CONVERGENT TECHNOLOGIES

DOCUMENTATION UPDATE for CTIX OPERATING SYSTEM MANUAL ASSEMBLIES

Version C, Second Edition, Volumes 1 thru 4

(SAC-20Bx,SAC-2014x,SAC-20Cx,SAC-20Fx)

Revised August 15, 1990

B-09-01977-01-F

TABLE OF CONTENTS

1. General Description of the Documentation Update

This Documentation Update is for the CTIX Operating System Manual, Version C, Second Edition. It contains manual pages for utilities provided on 6.2-2 CTIX releases (with number of pages for each).

The pages to be included in the *CTIX Operating System Manual, Volume 1, are:*

1. crash(1M) - examine system images (11p)
2. hinv(1M) - hardware inventory (2p)
3. iopdump(1M) - upload a Front-end I/O Processor's RAM (1p)

The pages to be included in the *CTIX Operating System Manual, Volume 2, are:*

1. serstat(1M) - display serial port error statistics (2p)

iopdump(1M) is a new utility, so this is the first version of a man page.

The pages to be included in the *CTIX Operating System Manual, Volume 3,* are:

1. gettimeofday(2) - get/set date and time (2p)
2. notify(2) - manage notifications (5p)
3. monitor(3C) - prepare execution profile (2p)
4. sleep(3C) - suspend execution for interval (1p)

NAME

crash - examine system images

SYNOPSIS

/etc/crash [**-d** dumpfile] [**-n** namelist] [**-w** outputfile]

DESCRIPTION

The *crash* command is used to examine the system memory image of a live or a crashed system by formatting and printing control structures, tables, and other information. Command line arguments to *crash* are *dumpfile, namelist,* and *outputfile.*

Dumpfile is the file containing the system memory image. The default *dumpfile* is **/dev/kmem**. The system image can also be slice zero of the raw disk that contains the dump area (for example, **/dev/rdsk/c0d0s0**); or it can be the pathname of a file produced using *dd* to copy slice zero or just the dump area; or in the case of a tape dump, the second file on the tape.

The unstripped executable file *namelist* contains the symbol table information needed for symbolic access to the system memory image to be examined. The default *namelist* is **/etc/lddrv/unix.exec** if examining a running system or **/etc/lddrv/prev.unix.exec** if examining a dump. If neither of these files exists, the default is **/unix**. If a system image from another machine is to be examined, the corresponding **prev.unix.exec** must be copied from that machine. The **prev.unix.exec** is preferred to **/unix** because it also contains the *namelist* for all the loaded drivers at the correct addresses.

When the *crash* command is invoked, a session is initiated. The output from a *crash* session is directed to *outputfile.* The default *outputfile* is the standard output.

Input during a *crash* session is of the form:

```
function [ argument ... ]
```

where *function* is one of the *crash* functions described in the FUNCTIONS section of this manual page, and *arguments* are qualifying data that indicate which items of the system image are to be printed.

The default for process-related items is the current process for a running system and the process that was running at the time of the crash for a crashed system. If the contents of a table are being dumped, the default is all active table entries.

The following function options are available to *crash* functions wherever they are semantically valid:

-e	Display every entry in a table.
-f	Display the full structure.
-p	Interpret all address arguments in the command line as *physical* addresses.
-s process	Specify a process slot other than the default.
-w file	Redirect the output of a function to *file*.

Note that if the **-p** option is used, all address and symbol arguments explicitly entered on the command line will be interpreted as physical addresses. If they are not physical addresses, results will be inconsistent.

The functions *mode*, *defproc*, and *redirect* correspond to the function options **-p**, **-s**, and **-w**. The *mode* function may be used to set the address translation mode to physical or virtual for all subsequently entered functions; *defproc* sets the value of the process slot argument for subsequent functions; and *redirect* redirects all subsequent output.

Output from *crash* functions may be piped to another program in the following way:

```
function [ argument ... ] ! shell_command
```

For example,

```
mount ! grep rw
```

will write all mount table entries with an *rw* flag to the standard output. The redirection option (-w) cannot be used with this feature.

Depending on the context of the function, numeric arguments will be assumed to be in a specific radix. Counts are assumed to be decimal. Addresses are always hexadecimal. Table address arguments larger than the size of the function table will be interpreted as hexadecimal addresses; those smaller will be assumed to be decimal slots in the table. Default bases on all arguments may be overridden. The C conventions for designating the bases of numbers are recognized. A number that is usually interpreted as decimal will be interpreted as hexadecimal if it is preceded by *0x* and as octal if it is preceded by *0*. Decimal override is designated by *0d*, and binary by *0b*.

Aliases for functions may be any uniquely identifiable initial substring of the function name. Traditional aliases of one letter, such as *p* for *proc*, remain valid.

Many functions accept different forms of entry for the same argument. Requests for table information will accept a table entry number, a physical address, a virtual address, a symbol, a range, or an expression. A range of slot numbers may be specified in the form *a-b* where *a* and *b* are decimal numbers. An expression consists of two operands and an operator. An operand may be an address, a symbol, or a number; the operator may be +, -, *, /, &, or |. An operand which is a number should be preceded by a radix prefix if it is not a decimal number (*0* for octal, *0x* for hexidecimal, *0b* for binary). The expression must be enclosed in parentheses (). Other functions will accept any of these argument forms that are meaningful.

Two abbreviated arguments to *crash* functions are used throughout. Both accept data entered in several forms. They may be expanded into the following:

table_entry = table entry | address | symbol | range | expression

start_addr = address | symbol | expression

FUNCTIONS

? [**-w** file] List available functions.

!cmd Escape to the shell to execute a command.

adv [**-e**] [**-w** file] [[**-p**] table_entry ...]
Print the advertise table.

base [**-w** file] number ...
Print *number* in binary, octal, decimal, and hexadecimal. A number in a radix other then decimal should be preceded by a prefix that indicates its radix as follows: *0x*, hexidecimal; *0*, octal; and *0b*, binary.

buffer [**-w** file] [-format] bufferslot

or

buffer [**-w** file] [-format] [**-p**] start_addr
Alias: **b**.
Print the contents of a buffer in the designated format. The following format designations are recognized: **-b**, byte: **-c**, character; **-d**, decimal; **-x**, hexadecimal; **-o**, octal; **-r**, directory; and **-i**, inode. If no format is given, the previous format is used. The default format at the beginning of a *crash* session is hexadecimal.

bufhdr [**-f**] [**-w** file] [[**-p**] table_entry ...]
Alias: **buf**.
Print system buffer headers.
The **-f** option produces different output depending on whether the buffer is local or remote (contains RFS data).

callout [**-w** file]
: Alias: **c**.
: Print the callout table.

cblk [**-e**] [**-p**] [**-w** file] [**-t** type] [table_entry ...]
: Display contents of cblocks.

clist [**-e**] [**-p**] [**-w** file] [**-t** type] [table_entry ...]
: Display usage of clists.

conbuf [**-w** file]
: Display console buffer.

dballoc [**-w** file] [class ...]
: Print the dballoc table. If a class is entered, only data block allocation information for that class will be printed.

dbfree [**-w** file] [class ...]
: Print free streams data block headers. If a class is entered, only data block headers for the class specified will be printed.

dblock [**-e**] [**-w** file] [**-c** class ...]

or

dblock [**-e**] [**-w** file] [[**-p**] table_entry ...]
: Print allocated streams data block headers. If the class option (**-c**) is used, only data block headers for the class specified will be printed.

defproc [**-w** file] [**-c**]

or

defproc [**-w** file] [slot]
: Set the value of the process slot argument. The process slot argument may be set to the current slot number (**-c**) or the slot number may be specified. If no argument is entered, the value of the previously set slot number is printed. At the start of a *crash* session, the process slot is set to the current process.

dis [**-w** file] [**-a**] start_addr [count]
: Disassemble from the start address for *count* instructions. The default count is 1. The absolute option (**-a**) specifies a non-symbolic disassembly.

disk [**-w** file]
: Display disk information.

drvtable [**-w** file]
: Display loadable driver table information.

ds [**-w** file] virtual_address ...
Print the data symbol whose address is closest to, but not greater than, the address entered.

fcallout [**-w** file]
Alias: **fc**.
Print the fast callout table.

file [**-e**] [**-w** file] [[**-p**] table_entry ...]
Alias: **f**.
Print the file table.

findaddr [**-w** file] table slot
Print the address of *slot* in *table*. Only tables available to the *size* function are available to *findaddr*.

findslot [**-w** file] virtual_address ...
Print the table, entry slot number, and offset for the address entered. Only tables available to the *size* function are available to *findslot*.

fs [**-w** file] [[**-p**] table_entry ...]
Print the file system information table.

gdp [**-e**] [**-f**] [**-w** file] [[**-p**] table_entry ...]
Print the gift descriptor protocol table.

gt Equivalent to

tty -t gt

(See **tty** function below.)

help [**-w** file] function ...
Print a description of the named function, including syntax and aliases.

inode [**-e**] [**-f**] [**-w** file] [[**-p**] table_entry ...]
Alias: **i**.
Print the inode table, including file system switch information.

iop16 Equivalent to

tty -t iop16

(See **tty** function below.)

kfp [**-w** file] [**-s** process] [**-r**]

or

kfp [**-w** file] [**-s** process] [value]
Print the frame pointer for the start of a kernel stack trace. The kfp

value can be set using the value argument or the reset option (-r), which sets the kfp from the saved kfp in a dump. If no argument is entered, the current value of the kfp is printed.

lck [**-e**] [**-w** file] [[**-p**] table_entry ...]
Alias: **l**.
Print record locking information. If the **-e** option is used or table address arguments are given, the record lock list is printed. If no argument is entered, information on locks relative to inodes is printed.

linkblk [**-e**] [**-w** file] [[**-p**] table_entry ...]
Print the linkblk table.

major [**-w** file] [entry ...]
Print the MAJOR table.

map [**-w** file] mapname ...
Print the map structure of the given mapname.

mbfree [**-w** file]
Print free streams message block headers.

mblock [**-e**] [**-w** filename] [[**-p**] table_entry ...]
Print allocated streams message block headers.

mode [-w file] [mode]
Set address translation of arguments to virtual (v) or physical (p) mode. If no mode argument is given, the current mode is printed. At the start of a *crash* session, the mode is virtual.

mount [**-e**] [**-w** file] [[**-p**] table_entry ...]
Alias: **m**.
Print the mount table.

msg [**-e**] [**-f**] [**-p**] [**-w** file] [-s process]
[table_entry ...]
Display IPC message queue headers.

msginfo [**-p**] [**-w** file]
Display IPC message information.

msgtext [**-e**] [**-p**] [**-w** file] [-s process]
[table_entry ...]
Display IPC message data.

nm [**-w** file] symbol ...
Print value and type for the given symbol.

notify [**-e**] [**-p**] [**-w** file] symbols

od [**-p**] [**-w** file] [-format] [-mode] [**-s** process]
start_addr [count]

Alias: **rd**.

Print *count* values starting at the start address in one of the following formats: character (**-c**), decimal (**-d**), hexadecimal (**-x**), octal (**-o**), ASCII (**-a**), or hexadecimal/character (**-h**), and one of the following modes: long (**-l**), short (**-t**), or byte (**-b**). The default mode for character and ASCII formats is byte; the default mode for decimal, hexadecimal, and octal formats is long. The format **-h** prints both hexadecimal and character representations of the addresses dumped; no mode needs to be specified. When format or mode is omitted, the previous value is used. At the start of a *crash* session, the format is hexadecimal and the mode is long. If no count is entered, 1 is assumed.

pdt [**-e**] [**-w** file] [**-s** process] section segment

or

pdt [**-e**] [**-w** file] [**-s** process] [**-p**] start_addr [count]

S/640 Only:

The page descriptor table of the designated memory *section* and *segment* is printed. Alternatively, the page descriptor table starting at the start address for *count* entries is printed. If no count is entered, 1 is assumed.

pfdat [**-e**] [**-w** file] [[**-p**] table_entry ...]

Print the pfdata table.

pfree [**-e**] [**-p**] [**-w** file] table_entry ...

Display free list entries.

phash [**-e**] [**-p**] [**-w** file]

Display page hash table.

proc [**-e**] [**-f**] [**-w** file] [[**-p**] table_entry ... #procid ...]

or

proc [**-f**] [**-w** file] [**-r**]

Alias: **p**.

Print the process table. Process table information may be specified in two ways. First, any mixture of table entries and process ids may be entered. Each process id must be preceded by a #. Alternatively, process table information for runnable processes may be specified with the runnable option (**-r**).

pt Equivalent to

tty -t pt

(See **tty** function below.)

qrun [**-w** file]
Print the list of scheduled streams queues.

queue [**-e**] [**-w** file] [[**-p**] table_entry ...]
Print streams queues.

quit Alias: **q**.
Terminate the *crash* session.

rcvd [**-e**] [**-f**] [**-w** file] [[**-p**] table_entry ...]
Print the receive descriptor table.

redirect [**-w** file] [**-c**]

or

redirect [**-w** file] [file]
Used with a file name, redirects output of a *crash* session to the named file. If no argument is given, the file name to which output is being redirected is printed. Alternatively, the close option (**-c**) closes the previously set file and redirects output to the standard output.

region [**-e**] [**-f**] [**-w** file] [[**-p**] table_entry ...]
Print the region table.

scsi [**-w** file]
Display SCSI tables.

scsirqb [**-f**] [**-w** file] [tbl_entry | start_addr]
Display SCSI request blocks.

sdt [**-e**] [**-w** file] [**-s** process] section

or

sdt [**-e**] [**-w** file] [**-s** process] [**-p**] start_addr [count]
S/640 Only:
The segment descriptor table for the named memory section is printed. Alternatively, the segment descriptor table starting at start address for *count* entries is printed. If no count is given, a count of 1 is assumed.

search [**-p**] [**-w** file] [**-m** mask] [-s process] pattern start_addr length

Print the words in memory that match *pattern*, beginning at the start address for *length* words. The mask is anded (&) with each memory word and the result compared against the pattern. The mask defaults to 0xffffffff.

ser Equivalent to

tty -t ser

(See **tty** function below.)

shm [**-e**] [**-f**] [**-p**] [**-w** file] table_entry ...

Display IPC shared memory headers.

shminfo [**-p**] [**-w** file]

Display system IPC shared memory information.

size [**-w** file] [**-x**] [structure_name ...]

Print the size of the designated structure. The (**-x**) option prints the size in hexadecimal. If no argument is given, a list of the structure names for which sizes are available is printed.

sndd [**-e**] [**-f**] [**-w** file] [[**-p**] table_entry ...]

Print the send descriptor table.

sptb [**-e**] [**-p**] [**-w** file] [start_addr]

Display sptballoc maps.

srmount [**-e**] [**-w** file] [[**-p**] table_entry ...]

Print the server mount table.

stack [**-w** file] [**-u**] [process]

or

stack [**-w** file] [**-k**] [process]

or

stack [**-w** file] [[**-p**] **-i** start_addr]

Alias: **s**.

Dump stack. The (**-u**) option prints the user stack. The (**-k**) option prints the kernel stack. The (**-i**) option prints the interrupt stack starting at the start address. If no arguments are entered, the kernel stack for the current process is printed. The interrupt stack and the stack for the current process are not available on a running system.

stat [**-w** file]
Print system statistics.

stream [**-e**] [**-f**] [**-w** file] [[**-p**] table_entry ...]
Print the streams table.

strstat [**-w** file]
Print streams statistics.

swap Display swap map statistics.

swapinfo
Display swap statistics.

trace [**-w** file] [**-r**] [process]

or

trace [**-w** file] [[**-p**] **-i** start_addr]
Alias: **t**.
Print stack trace. The kfp value is used with the **-r** option. The interrupt option prints a trace of the interrupt stack beginning at the start address. The interrupt stack trace and the stack trace for the current process are not available on a running system.

ts [-w file] virtual_address ...
Print closest text symbol to the designated address.

tty [-e] [**-f**] [-w file] [**-t** type [[**-p**] table_entry ...]]

or

tty [-e] [**-f**] [-w file] [[**-p**] start_addr]
Valid types: **ser**, **pt**, **gt**, **vt**, **wxt**, **iop16**.
Print the tty table. If no arguments are given, the tty table for all tty types is printed. If the **-t** option is used, the table for the single tty type specified is printed. If no argument follows the type option, all entries in the table are printed. A single tty entry may be specified from the start address.

unnotify [**-e**] [**-p**] [**-w** file] [**-s** process] symbols
Display queued notifications for process.

user [**-f**] [-w file] [process]
Alias: **u**.
Print the ublock for the designated process.

var [**-w** file]
Alias: **v**.
Print the tunable system parameters.

vt Equivalent to

tty -t vt

(See **tty** function above.)

vtop [**-w** file] [**-s** process] start_addr ...
Print the physical address translation of the virtual start address.

wxt Equivalent to

tty -t wxt

(See **tty** function below.)

FILES

/dev/kmem	system image of currently running system
/dev/rdsk/c?d?s0	used to access system image on disk

NAME

hinv - hardware inventory

SYNOPSIS

/etc/hinv option

/etc/hinv hardware-item

DESCRIPTION

The *hinv* command provides hardware configuration information. There are two forms of the command: in the first form, an *option* is given and the result is printed on *stdout*; in the second form, a particular hardware item is specified, and *hinv* exits with **0** if it exists, or with **1** otherwise.

Option is one of the following:

-p	Print hardware configuration. Items are printed one per line.
-c	Print CPU type.
-f	Print FPU type.
-s	Print system type.
-u	Returns a meaningless value of 128; included for compatibility only.
-m	Print total physical memory in bytes.

Hardware-item is one of the following:

68881	68881 floating-point processor.
iop	Terminal accelerator board.
422	Any RS-422 cluster board.
422-2	Two-channel RS-422 cluster board.
422-4	Four-channel RS-422 cluster board.
vme	VME interface board.
s*n*	RS-232 board *n*.
scsi	A SCSI interface is present.
S0	On-board SCSI is present.
S*n*	SCSI Combo board *n*.
ipt	Interphase tape controller is in EEPROM.
smd	Interphase SMD controller is in EEPROM.

mpcc	Multiprotocol Communications Controller is in EEPROM.
serial	Gives number of serial ports present.
disks	Gives number of disks present.
eeprom	VME EEPROM valid for UNIX.
enet	Ethernet Combo Board is present or a CMC Ethernet board is in EEPROM.
cmcenp	CMC Ethernet board is in EEPROM.
E*n*	Ethernet Combo board *n*.
I*n*	IOP16 board *n* is present (n = 1 .. 4).

BUGS

The *hinv* command does not know about VME cards.

NAME

iopdump - upload a Front-end I/O Processor's RAM

SYOPNSIS

/usr/local/bin/iopdump [-p] [-i iop16number] address length

DESCRIPTION

Iopdump uploads and displays **length** number of memory data bytes beginning at **address** from a Front-end I/O Processor.

The **address** argument is a hexadecimal value.

The **length** argument is a decimal value.

The default Front-end processor is an IOP.

The **-i** option specifies that the type of Front-end I/O Processor is an IOP16. For the **-i** option, the number **iopnumber** must be a decimal number in the range 0 to 3. There is no default number.

The **-p** option causes the retrieved data to be printed as an ascii hexadecimal dump to the standard output. Without this option the binary data is sent to the standard output.

BUGS

For the IOP16, *iopdump* obtains only the data from the first board.

NAME

serstat - display serial port error statistics

SYOPNSIS

serstat

DESCRIPTION

The *serstat* command reports error status information about groups of serial tty ports. The command supports IOP16 ports. The command does not currently support IOP and RIOP ports. When first invoked, *serstat* finds the four ports with the largest number of total errors logged and displays the logged errors.

The command then runs in ''automatic'' mode, in which it scans all serial ports for any change of status. As port status changes, *serstat* updates the display to ensure that the four ports with the largest number of errors logged are displayed at all times. Ports with fewer errors logged are replaced as other ports with more errors logged are displayed. A message at the bottom of the screen indicates which port has most recently changed.

The *serstat* program can also be run in ''scan'' mode and ''continuous'' mode. In scan mode, *serstat* scans sequential groups of ports every three seconds and displays the errors. In continuous mode, *serstat* continues to scan and update the currently-displayed ports only.

To exit *serstat*, generate a keyboard interrupt.

Once *serstat* is running, use any of the following one-character commands:

r	Redraw the screen. No mode change.
a	Redraw the screen. Start automatic mode.
m	Redraw the screen with ports having the most errors. Start automatic mode.
s	Redraw the screen. Start scan mode.
c	Redraw the screen. Start continuous mode.

The program displays data from the following status structure maintained by the serial driver in the kernel:

```
struct sererrstat {
        uint    se_ttyhog;      /* tty input hog status achieved (ttin) */
        uint    se_iflushed;    /* hogs input queues discarded (ttin) */
        uint    se_idropped;    /* input char(s) dropped (ttin, serrint) */
        uint    se_norbuf;      /* no receive buffer available (serrint) */
        uint    se_othrottle;   /* output throttled, low clists (T_HIWATER) */
        uint    se_oflushed;    /* hogs output queue discarded (ttxput) */
        uint     se_odropped;    /* output char(s) dropped (serxsend, sersend) */
        uint     se_notbuf;      /* no transmit buffer available (ttout) */
        uint     se_rxorun;      /* receiver overrun (serrint) */
        uint     se_exstat;      /* external status change (sertint) */
        uint     se_pe;          /* parity errors (serrint) */
        uint     se_frame;       /* CRC/framing error (serrint) */
};
```

All fields are incremented once per event occurrence except **se_idropped** and **se_odropped**. These two fields try to keep track of the number of characters dropped for that particular error event instead of the number of times that error event occurred. Note that 256 characters or more can be lost when the input queue is flushed, but the only record of this event is a single increment to the **se_iflushed** field.

The field **se_exstat** counts the number of external status changes occurring on a port. A break condition or change in the Carrier Detect or Clear To Send lines increments this number. These are not normally error conditions, but may be of interest.

SEE ALSO

termio(7).

NOTE

This utility is intended for diagnostic use by qualified system administrators; it is not a basic user command.

BUGS

Scan mode displays ports which might not be present in the system.

If run on a system without an IOP or IOP16, **serstat** reports a read error and terminates.

NAME

gettimeofday, settimeofday - get/set date and time

SYNOPSIS

```
#include <sys/time.h>

int gettimeofday(tp, tzp)
struct timeval *tp;
struct timezone *tzp;

int settimeofday(tp, tzp)
struct timeval *tp;
struct timezone *tzp;
```

DESCRIPTION

The system's notion of the current Greenwich time and the current time zone is obtained with the *gettimeofday* call, and set with the *settimeofday* call. The time is expressed in seconds and microseconds since midnight (0 hour), January 1, 1970 (GMT). The resolution of the system clock is hardware dependent, and the time may be updated continuously or in ''ticks.'' If *tzp* is a NULL pointer, the time zone information will not be returned or set.

The structures pointed to by *tp* and *tzp* are defined in <*sys/time.h*> as:

```
struct timeval {
        long  tv_sec;      /* seconds since Jan. 1, 1970 */
        long  tv_usec;     /* and microseconds */
};

struct timezone {
        int   tz_minuteswest; /* of Greenwich */
        int   tz_dsttime;  /* type of dst correction to apply */
};
```

The *timezone* structure indicates the local time zone (measured in minutes of time westward from Greenwich), and a flag that, if nonzero, indicates that Daylight Savings Time applies locally during the appropriate part of the year.

Only the super-user can set the time of day or time zone.

Note that you must link the sockets library to your program. Use **-lsocket** in the compile command line.

SEE ALSO

date(1), adjtime(2), ctime(3C).

RETURN VALUE

A 0 return value indicates that the call succeeded. A -1 return value indicates an erro occurred, and in this case an error code is stored into the global variable *errno*.

ERRORS

The following error codes may be set in *errno*:

[EFAULT]	An argument address referenced invalid memory.
[EPERM]	A user other than the super-user attempted to set the time.

NAME

notify, unnotify, evwait, evnowait - manage notifications

SYNOPSIS

```
#include <notify.h>

int notify(type, arg, tag)
ushort type;
char *arg;
char *tag;

int unnotify(type, arg)
ushort type;
char *arg;

ushort evwait(tag, datum)
char **tag;
char **datum;

ushort evnowait(tag, datum)
char **tag;
char **datum;
```

DESCRIPTION

The *notify* system call interface allows a user process to record a number of events that it is interested in, and then waits for any one of them. Like *select*(2), it does synchronous I/O multiplexing, but *notify* waits for a wider range of events and thus has greater functionality than *select*.

The *notify* call requests a notification or set of notifications.

The *unnotify* call retracts an earlier request (or set of requests) for notification.

The *evwait* call waits for a notification to be posted to the calling process.

The *evnowait* call returns the first notification if one exists, returning immediately otherwise.

Notifications are posted FIFO (first-in, first-out) in the user process, each *evwait* returning the first notification or blocking until one is posted. When a *notify* call is given the user must supply the *type* of notification, a *tag*, and an *argument*. The *tag* is an arbitrary number the size of a (**char ***), which is returned by any *evwait* call triggered by that notification request. The *argument* is type specific and is described below.

The return values of *evwait* and *evnowait* are the *type* of the notification.

It is an error for *notify* to be called with a *type* and *arg* matching a currently active notification.

The *notify* calls support the following *types*:

N_FDREAD

Queue a notification if the file descriptor *arg* is readable at the time of the *notify* call, and subsequently whenever there is data to be read. A notification is also queued at end-of-file or when the number of writers on a pipe goes to zero. The datum returned from an *evwait* is a count of the number of bytes available to be read. At EOF the datum is -1, and the request is deleted. This type is implemented for sockets, pipes, ttys, and streams.

N_FDWRITE

Queue a notification if the file descriptor *arg* is writable at the time of the *notify* call, and subsequently when the file goes from a non-writable to a writable state (that is, output is not blocked). *Datum* is the number of characters writable. This type is implemented for sockets, pipes, and streams.

N_SIGNAL

Queue a notification on receipt of a signal. This is used in conjunction with regular signal catching [see signal(2)]. When signal notification is in effect, all caught signals queue notifications instead of causing pseudo-interrupts. If multiple instances of a caught signal occur before the process has received the notification, the returned type is N_LOSTSIG rather than N_SIGNAL. Ignored or defaulted signals are handled normally. Signals are not reset upon notification.

Note that only one call to *notify*

notify(N_SIGNAL,ignored,tag)

is required to enable notification of all signals that have a signal catching function (use a null function). *Evwait* and *evnowait* return the *tag* and *datum*. *Datum* is a bitwise OR of all queued signals: that is, low-numbered signals are represented as low-order bits (signal *n* sets 2^{n-1}).

N_UMSGREAD, N_UMSGWRITE

Queue a notification if the message queue described by *arg* is or becomes readable or writable, respectively. The datum returned is the number of messages received or the number of characters that can be sent, respectively. When the message queue is removed, *datum* is -1, and the request is deleted.

N_INDIR

If *type* is N_INDIR, *arg* is acually a pointer to an array of the following structure (defined in **/usr/include/notify.h**):

```
struct n_request {
     ushort type;
     char *arg;
     char *tag;
}
```

The array should be terminated with an entry having *type* **N_INDIR**. The entire set of notifications is either placed or removed. **N_INDIR** is never returned by *evwait* or *evnowait.*

N_QUERY

Type N_QUERY is valid only as an argument to the *notify* call. *arg* is a pointer to an array of **struct n_indir**, and *tag* is a pointer to an **int** containing the number of elements in the array.

On return, the array contains the current active notifications in a form suitable for passing to *notify* or *unnotify* (that is, terminated by **N_INDIR**), and the **int** pointed to by *tag* contains the number of active notifications (even if there was not enough space to copy them all back).

N_SEMOP

Queue a notification if the semaphore described by the **struct n_semop** (below) pointed to by *arg* would not block, is released, or is removed. *Datum* is **semval** unless the semaphore has been removed, in which case it is -1.

```
struct n_semop {
          int semid;     /* semaphore ID */
          short sem_num; /* semaphore number */
          short sem_op;  /* semaphore operation */
}
```

SEE ALSO

fcntl(2), msgop(2), pipe(2), read(2), select(2), signal(2), socket(2), wait(2), termio(7).

DIAGNOSTICS

All calls return -1 on error, setting *errno* to one of the following:

[EINVAL]	Invalid type was given
[EINVAL]	Caller never did a *notify* (*unnotify, evwait, evnowait*)
[EINVAL]	File is not of a valid type (N_FDREAD, N_FDWRITE).
[EBADF]	File is not open (N_FDREAD, N_FDWRITE)
[EBADF]	Invalid message queue descriptor (N_UMSG)
[ENOSPC]	No space available to allocate notification queue header
[ENOSPC]	No space available to allocate table entry for this notification
[ENOSPC]	Too many active notification requests for given space (N_QUERY)
[EFAULT]	An address fault was generated by a user-supplied pointer

EXAMPLE

```
#include "sys/types.h"
#include <sys/notify.h>
#include <stdio.h>
#include <signal.h>

int sig_catch();

main()
{
	int tag, datum, i;
	char buf[BUFSIZ];
	ushort rv, evwait();

	setbuf(stdout, NULL);
	if (notify(N_FDREAD, 0, 't') < 0)
		perror("notify for N_FDREAD of stdin failed"), exit(1);

	if (notify(N_SIGNAL, 2, 's') < 0)
		perror("notify failed"), exit(1);

	for (i=0; i<20; i++)
		signal(i, sig_catch);
```

```
            for(;;) {
                /* Wait for an event */
                rv = evwait(&tag, &datum);

                /* Tell the user about it */
                printf("0v: %d tag: %d datum: %d0, rv, tag, datum);

                switch (tag) {
                case 's':
                    break;
                case 't':
                    /* Read the input */
                    gets(buf);
                    printf("read '%s'0, buf);
                    if (*buf == 'q')
                    exit(0);
                    break;
                }
            }
        }
        sig_catch()
        {
        }
```

WARNING

The *notify* system call interface is not portable, has little likelihood of becoming so, and may disappear in future releases of CTIX. It is therefore recommended that you use the *poll*(2) system call, and that existing software using *notify* be changed to use *poll* .

NAME

monitor - prepare execution profile

SYNOPSIS

#include <mon.h>

void monitor (lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc)( ), (*highpc)( );
WORD *buffer;
int bufsize, nfunc;

DESCRIPTION

An executable program created by **cc -p** automatically includes calls for *monitor* with default parameters; *monitor* need not be called explicitly.

monitor is an interface to *profil*(2). *lowpc* and *highpc* are the addresses of two functions; *buffer* is the address of a (user supplied) array of *bufsize* WORDs (defined in the <*mon.h*> header file). *monitor* arranges to record a histogram of periodically sampled values of the program counter, and of counts of calls of certain functions, in the buffer. The lowest address sampled is that of *lowpc* and the highest is just below *highpc*. *lowpc* may not equal 0 for this use of *monitor*. At most *nfunc* call counts can be kept; only calls of functions compiled with the profiling option **-p** of *cc*(1) are recorded.

prof(1) can then be used to examine the results.

The name of the file written by *monitor* is controlled by the environment variable PROFDIR. If PROFDIR does not exist, mon.out is created in the current directory. If PROFDIR exists but has no value, *monitor* does not do any profiling and creates no output file. Otherwise, the value of PROFDIR is used as the name of the directory in which to create the output file. If PROFDIR is *dirname*, then the file written is *dirname/pid*.mon.out, where *pid* is the program's process ID. (When *monitor* is called automatically by compiling via **cc -p**, the file created is *dirname/pid.progname*, where *progname* is the name of the program.)

The following discussion is a sketch of *monitor* usage.

For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no less than one half of the range of locations sampled.

To profile the entire program, put the following at the start of **main()** :

```
extern etext;
...
monitor ((int (*)())2, &etext, buf, bufsize, nfunc);
```

etext lies just above all the program text; see *end*(3C).

To stop execution monitoring and write the results, put the following at the end of **main()** :

```
monitor ((int (*)())0, 0, 0, 0, 0);
```

Do not compile with the **-p** option. Run the program and use *prof(1)* to view the results in the output file **mon.out** .

FILES

mon.out

SEE ALSO

cc(1), prof(1), profil(2), end(3C).

BUGS

The ``*dirname/pid*.mon.out'' form does not work; the ``*dirname/pid.progname*'' form (automatically called via **cc -p**) does work.

NAME

sleep - suspend execution for interval

SYNOPSIS

unsigned sleep (seconds)
unsigned seconds;

DESCRIPTION

The current process is suspended from execution for the number of *seconds* specified by the argument. The actual suspension time may be less than that requested for two reasons: (1) Because scheduled wakeups occur at fixed one-second intervals, (on the second, according to an internal clock) and (2) because any caught signal terminates the *sleep* following execution of that signal's catching routine. Also, the suspension time may be longer than requested by an arbitrary amount due to the scheduling of other activity in the system. The value returned by *sleep* will be the "unslept" amount (the requested time minus the time actually slept) in case the caller had an alarm set to go off earlier than the end of the requested *sleep* time, or premature arousal due to another caught signal.

The routine is implemented by setting an alarm signal and pausing until it (or some other signal) occurs. The previous state of the alarm signal is saved and restored. The calling program may have set up an alarm signal before calling *sleep*. If the *sleep* time exceeds the time till such alarm signal, the process sleeps only until the alarm signal would have occurred. The caller's alarm catch routine is executed just before the *sleep* routine returns. But if the *sleep* time is less than the time till such alarm, the prior alarm time is reset to go off at the same time it would have without the intervening *sleep*.

SEE ALSO

alarm(2), pause(2), signal(2).

WARNING

Sleep uses *signal(2)* , not *sigset(2)* , to reset the caller's **SIGALRM** handler routine. Therefore the signal action is reset to its default action on execution of the **SIGALRM** handler. This is probably not what the programmer intended if *sigset(2)* had originally been used to set the signal action.

Sleep uses a longjmp which returns to the *sleep* context when the *alarm(2)* signal handler routine is executed. This may cause premature preemption and loss of context from other nested signal handler routines.