirect block
+20	b_blkno	block # on device
+24	*b_mptr	ptr to mount table entry (null unless B_MOUNT set)

+28	b_taskid	taskid information (null unless B_BUSY set)
+2c	b_tstamp	time when io started (2 minutes from this time result in automatic task teardown)
+30	b_extra[16]	structure padding

Structure: cap - length: 0x18

Source: os/fmgr/head/cap_tbl.h

Location: An externally declared array called ‘‘cap_tbl’’.

Use: One is maintained for each open or fork.

+0	c_cap	capability
+4	c_pid	client process number
+8	*c_fptr	ptr to corresponding file table entry
+c	*c_cptr	ptr to next capability for this file (cap table entries are chained only on forks off of the same open, each points to the same file table entry)
+10	c_tstamp	time c_pid was first noted as invalid

Structure: cpmsghdr - length: 0x28

Source: head/cpmsghdr.h, see also head/fmgr/....

Location: Message buffer formats found in the message segment. Pointers to dequeued messages will be found in ‘‘tasktab’’ or one of the delayed_q’s.

Use: The means in which requests are made to the file manager. All message formats begin with a capability message header.

+0	cpm_mshd	standard message header (msghdr.h)
+0	ms_link	link to next message (delay queues only)
+4	ms_from	sending process
+8	ms_to	receiving process
+c	ms_nblks	message size in blocks
+d	ms_flags	message flags
+e	ms_type	message type (see types below)
+f	ms_stat	message status
+10	ms_size	message size in bytes
+12	ms_otype	original message type
+13	ms_seqnu	unused
+14	ms_ident	message identifier
+18	cpm_mc	capability field
+18	mc_num	capability number
+1c	cp_owner	owner process
+20	cp_cap	capability bits 0 -> 7: i_use count bits 8 -> 15: permissions bits 16 -> 31: file table index
+24	cpm_guid	group/user id
	type 0x01	FM_READ - read file
		request:
+28	fm_iosid	segment id
+30	fm_iobyoff	byte offset into segment
+34	fm_iocnt	number of bytes for I/O
+38	fm_ioblk	block number for I/O
+40	fm_iores	remaining bytes to be transferred
		reply:
+28	ret0	not used
+2c	ret1	not used
+30	fm_aiocnt	no bytes transferred
+34	fm_iortry	memory manager used field
+38	fm_err0	not used
+3c	fm_err1	error return from driver

	type 0x02	FM_WRITE - write file
		see type 0x01 request and reply
	type 0x03	FM_OPEN - open file
		request:
+28	fm_ocfoff	offset to name
+2c	fm_ocmode	mode for open or create

0x000 - OP_READ	- open: read
0x001 - OP_WRITE	- open: write
0x002 - OP_RW	
0x001 - CR_XBYOT	- create: execution by others
0x002 - CR_WBYOT	- create: write by others
0x004 - CR_RBYOT	- create: read by others
0x008 - CR_XBYG	- create: execute by group
0x010 - CR_WBYG	- create: write by group
0x020 - CR_RBYG	- create: read by group
0x040 - CR_XBYOW	- create: execute by owner
0x080 - CR_WBYOW	- create: write by owner
0x100 - CR_RBYOW	- create: read by owner
0x200 - CR_SVTXAX	- create: save text after execute
0x400 - CR_SGIDX	- create: set group id on execute
0x800 - CR_SUIDX	- create: set user id on execute

+30	fm_nopcr[ ]	filename
		reply:
+28	fm_ocapno	capability number (-1 for kernel process opens) (if pipe - cap num of read end)
+2c	fm_otype	type of file (if pipe - cap num of write end)
+30	fm_procid	process number of device file
+34	fm_unused	unused
+38	fm_odevid	mdct and partition
+40	fm_ofilsiz	file size
	type 0x04	FM_CLOSE - close file
+28	fm_usecnt	# file descriptors using inode
+2c	fm_clmode	close mode
	type 0x05	FM_EXEC - open file for execution
		request:
+28	fm_off	filename offset
+2c	fm_name[ ]	filename
		reply:
+28	fm_ecapno	capability number
+2c	fm_segname	unique segment name

	type 0x06	FM_FORK - increment count for open file
+28	fm_fkcapc	number of capabilities
+2c	fm_fkchpid	child process number
+30	fm_fkchpid	fcount increment/decrement
+34	fm_fkmce[ ]	list of capnums and caps
	type 0x08	FM_CREAT - create file
		see type 0x03 request and reply
	type 0x09	FM_LINK - link to a file
+28	fm_loff	offset to existing pathname
+2c	fm_loff1	offset to link name
+30	fm_nlink[ ]	existing and link names
	type 0x0a	FM_UNLINK - remove link from file
		see type 0x05 request
	type 0x0b	FM_UTIME - modify date of file
+28	fm_utoff	filename offset
+2c	fm_uflag	modification time flag
		0: modification time is current time
		1: modification time supplied by user
+30	fm_utime	utime structure
+30	f_atime	new access time
+34	f_mtime	new modify time
+38	f_ctime	new create time
+3c	fm_nutime[ ]	filename
	type 0x0c	FM_CHDIR - change directory
		request:
		see type 0x05 request
		reply:
+28	fm_chcapno	capability number
	type 0x0d	FMINIT - file manager initialization
+28	fm_idevid	mdct and partition of root device
+34	fm_idcnid	major device number root device
+38	fm_ipnum	root device process number
	type 0x0e	FM_MKNOD - make a node
+28	fm_mkoff	offset to name
+2c	fm_mkmode	permissions and file type
+30	fm_mkdvid	mdct and partition
+38	fm_mkdcn	major device number
+3a	fm_nmknod[ ]	name of the file
	type 0x0f	FM_CHMOD - change mode of file
+28	fm_chmoff	filename offset
+2c	fm_chmode	new mode of file
+30	fm_nchmod[ ]	filename
	type 0x10	FM_CHOWN - change owner of file
+28	fm_choff	filename offset

+2c	fm_uown	user id of new owner
+30	fm_gown	group id of new owner
+34	fm_nchown[ ]	filename
	type 0x11	FM_SYNC - update file systems on secondary
		capability header only
	type 0x12	FM_STAT - get file status
		request:
		see type 0x05 request
		reply:
+28	fm_stat	‘‘stat’’ structure
+28	st_dev	device number
+2c	st_ino	inode number
+30	st_mode	file mode (see mode field of inode)
+32	st_nlink	link count
+34	st_uid	user id
+36	st_gid	group id
+38	st_rdev	special file device name
+3c	st_size	length in bytes
+40	st_atime	time last accessed
+44	st_mtime	time last modified
+48	st_ctime	time created
	type 0x13	FM_SIZE - get file size
+28	fm_fsize	size of file (reply only)
+2c	fm_fssize	total amount allocated contiguous (reply only)
	type 0x14	FM_FSTAT - get status of open file
		request:
		capability header only
		reply:
		see type 0x12 reply
	type 0x15	FM_MOUNT - mount file system
+28	fm_moff	offset to device name
+2c	fm_moff1	offset to mount point name
+30	fm_mronly	action flags 0x00000001 - read only access 0x00000002 - audit the file system
+34	fm_nmount[ ]	device and mount point names
	type 0x16	FM_UMOUNT - unmount file system
		see type 0x05 request
	type 0x17	FM_MOVE - move file to contiguous area
		request:
		see type 0x05 request
		reply:
+28	fm_fmblk	number of blocks available
	type 0x18	FM_ALLOC - allocate contiguous space
		request:
+28	fm_faoff	filename offset

+2c	fm_famode	permissions and file type
+30	fm_fasize	file size
+34	fm_nfall[ ]	filename
		reply:
+28	fm_facapno	capability number
+2c	fm_fatype	type of file
+30	fm_faprocid	not used
+34	fm_fachan	not used
+38	fm_fancblk	number of blocks in file
	type 0x19	FM_MNTSTAT - mount status
	type 0x1a	FM_TASKAUD - high priority task audit
	type 0x1b	FM_NMCODE - get segment name
		not currently available
	type 0x1c	FM_ACCESS - check access permissions
+28	fm_acoff	filename offset
+2c	fm_acmode	access request
+30	fm_naccess[ ]	filename
	type 0x1d	FM_USTAT - pack label
		request:
+28	fm_mvdev	mdct and partition
		reply:
+28	fm_ustat	ustat structure
+28	f_tfree	total free
+2c	f_tinode	total inodes free
+30	f_fname[ ]	file system name
+36	f_fpack[ ]	file system pack name
	type 0x1e	FM_SEGCODE - return segment name
		request:
+28	fm_segcode	code byte
+29	fm_segflag	flag byte
		reply:
+28	fm_segname	unique name for segment
	type 0x1f	FM_TEMP - no disk writes on file
		see type 0x05 request
	type 0x20	FM_BACKOUT - restore (core copy of) file
		see type 0x05 request
	type 0x21	FM_PERM - untemp file (write to disk)
		see type 0x05 request
	type 0x22	FM_MV - windowless move
+28	fm_off	offset to ‘‘from’’ filename
+2c	fm_off1	offset to ‘‘to’’ filename
+30	fm_nmv[ ]	‘‘from’’ and ‘‘to’’ filenames
	type 0x23	FM_BUFRD - buffered read
		request:
+28	fm_bfsg1	1st segment id
+2c	fm_bfsg2	2nd segment id
+30	fm_bfoff	offset into 1st segment

+34	fm_bfcnt	number of bytes for I/O (max 128k)
		reply:
+28	ret0	not used
+2c	ret1	not used
+30	fm_aiocnt	# bytes transferred
+34	fm_iortry	memory manager used field
+38	fm_err0	no used
+3c	fm_err1	error return from driver
	type 0x24	FM_BUFWRT - buffered write
		see type 0x23 request and reply
	type 0x25	FM_LSEEK - lseek
		request:
+28	fm_lsoff	file offset
+2c	fm_lscmd	lseek command type
		reply:
+28	fm_lsaoff	resultant file offset
	type 0x26	FM_PIPE - open pipe
		see type 0x03 request and reply
	type 0x27	FM_ATOMSW - atomic switch
		see type 0x22
	type 0x28	FM_PERF - performance reporting
		request:
+28	fm_prtyp	performance request type
		0x00000000 FMP_REQ - report request frequency
		0x00000001 FMP_ERR - report error frequency
		0x00000002 FMP_MISC - report misc. info
		misc. info reply:
+28	fm_fault	fault count
+2c	fm_xfault	external faults
+30	fm_pfault	phase-1 faults
+34	fm_taskfault	task faults
+38	fm_init	initialization events
+3c	fm_util	utility events
+40	fm_retry	driver retry errors
+44	fm_unknown	unknown acknowledgements
+48	fm_bfhit	buffer hits
+4c	fm_bfmiss	buffer misses
+50	fm_bfgbusy	times buffer was busy
+54	fm_bfgempty	
+58	fm_bfgdelw	delayed writes
+5c	fm_nam	namei calls
+60	fm_restraint	restrained requests
+64	fm_teardown	task torn down
+68	fm_mntdown	mount table entries restored
+6c	fm_bufdown	buffers freed by teardown
+70	fm_inodown	inodes freed by teardown
+74	fm_fildown	file entries freed by teardown
+78	fm_sbidown	unlocks of SUPERB ilock
+7c	fm_sbfdown	unlocks of SUPERB flock
+80	fm_inocnt	active inode slots

+84	fm_inomax	high water active inode slots
+88	fm_filecnt	active file table slots
+8c	fm_filemax	highwater active file slots
+90	fm_extra	
	type 0x2f	FM_FSLAUD - file system link audit
	type 0x30	FM_FSBAUD - file system block audit
	type 0x31	FM_AUD - audit request from sim
	type 0x33	FM_FUAUD -msg from field update audit
	type 0x34	FM_SHRTRD -short buffered read
	type 0x35	FM_SHRTWRT -short buffered write
	type 0x36	FM_DIRSW - atomic directory switch
	type 0x47	FM_LAST -including all badmsg slots in fmsegs

Structure: delayed_q - length: 0x10

Source: head/fmgr/fmgr.h

Location: One externally declared array for each restraint queue, currently ‘‘open_q’’ and ‘‘mount_q’’.

Use: Certain file manager requests are throttled. This structure is used to chain requests which must wait for completion of earlier requests.

+0	q_count	# of active requests

+4	q_max	max # of active requests allowed
		for ‘‘open_q’’: 4 (NSOPEN)
		for ‘‘mount_q’’: 1 (NSMOUNT)

+8	*q_firstp	pointer to 1st message on queue
+c	*q_lastp	pointer to last message on queue

Structure: file - length: 0x20

Source: head/fmgr/file.h

Location: An externally declared array called ‘‘file’’.

Use: Contains file specific information with one entry for each original open (subsequent opens are chained).

+0	f_flags	file flags 0x00000001 - FLOCK - file table entry locked 0x00000002 - FWANT - another task wants (is asleep waiting)
+4	*f_iptr	pointer to incore inode
+8	*f_cptr	pointer to capability (chain)
+c	*f_fptr	pointer to next file entry on chain (there is one file table entry for each open, all point to the same inode)
+10	f_taskid	taskid information (valid only if FLOCK set)
+14	f_count	use count, > 1 indicates fork has occurred and is the number of chained cap_tbl entries
+18	f_ocount	previous (to this task) count (used for task teardown)
+1c	f_offset	read/write character pointer

Structure: filsys - length: 0x1dc

Source: head/sys/filsys.h

Location: The superblock of a file system. Mounted file systems will have the superblock in a buffer pointed to by a buf structure which is in turn pointed to by an entry in the mount table.

Use: Controls the access to the file system.

+0	s_isize	size in blocks of I list
+2	sent1	
+4	s_fsize	size in blocks of entire volume
+8	s_nfree	number of incore free blocks
+c	s_free[ ]	incore free blocks
	s_free[0]	ptr to blk containing 2nd free block array
	s_free[1]	ptr to blk containing 1st free block array
	s_free[2]-[49]	contains fsysdiag structure
+d4	s_ninode	number of incore free Inodes (index +1 into s_inode for next free inode)
+d6	s_inode[100]	incore free Inodes
+19e	s_flock	lock during free list manipulation
+19f	s_ilock	lock during I list manipulation
+1a0	s_fmod:1	super block modified flag
	s_ronly:1	mounted read-only flag
	s_ifull:1	ilist full flag
	s_bfull:1	blocks full flag
+1a4	s_time	current date of last update
+1a8	s_dinfo[4]	device information
+1b0	s_tfree	total free, for subsystem examination
+1b4	s_tinode	free inodes, for subsystem examination
+1b6	s_fname[6]	file system name
+1bc	s_fpack[6]	file system pack name
+1c4	s_ftaskid	(struct) taskid information (flock)
+1c8	s_itaskid	(struct) taskid information (ilock)
+1cc	s_cfree	free blocks on chain
+1d0	s_nxtblk	next free block not on chain (first zero in file system bit map)
+1d4	s_nxtcon	next available contiguous area
+1d8	s_label	file system label (0xdead3bcc)

Structure: inode - length: 0x60

Source: head/sys/inode.h

Location: The incore inode structure maintained in the file manager’s inode table in a segment by itself under the external name ‘‘inode’’ (currently segment address 0x2c0000).

Use: One for each active file, each current directory, each mounted-on file, and root.

+0	i_hchain	inode hash chain pointer
+4	i_flag	inode flags x’0001 - ILOCK - locked x’0002 - IUPD - has been modified x’0004 - IACC - update access time x’0008 - IMOUNT - mounted-on x’0010 - IWANT - process is waiting on lock x’0020 - ITRUNC - to be truncated x’0040 - ICRT - has been created x’0080 - IXSYNC - has been temped (don’t write inode to disk) x’0100 - ITRUNC2 - has been truncated without freeing blocks at least once. x’0200 - IFSAUD - being audited (fsaud) x’0400 - ITRSHD - fsaud says is trashed. x’0800 - IBACKUP - this is a backup inode. x’1000 - IREADING - inode read in progress. x’2000 - ISKIPSYNC - inode skipped during sync. x’4000 - IATOMSW - temped and atomic switched.
+8	i_count	total reference count (incremented each task use - inode slot not reused unless count goes to zero)
+9	i_wcount	reference count (writes)
+a	i_invoc	invocation count for inode on disk
+b	i_use	usage count for incore inode (incremented only when inode slot is assigned - put in capabilities)
+c	i_number	i number, 1-to-1 with device address
+e	i_mindx	mount table index at mount point
+f	i_unused	

+10	i_dev	mount table pointer of the file system containing this inode.
+14	i_taskid	(struct) taskid information (valid only if ILOCK set)
+18	i_tstamp	time stamp of last disk update
+1c	*i_fptr	pointer to file entry chain
+20	i_mode	mode of inode bit flags: x’0040 - IEXEC - execute permission x’0080 - IWRITE - write permission x’0100 - IREAD - read permission x’0400 - ISGID - set group id on execution x’0800 - ISUID - set user id on execution inode types: x’1000 - IFIFO - FIFO special x’2000 - IFCHR - character special x’3000 - IFMPC - multiplexed character x’4000 - IFDIR - directory x’5000 - IPIPE - *UNIX* system pipe x’6000 - IFBLK - block special x’7000 - IFMBP - multiplexed block x’8000 - IFREG - regular x’a000 - IFEXT - contiguous extents x’b000 - IF1EXT- one contiguous extent x’c000 - IFIOP - IOP special x’e000 - IFREC - record x’f000 - IFMT - inode file type mask
+22	i_nlink	directory entries (# of links)
+24	i_uid	owner
+26	i_gid	group of owner
+28	i_size	size of file - end of actual data
+2c	i_addr[ ]	disk addresses
+2c	i_addr[0]	direct block address (normal file) mdct-rid (special device file)
+30	i_addr[1]	direct block address (normal file) dcn (special device file)
+34	i_addr[2]	direct block address (normal file) partition (special device file) name (FIFO or pipe special device file)
+38	i_addr[3]->[9]	direct block addresses (normal file)
+54	i_addr[10]	single indirect block address (normal file)
+58	i_addr[11]	double indirect block address (normal file)
+5c	i_addr[12]	triple indirect block address (normal file)

Structure: measf - length: 0x234

Source: os/fmgr/head/meas.h

Location: An externally declared structure named ‘‘mf’’.

Use: A collection of counters which characterize the activity of the file manager since the last boot.

+0	m_fill1	x’dead3b indicates start
+4	m_fhist[FM_LAST+1]	frequency histogram for requests
+124	m_fill2	x’dead3b indicates end m_fhist
+128	m_err[NERR]	frequency histogram for errors
+1f0	m_fill3	x’dead3b indicates end of m_err
+1f4	m_fault	# of faults
+1f8	m_xfault	# of external faults
+1fc	m_pfault	# of phase 1 faults
+200	m_taskfa	# of task faults
+204	m_retry	# of retry errors from drivers
+208	m_unknow	# of unknown acks received
+20c	m_bfhit	# of buffer hits
+210	m_bfmiss	# of buffer misses
+214	m_bfgbus	# of times buffer was busy
+218	m_bfgemp	unknown
+21c	m_bfgdel	# of delayed writes
+220	m_restra	# of restrained requests
+224	extra[16]	

Structure: mnttab - length: 0x54

Source: head/mnttab.h

Location: An externally declared array called ‘‘mnttab’’.

Use: One entry for each mounted file system telling the pathname and file system name.

+0	mt_dev	device filename
+20	mt_filsys	file system name
+40	mt_ro_fl	read only flag
+44	mt_time	time stamp
+48	mt_devid	new device id
+48	dev_mdct	
+4c	dev_part:16	
+50	mt_dcn	major device

Structure: mount - length: 0x48

Source: head/fmgr/fmgr.h

Location: An externally declared array called ‘‘mount’’.

Use: One entry is maintained for each file system mount. Primarily used for file access across file systems. Points to the inode of the root directory of the file system as well as the inode of the mount point (both of which have their use counts incremented).

+0	m_mdct	device mdct-rid
+4	m_dcn	device major number
+6	m_part	device partition
+8	*m_bptr	pointer to superblock bfr header
+c	*m_fcbptr	pointer to free chain bfr header
+10	*m_iptr	pointer to mounted inode (inode of directory mounted upon)
+14	*m_rootp	pointer to root inode of filesys
+c0	m_use	use count bits 0 -> 4: has been audited flag bits 5 -> 7: audit flags
+c8	m_audited	file system has been audited
+cc	m_audflag	file system audits in progress
+1c	m_taskid	taskid information
+20	m_prevmn	flag for previous mount (used for task teardown)
+21	m_rdonly	flag for previous read only (used for task teardown)
+22	m_syncmax	number of syncs since last SB write
+23	m_spare1	unused
+24	m_fsinfo	fsysdiag structure
+24	fs_mts	mount time stamp
+28	fs_opens	opens since mount
+2c	fs_close	closes since mount
+30	fs_reads	total direct reads
+34	fs_writes	total direct writes
+38	fs_seio	buffered block I/O errors
+3a	fs_bdeio	buffered data block I/O errors
+3c	fs_deio	not-buffered data block I/O errors
+40	fs_aflag	file system audit flags
+42	fs_audcnt	audits since mount
+44	fs_blker	block audit error count
+46	fs_lnker	link audit error count

Structure: msgbufe - length: 0x10

Source: head/msgbufe.h

Location: In the kernel’s address space at 0x360000 (may vary per head/va.h). The index into the segment for each element is the same as the index into the message segment for its corresponding message buffer.

Use: A kernel private segment used for queuing and auditing the message buffers.

+0	*me_link	message queue link field
+4	me_owner	current owner of the associated message buffer
+8	me_tstamp	the time this message buffer last changed status (allocated buffers only)
+c	me_blks	total blocks allocated
+d	me_flags	message extender flags x’04’ - ME_AUDRPT - old non-queued message x’08’ - ME_ALOC - allocated x’10’ - ME_AUDERR - old message (audit flag) x’20’ - ME_EXTTAG - old queued message x’40’ - ME_ONQUE - currently on a message queue x’80’ - ME_OFFQUE - has been dequeued at least once
+e	me_type	message type of the associated message buffer
+f	me_unused	unused at this time

Structure: tasktab - length: 0x800

Source: os/fmgr/task.h

Location: One per task segment as declared in task0.c through task15.c (currently segment addresses 0x40000 through 0x220000). The addresses of each of the tasks are located in an externally declared array called ‘‘tasktab’’. The declaration ‘‘taskptr’’ points to the currently active task.

Use: Each defines the state of its associated task.

+0	t_status	task status x’00000000’ - TAVAIL - slot available x’00000001’ - TINUSE - slot in use x’00000002’ - TREADY - ready to run x’00000004’ - TACTIVE - currently running x’00000008’ - TNOACK - do not ack message x’00000010’ - TFUNC - function completed x’00000020’ - TDECQU - restraint queue decremented x’00000040’ - TDOWN - tear task down when audit runs x’00000080’ - TUNLKED - task did uniolock
+4	t_taskid	task identification
+8	*t_slpaddr	task sleep address
+c	*t_stack	pointer to task stack
+10	*t_msg	pointer to recvd message
+14	t_err	task error status x’00000000 - ENOERR - no error has occurred x’00000001 - EPERM - Not super-user x’00000002 - ENOENT - No such file or directory x’00000003 - ESRCH - No such process x’00000004 - EINTR - Interrupted system call x’00000005 - EIO - I/O error x’00000006 - ENXIO - No such device or address x’00000007 - E2BIG - Arg list too long x’00000008 - ENOEXEC - Exec format error x’00000009 - EBADF - Bad file number x’0000000a - ECHILD - No children x’0000000b - EAGAIN - No more processes x’0000000c - ENOMEM - Not enough core x’0000000d - EACCES - Permission denied x’0000000e - EFAULT - Bad address x’0000000f - ENOTBLK - Block device required x’00000010 - EBUSY - Mount device busy x’00000011 - EEXIST - File exists x’00000012 - EXDEV - Cross-device link x’00000013 - ENODEV - No such device x’00000014 - ENOTDIR - Not a directory x’00000015 - EISDIR - Is a directory x’00000016 - EINVAL - Invalid argument

		x’00000017 - ENFILE - File table overflow x’00000018 - EMFILE - Too many open files x’00000019 - ENOTTY - Not a typewriter x’0000001a - ETXTBSY - Text file busy x’0000001b - EFBIG - File too large x’0000001c - ENOSPC - No space left on device x’0000001d - ESPIPE - Illegal seek x’0000001e - EROFS - Read only file system x’0000001f - EMLINK - Too many links x’00000020 - EPIPE - Broken pipe x’00000021 - ETEMP - Temped file x’00000022 - ENOTRAP x’00000023 - ENOMSG - no message x’00000024 - ENOALOC - not allocated x’00000025 - EAUD - mount audit failure x’00000026 - EFSAUD x’00000027 - EFIRST - first access of logical block x’00000028 - ENOMOVE - fmove failed x’00000029 - ENOEXT - no extents x’0000002a - EPATH - pathname too long x’0000002b - ETABLE - no entries left x’0000002c - EFUNC - invalid operation x’0000002d - EFMAUD - failure due to an audit x’0000002e - EDLM - disk limp mode indication x’0000002f - EDISKRM - task torndown - disk remove x’00000030 - EMBUFAUD - message buffer audit failure x’00000031 - ECANNOTCOMPLY
+18	t_tstamp	task start time stamp
+1c	t_patchcnt	system patch cnt at task start
+20	t_proc	temp storage of proc number when device driver created or opened
+24	t_ncblks	temp storage of #/contig blks
+28	t_offset	temp storage of directory offset
+2c	t_dent	temp storage of directory entry
+2c	d_ino	inode number
+2e	d_name[ ]	last component of pathname
+3c	*t_pdir	temp storage of ptr to incore directory inode
+40	t_start	used for performance task timing
+44	t_dbuf[DIRSIZ]	temp storage of pathname component
+52	t_sdfpath	special device file pathname for ECD access

+92	t_bufreq	buffered-io flag x'00' - no x'01' - yes
+93	t_extra[16]	structure padding
+a4	t_stackarea	task private stack (remainder of page)

UCB Record

This section provides the offset for the ucb_rec structure which is used by various processes.

Descriptions of Register Layouts

10

Contents

Descriptions of Register Layouts

10

Introduction

This section provides selected 3B21D/3B20D computer register layouts useful for troubleshooting software problems. For detailed descriptions on these and other 3B20D/3B21D registers, refer to 254-303-105, *3B21D Computer Hardware Reference Manual* and 254-303-106, *3B20D and 3B21D Computers UNIX® RTR Operating System Maintenance Manual*. The registers are arranged in alphabetical order by name. Tables 10-1 through 10-8 shows the register layouts. The bits and bytes in all registers are numbered as follows:

bit	31......24	23 16	15 8	7 0
byte	0	1	2	3

* UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Table 10-1. Error Register (ER)*

Bit(s)	Error	Class
0	Source bus/bit rotate parity error	Stop-and-switch
1	Microcontrol parity error (MIR parity)	Stop-and-switch
2	Clock match error (mismatch of auxiliary and primary results clock)	Stop-and-switch
3	IB parity error	Stop-and-switch
4	Address translation buffer (ATB) parity error†	Stop-and-switch/ microinterrupt
5	Cache error	Stop-and-switch
6	My store error A (MYSERA)	Stop-and-switch
7	My store time-out error†	Microinterrupt/stop-and-switch
8	My store data (MYSERC) parity error†	Microinterrupt/on-line error interrupt
9	Data manipulation unit (DMU) error	Stop-and-switch
10	Store Address Controller (SAC) error	Stop-and-switch
11	Invalid maintenance channel (MCH) error	Off-line error interrupt
12	Other store error A (OTHSERA)	Off-line error interrupt
13	Other store refresh parity error (OTHSERD)	Off-line error interrupt
14	Other store data parity error (OTHSERC)	Off-line error interrupt
15	Other store time-out error† microinterrupt	Off-line error interrupt/
16	Channel error (CER)	On-line error interrupt
17	Input/Output (I/O) response error	On-line error interrupt
18	I/O addressing/pulse point error	On-line error interrupt
19	Parity divert error	On-line error interrupt
20	My store refresh parity error (MYSERD)	On-line error interrupt
21	Protection violation†	Microinterrupt/software error interrupt
22	Virtual address out-of-range (VORA)	Microinterrupt/software error interrupt
23	Out-of-range address (MYSERB) †	Microinterrupt/software error interrupt

See footnotes at end of table.

Table 10-1. Error Register (ER)* (Contd)

Bit(s)	Error	Class
24	Out-of-range reference, other store (OTHSERB)	Software error interrupt
25	Privileged instruction error†	Microinterrupt/software error interrupt
26	Bad alignment on memory reference	Software error interrupt
27	Reserved	-
28-31	Source bus parity bits	-

* This register is active low except for the source bus parity bits. When a bit is equal to zero, the corresponding error condition is present.

† These error bits are copied into the microinterrupt error register (UER) and cleared in the error register on a microinterrupt.

Table 10-2. Interrupt Source (IS)* Register

Bit	Meaning
0	On-line error interrupt
1	Other control unit (CU) error interrupt
2	Software error interrupt
3	Reserved
4	Reserved
5	TIMERS interrupt
6	Reserved
7	Reserved
8	Utility circuit interrupt
9	Stop the world interrupt
10	Direct memory access (DMA) and CH11 and CH13 interrupts (optional for CH12, CH14, and CH16-CH19)
11	(optional CH12, CH14, and CH16-CH19)
12-15	Reserved
16	Emergency action interface (EAI) interrupt
17	Programmed interrupt request (PIR) 15
18	Programmed interrupt request (PIR) 14
19	Programmed interrupt request (PIR) 13
20	Programmed interrupt request (PIR) 12
21	Programmed interrupt request (PIR) 11
22	Programmed interrupt request (PIR) 10
23	Programmed interrupt request (PIR) 9
24	Programmed interrupt request (PIR) 8
25	Programmed interrupt request (PIR) 7
26	Programmed interrupt request (PIR) 6
27	Programmed interrupt request (PIR) 5
28	Programmed interrupt request (PIR) 4
29	Programmed interrupt request (PIR) 3
30	Programmed interrupt request (PIR) 2
31	Programmed interrupt request (PIR) 1

* This register is active low (if bit=0, function is asserted).

Table 10-3. Microinterrupt Error Register (UER)*

Bit	Error	Class
0	Not used	-
1	Not used	-
2	Not used	-
3	Not used	-
4	Address translation buffer (ATB) error	Stop-and-switch
5	Not used	-
6	Not used	-
7	My store timeout	Stop-and-switch
8	My store noncorrectable error (MYSERC)	On-line error interrupt
9	Not used	-
10	Not used	-
11	Not used	-
12	Not used	-
13	Not used	-
14	Not used	-
15	Other store timeout	Off-line error interrupt
16	Not used	-
17	Not used	-
18	Not used	-
19	Not used	-
20	Not used	-
21	Protection violation	Software error interrupt
22	VORA error	Software error interrupt
23	My store out-of-range reference MYSERB	Software error interrupt
24	Not used	-
25	Privileged instruction violation	Software error interrupt
26	Not used	-
27	Not used	-
28	Not used	-
29	Not used	-
30	Not used	-
31	Not used	-

* This register is active low (if bit = 0, function is asserted). This register is also known as firm register B.

Table 10- 4. Microinterrupt Error Register 1 (UER1)

Value	Error	Class
00000001	Segment index too large	Software error interrupt
00000002	Segment invalid	Software error interrupt
00000008	Page invalid	Software error interrupt
FFFFFFFF	I/O parity divert error if found on Hardware Interrupt with ER bit 19 active	Hardware error interrupt
7FFFFFFF	Illegal switch from private to kernel stack	Software error interrupt
3FFFFFFF	Illegal instruction	Software error interrupt
1FFFFFFF	Illegal instruction or operand subdecode	Software error interrupt
0FFFFFFF	No interrupt source during external interrupt entry	Software error interrupt
07FFFFFF	Unused microstore location was executed	Software error interrupt
03FFFFFF	No error showing during error microinterrupt entry	Software error interrupt
01FFFFFF	Unable to flush contents of ATB in purge ATB instruction	Software error interrupt
0001FFFF	Virtual address out-of-range error. Tried to read memory out of virtual address space, beyond 64 megabytes	Software error interrupt
12345678	MRF caused by the microcode or software	Software error interrupt
12345679	Processor going through level 2 [stop-and-switch (SAS)] MRF	Software error interrupt
1234567A	Stop-and-switch due to error microinterrupt that proved fatal	Software error interrupt

* UER1 is also known as firm register C.

Table 10-5. Processor Status Word (PSW) Register*

Bit	Description
0	Carry flag
1	Negative flag
2	Overflow flag
3	Zero flag
4	Kernel stack on (yes=1)
5	Not used
6	Interrupt stack on (yes=1)
7	Memory management on (yes=1)
8	Source: SSBR Mode=1, PSBR Mode=0
9	Destination: SBR Mode=1, PSBR Mode=0
10	Primary segmentation base register index
11	Primary segmentation base register index
12	Primary segmentation base register index
13	Secondary segmentation base register index
14	Secondary segmentation base register index
15	Secondary segmentation base register index
16	Emulation control: opcode decoding, spare in 3B21D computer
17	Emulation control: opcode decoding, spare in 3B21D computer
18	Emulation control: program counter increment: half-word=0, fullword=1, spare in 3B21D computer
19	Emulation control: interrupt control — control interrupt recognition when instruction halfword is available, spare in 3B21D computer
20	Set execution level privilege (yes=1)
21	Normal I/O privilege (yes=1) for 3B20D computer, Maintenance privilege (yes=1) for 3B21D computer
22	System I/O privilege (yes=1)
23	Write processor status word (PSW) privilege (yes=1)
24	Execution level (0-15)
25	Execution level
26	Execution level
27	Execution level

* See footnote at end of table.

Table 10-5. Processor Status Word (PSW) Register* (Contd)

Bit	Description
28	Spare
29	Spare
30	Processor mode
31	Processor mode 00 kernel mode 01 kernel process 10 supervisor process/*UNIX*† system process 11 user mode process/*UNIX* system process

* The PSW register is active high (if bit=1, function is asserted). In the 3B21D computer hardware platform, the emulation bits (bits 16, 17, 18, and 19) are not used.

† UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Table 10-6. Pulse Point Register (PPR)*

Bit(s)	Meaning
0	Read the channel data buffer (RD)
1	Read the channel status register (RST)
2	Read channel interrupt state (RINT)
3	Read channel service request (RSR)
4	I/O interrupt acknowledge (IAK)
5	Channel error acknowledge (EACK)
6	Service request acknowledge (SRACK) for 3B20D computer, unused in 3B21D computer
7	Idle channel sequencer (IDLE)
8	Clear channel errors (CLRER)
9	Write the channel data buffer (WD)
10	Write channel control/address Register (WCA)
11	EAI for 3B21D computer, unused in 3B21D computer
12-13	Not used
14	Backup maintenance channel
15	Backup maintenance channel
16	BGB pulse points
17	BGB pulse points
18	BGB pulse points
19	BGB pulse points
20	BGB pulse points
21	BGB pulse points in 3B21D computer, Test sync for 3B20D computer
22	Interrupt EAI
23	Interrupt EAI
24	Clear timer
25	Increment timer
26	Test sync for 3B21D computer
27	I/O read clock
28	I/O response clock
29	Error register clear
30	Enable a
31	Enable b

* This register is active high (if bit = 1, function is asserted).

Table 10-7. Store Control Register (SCR)*

Bit	Description
0-2	Invalidation counter bits muxed with SAR11-SAR13.
3-7	Invalidation counter bits muxed with SAR17-SAR21.
8-10	ATB block select. Each muxed with corresponding PSBR and SSBR bit.
11	scr_hwa: Half-word available
12-15	scr_pashdw: Program address (PA) shadow
16	scr_acc: Access (store go)
17	scr_f: Fetch (read)
18	scr_w: Write
19	scr_c: Clear for 3B20D computer, read-modify-write for 3B21D computer
20	scr_byatb: Bypass ATB
21	scr_byte: Byte access
22	scr_half: Half-word access (requires SAR00 to be 0)
23	scr_stk: Stack mode (always a cache write)
24	scr_sarauto: SAR auto-increment
25	Memory arbiter reset for 3B21D computer, unused in 3B20D computer
26	Enable counter in very large main memory (VLMM) for 3B20D computer, ATB flush counter enable for 3B21D computer
27	scr_atbsl: ATB select (=1 select ATBA, =0 select ATBB)
28	Counter select
29	Invalidate ATB
30	scr_os: Other store go
31	scr_mtce: Maintenance mode (also terminates store operations)

* This register is active low (if bit=0, function is asserted).

Table 10-8. System Status Register (SSR)*

Bit	Description
0	CU identification (CU0 = 1, CU1 = 0)
1	microcode hardware decision bit: indicates ECSU for RTR R1 (3B20D computer), indicates VLMM MASC for RTR R6 (3B20D computer), LDFT tape drive indicator for 3B21D computer (SCSI = 1, IOPTAPE = 0)
2	(Simplex=1, duplex=0) for 3B20D computer, reserved in 3B21D computer
3	Request out-of-service key
4	Initialization sequence control
5	Initialization sequence control
6	Force boot device primary
7	Force boot device secondary
8	Panel interrupt
9	Force off-line and inhibits CC I/O
10	Force on-line (converts stop-and-switch to MRF)
11	Inhibit sanity timer
12	Enable panel interrupt for 3B21D computer, unused in 3B20D computer
13	Cache bypass
14	Emergency action interface MRF
15	Power clear
16	CC on line
17	Halt
18	Block interrupts
19	Enable update writes
20	Isolate DMA from my store
21	Isolate update from my store
22	Isolate expansion slots from my store for 3B21D computer, unused in 3B20D computer
23	Block hardware checks
24	EAI bus
25	EAI bus
26	EAI bus
27	EAI bus

* See footnote at end of table.

Table 10-8. System Status Register (SSR)* (Contd)

Bit	Description
28	Stop
29	Block timer circuit
30	I/O disable
31	Power key

* This register is active low (if bit = 0, function is asserted) unless otherwise indicated.

Conversion Chart

11

Contents

Conversion Chart

11

ASCII/Hexadecimal Conversion Chart

Table 11-1 equates American Standard Code for Information Exchange (ASCII) characters to their respective hexadecimal values.

Table 11-1. ASCII/Hexadecimal Conversion Chart

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht	nl	vt	np	cr	so	si
1	dle	dc1	dc2	dc3	dc4	nak	syn	etb	can	em	sub	esc	fs	gs	rs	us
2		!	"	#	$	%	&	’	(	)	*	+	,	–	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	__
6	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	≈	del

Glossary

A

ASCII
American Standard Code for Information Interchange.

B

BWM
Broadcast Warning Message.

C

CCIO
Central Control Input/Output.

CTIP
Customer Training and Information Products.

CU
Control Unit.

D

DAP
Display Administrative Process.

DAT
Digital Audio Tape.

DCI
Dual Serial Channel Computer Interconnect.

DCT
Dispatcher Control Table.

DDLP
Data Definition Language Processor.

DFC
Disk File Controller.

DGN
Dispatcher Control Table.

DMA
Direct Memory Access.

DML
Data Manipulation Language.

DUC
Dual Utility Circuit.

E

EGRASP
Enhanced Generic Access Package.

ECD
Equipment Configuration Database.

EMM
Extended Main Memory.

EOF
End of File.

EOT
End of Tape (marker).

F

FTS
Field Test Set.

G

GRASP
Generic Access Package.

I

IP Information Product.

L

LLA
Low-Level Access.

M

MHD
Moving Head Disk.

MML
Man-Machine Language.

MRF
Maintenance Reset Function.

MTTY
Maintenance Terminal Teletypewriter.

O

OST
Operating System Trap.

OOS
Out of Service.

P

PA Program Address.

PCB
Process Control Block.

PDS
Program Documentation Standard.

PDT
Physical Disk-to-Tape.

PGT
Page Table.

PID
Process Identification number.

PMDB
Plant Measurement Database.

PMS
Plant Measurement System.

PRM
Processor Recovery Messages.

PSBR
Primary Segmentation Base Register.

R

RID
Record Identification.

ROP
Receive-Only Printer

RTR
Real-Time Reliable.

S

SCC
Switching Control Center.

SCSI
Small Computer System Interface.

SDE
Segment Descriptor Entry.

SDP
Software Demand Paging.

SIM
System Integrity Monitor.

T

TTYC
Teletypewriter Controller.

U

UC
Utility Circuit.

UCB
Unit Control Block.

UCL
Unconditional.

UID
Utility Identification.

ULARP
User-Level Automatic Restart Process.

V

VLMM

Very Large Main Memory.