CTIX™ OPERATING SYSTEM MANUAL

Version C
Volume 2

This manual was prepared on a Convergent Technologies S/640 Computer System and was printed on an Apple LaserWriter II Laser Printer.

Second Edition (November 1989) 09-02263-01
Update Notice 1 (November 1990) 09-02578

TABLE OF CONTENTS: VOLUME 2

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 current Greenwich time and the current time zone are 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 follows:

```
struct timeval {
        long  tv_sec;     /* seconds since Jan. 1, 1970 */
        long  tv_usec;    /* and microseconds (always 0 on CTIX) */
};

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 superuser 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 error 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 superuser 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 *type*s:

N_FDREAD

Queues 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 the end-of-file (EOF) 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

Queues 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

Queues 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

Queues 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

Queues a notification if the semaphore described by the **struct n_semop** 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

shmop - shared memory operations

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

char *shmat (shmid, shmaddr, shmflg)
int shmid;
char *shmaddr;
int shmflg;

int shmdt (shmaddr)
char *shmaddr;
```

DESCRIPTION

shmat attaches the shared memory segment associated with the shared memory identifier specified by *shmid* to the data segment of the calling process. The segment is attached at the address specified by one of the following criteria:

- If *shmaddr* is equal to zero, the segment is attached at the first available address as selected by the system.
- If *shmaddr* is not equal to zero and (*shmflg* & **SHM_RND**) is "true", the segment is attached at the address given by [*shmaddr* - (*shmaddr* modulus **SHMLBA**)].
- If *shmaddr* is not equal to zero and (*shmflg* & **SHM_RND**) is "false", the segment is attached at the address given by *shmaddr*.

shmdt detaches from the calling process's data segment the shared memory segment located at the address specified by *shmaddr*.

The segment is attached for reading if (*shmflg* & **SHM_RDONLY**) is "true" {READ}, otherwise it is attached for reading and writing {READ/WRITE}.

shmat will fail and not attach the shared memory segment if one or more of the following are true:

[EINVAL]	*shmid* is not a valid shared memory identifier.
[EACCES]	Operation permission is denied to the calling process [see *intro*(2)].
[ENOMEM]	The available data space is not large enough to accommodate the shared memory segment.

[EINVAL] *shmaddr* is not equal to zero, and the value of [*shmaddr* - (*shmaddr* modulus **SHMLBA**)] is an illegal address.

[EINVAL] *shmaddr* is not equal to zero, (*shmflg* & **SHM_RND**) is "false", and the value of *shmaddr* is an illegal address.

[EMFILE] The number of shared memory segments attached to the calling process would exceed the system-imposed limit.

[EINVAL] *shmdt* will fail and not detach the shared memory segment if *shmaddr* is not the data segment start address of a shared memory segment.

SEE ALSO

exec(2), exit(2), fork(2), intro(2), shmctl(2), shmget(2).

DIAGNOSTICS

Upon successful completion, the return value is as follows:

- *shmat* returns the data segment start address of the attached shared memory segment.
- *shmdt* returns a value of 0.

Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

NOTES

The user must explicitly remove shared memory segments after the last reference to them has been removed.

NAME

syslocal - special system requests

SYNOPSIS

#include <syslocal.h>
int syslocal (cmd [, arg] ...)
int cmd;

DESCRIPTION

The *syslocal* routine executes certain special system calls. The specific call is indicated by the first argument. See the **<sys/syslocal.h>** file for complete documentation of parameters.

System Type

int syslocal(SYSL_SYSTEM)

Returns SYSL_MITI for S/Series.

Family Member

int syslocal(SYSL_FAMILYMEMBER)

Returns a value identifying the specific system: SYSLFMITI1 for S/120, S/221-2, or S/320; SYSLFMITI2 for S/480 or S/640; SYSLFS80 for S/80; SYSLFS280 for S/280; and SYSLFS4000 for S/4040.

Superblock Resynchronization

int syslocal(SYSL_RESYNC, devnum)
short devnum

Rereads contents of superblock from disk. *devnum* specifies the file system: The high-order byte contains the major device number of the character special device; the low-order byte contains the minor device number. Only the superuser can reread the contents of the superblock from disk.

Maximum Number of Users

int syslocal(SYSL_MAXUSERS)

Returns maximum number of users configured for this system.

Kernel Addresses

syslocal(SYSL_KADDR, arg)

Returns certain addresses of kernel data structures. This allows certain programs (*ps*, *killall*) to run properly, even if **/unix** is not the currently running operating system.

arg is one of the following:

SLA_V	Returns address of var structure (sys/var.h).
SLA_PROC	Returns address of proc structure (sys/proc.h).
SLA_ERR	Returns address of err structure (sys/err.h).
SCA_TIME	Returns address of int time.
SLA_CDT	Returns address of crash dump table (CDT) = (sys/hardware.h).
SLA_GDUTAB	Returns address of gdutab (sys/iobuf.h).
SLA_USRSTK	Returns highest address of user stack.
SLA_USIGN	Returns signature of running UNIX (may be compared with **/unix** to see if they are identical).
SLA_MEM	Returns number of bytes of physical memory.
SLA_BDEVCNT	Returns number of slots in struct bdevsw (sys/conf.h).
SLA_CDEVCNT	Returns number of slots in struct cdevsw (sys/conf.h).
SLA_PRELD	Returns the address of the preloaded driver table.

Object Module Type

syslocal(SYSL_0413MAGIC)

Returns 1 if the kernel can support the **-F** option of *ld*().

Read Real-Time Clock

syslocal(SYSL_RDRTC, arg)

Reads current state of real-time (battery supported) clock. *arg* is a pointer to struct **rtc** (**sys/rtc.h**)

Write Real-Time Clock

syslocal(SYSL_WTRTC, arg)

Writes new state of real-time clock. *arg* is a pointer to a struct **rtc** (**sys/rtc.h**). EIO is returned if any of the values are illegal. Only the superuser can write the real-time clock.

Reboot System

syslocal(SYSL_REBOOT)

Forces a software reset. Only the superuser may reset. Obsolete: retained for compatibility. Use *uadmin*(2) instead.

Allocate or Bind a Loadable Driver

syslocal(SYSL_ALLOCDRV, option, arg)

syslocal(SYSL_BINDDRV, option, arg)

These two functions implement the loadable driver functions of CTIX. They both require superuser privilege.

Loading drivers consists of two phases: allocation of virtual space, device numbers, and device IDs; and binding. Fully relocating a driver into memory, allocating physical space, plugging the device switch tables, calling initialization routines, and unloading require the same two phases in reverse. For information on the arguments, see **/usr/include/sys/drv.h**.

Determine Processor Type

syslocal(SYSL_PROCESSOR)

Returns a value that can be used to determine what kind of processor (68010, 68020, or 68040) is running and whether floating-point hardware (68881/68040) is available.

Enable Fixed Priority Range

syslocal(SYSL_RTNICE,flag)

Enables/disables the fixed priority range [see *nice*(2)]. *flag* is 1 for enable, 2 for disable. Only the superuser can execute this call, which affects every process.

S/80, S/280, S/480, and S/640 Hardware Configuration

syslocal(SYSL_MITICFIG)

Returns a bit mask of the hardware that is present. Values can be found in **syslocal.h.** A more convenient way to get this information is by using *hinv*(1M).

S/4000 Hardware Configuration

syslocal(SYSL_SXCFIG, arg1, arg2)

Returns a bit mask of the hardware that is present. Values can be found in **syslocal.h.** A more convenient way to get this information is by using *hinv*(1M).

arg1 is one of the following:

SLA_SER	Returns a bit mask of the serial boards present.
SLA_IEN	Returns a bit mask of the Ethernet boards present.
SLA_SCSI	Returns a bit mask of the SCSI boards present.
SLA_SERNPORT	Returns number of serial ports for board denoted by *arg2*.

SLA_IENPORT	Returns number of Ethernet ports for board denoted by *arg2*.
SLA_SCSINPORT	Returns number of SCSI ports for board denoted by *arg2*.

SEE ALSO

fsck(1M), nice(2).

DIAGNOSTICS

Note that *syslocal* fails if one of the following is true:

[EINVAL]	*cmd* or any suboption is illegal.
[EFAULT]	An *arg* points outside the process's space.