NAME

seek − move read/write pointer

SYNOPSIS

seek (fildes, offset, ptrname)

DESCRIPTION

Seek has been droped in this version of the library. Use *lseek*(2) instead.

ASSEMBLER

(seek = 19.)
(file descriptor in r0)
sys seek; offset; ptrname

NAME

sema, p, v, test, post, block, setsem, rdsem, lock, unlock, tlock, noulk − semaphore operations

SYNOPSIS

p (sema)
v (sema)
test (sema)
post (sema)
block (sema)
setsem (sema,value)
rdsem (sema)
lock (sema)
unlock (sema)
tlock (sema)
noulk ( )

DESCRIPTION

The indicated function is performed on the specified semaphore. Semaphores are assigned on a system-wide basis. By convention the file **/usr/include/sema.h** contains define symbols for the usage of semaphores. Also by convention, semaphore numbers less than zero are reserved for system programs such as the line printer spooling system.

The various semaphore operations are defined as follows:

post causes all users doing a *block* on the specified semaphore to be awakened. As a side effect, the semaphore is incremented.

block causes the current user to roadblock until a subsequent *post* on the specified semaphore.

p causes the current user to roadblock if the specified semaphore's value is zero until it becomes nonzero. If the semaphore's value is nonzero, the semaphore is decremented and the user is not roadblocked.

v causes the specified semaphore to be incremented.

test causes the specified semaphore to be decremented if the current value is nonzero.

rdsem returns the current value of the specified semaphore.

setsem sets the specified semaphore to the given value.

lock roadblocks if the specified semaphore is nonzero until it becomes zero. Once the semaphore is zero, the negative of the current process' pid is stored in the semaphore and *lock* returns a zero.

unlock sets the specified semaphore to zero if its current value is the negative of the current process' pid; an error is returned otherwise.

tlock is like *lock* except the current process is not roadblocked if the semaphore's value is nonzero. Instead, the value is returned.

noulk prevents the system from automatically unlocking any semaphores which may be locked by the current process at termination time.

In all cases, if the function is successfully performed, the system returns the old semaphore value (in R0).

Users of semaphores should be wary of the interaction between caught signals and the use of *p*, *block*, and *lock*. If a signal is caught while waiting for a semaphore, the call to the semaphore primative will return with a −1 error and the error number of EINTR. The base level routine

should then call the primative again if this is desired.

The use of the various semaphore primatives in an intermixed manner may produce undefined results. In particular a single semaphore should be used with only one of the following tuples: *block-post*, *p-v-test*, or *lock-unlock-tlock*. *Rdsem* may be used on any semaphore at any time. *Setsem* should only be used to set a semaphore to an inital value.

The system will only automatically *unlock* those semaphores which have been *locked* or *tlocked*.

A counting semaphore (*p-v-test*) may only assume values between zero and 32767. If an overflow occurs the new value is set to one.

SEE ALSO

sema(1)

DIAGNOSTICS

From C, a −1 value indicates an error.

ASSEMBLER

(semas = 63.; not in assembler)
(new value in R0)
sys semas; func; sema
(old value in R0)

NAME

setgid − set process group ID

SYNOPSIS

setgid (gid)

DESCRIPTION

The group ID of the current process is set to the argument, *gid*. Both the effective and the real group ID are set. This call is only permitted to the super-user, unless the argument is the real group ID.

SEE ALSO

getgid(2)

DIAGNOSTICS

Error bit (c-bit) is set as indicated; from C, a −1 value indicates an error.

ASSEMBLER

(setgid = 46.)
(group ID in r0)
sys setgid

NAME

setpgrp − set process group

SYNOPSIS

setpgrp (newgrp)

DESCRIPTION

The process group of the current process group is set to the argument. The old process group is returned. If the argument is **0**, the current process group is not changed but simply returned. If the argument is **−1**, the current process is dissassociated from any process group (i.e., given a NULL (0) process group).

SEE ALSO

setpgrp(1)

ASSEMBLER

(setpgrp = 39.)
(not in assembler)
(new process group in r0)
sys setpgrp
(old process group in r0)

NAME

setuid − set process user ID

SYNOPSIS

setuid (uid)

DESCRIPTION

The user ID of the current process is set to the argument, *uid*. Both the effective and the real user ID are set. This call is only permitted to the super-user, unless the argument is the real user ID.

SEE ALSO

getuid(2)

DIAGNOSTICS

Error bit (c-bit) is set as indicated; from C, a −1 value indicates an error.

ASSEMBLER

(setuid = 23.)
(user ID in r0)
sys setuid

NAME

smcreat, smopen, smclose, smget, smput − shared memory operations

SYNOPSIS

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

smcreat (path, access, size);
char *path;
short access;
long size;

sm_des = smopen (path, mode);
smdes_t sm_des;
char *path;
short mode;

smclose (sm_des);
smdes_t sm_des;

vaddr = smget (sm_des, mode, offset, size, time_lmt);
caddt_t vaddr;
smdes_t sm_des;
short mode;
long offset;
long size;
short time_lmt;

smput (vaddr)
caddt_t vaddr;
```

DESCRIPTION

Shared memory is a form of memory which can be attached to a process's address space, read or written, and then released, with the contents preserved for later or simultaneous attachment by another process. Shared memory can be used as multiple access user memory (MAUS), or as a means of passing large pieces of data from one process to another with only one process accessing the data at a time. Exactly how it is used is determined by the user processes. Each piece of shared memory is referenced orginally via the UNIX file system and can also be accessed as a file, via the normal *open, read, write, lseek,* and *close* system calls.

smcreat Shared memory is created dynamically with the *smcreat* system call. *Smcreat* is analogous to *creat* except that a size in bytes must be specified at creation time. Shared memory retains a fixed size during whatever time it exists as a part of the file system. If *size* is not a multiple of **BSIZE** (See **/usr/include/sys/param.h**) bytes it is rounded up. *Path* is a pointer to a normal UNIX pathname. *Access* specifies in the normal way (See *chmod*(2)) who may *open* or *smopen* a specific piece of shared memory.

If the specified piece of shared memory already exists and no process has it attached and this user has the proper permissions, the old shared memory will be recreated to the new size. All newly created shared memory is initialized to all zeros.

smopen Once a piece of shared memory exists, it can be opened for attachment to a process's address space via the *smopen* system call. *Path* is again a UNIX pathname specifying which piece of shared memory. *Mode* is the normal file system type mode, 0 for read, 1 for write, 2 for read and write permissions. *Write,* actually means *read-write* on most systems, since write permissions on memory imply read permissions. *sm_des* (shared memory descriptor) is returned upon a successful *smopen,* and is used when attaching shared memory via the *smget* system call.

smclose *Smclose* releases a specified *sm_des*. Any sections attached at the time of the *smclose* will remain attached, but once detached, will not be accessable again without a new *smopen*.

smget *Smget* performs the actual attachment of shared memory to the user process's address space. *Sm_des* is a shared memory descriptor previously returned from *sm_creat* or *sm_open*. *Mode* specifies how the user wishes to access the memory once it is attached. There are three modes as defined in **<sys/shmem.h>**:

SMREAD
Attach the memory read only.

SMWRITE
Attach the memory for reading and writing. Note that there is no equivalent of 'write only'.

SMREADLOCK
Attach the memory for reading only, but only when no one else has it attached for writing. If someone else has it attached for writing, wait. Do not allow new requests for writing to succeed. This means that eventually the request will succeed. **SMREADLOCK** guarantees that the data being read is stable.

SMWRITELOCK
Attach the memory for reading and writing, but only when there is no one else who has this section attached for writing. If other people have any portion of this section attached for writing, wait for the time period specified by *time_lmt* while they drain away. Do not allow any new people to attach the requested section for writing so that eventually the *smget* will succeed. Using the **SMWRITELOCK** feature to access shared memory removes the requirement for any outside locking procedures such as semaphores. The user is guaranteed that when *smget* with a mode of **SMWRITELOCK** succeeds, that this is the only process writing that memory at this time. Until an *smput* is done on this section of memory, it will remain so locked.

The following state table shows the interactions between the current state of a piece of shared memory and a new request to have it attached with a particular mode via *smget*. **T** means request will be granted immediately. **F** means requester will have to wait.

Requested State	Current State				
—	r	w	rl	wl	unattached
r	T	T	T	T	T
w	T	T	F	F	T
rl	T	F	T	F	T
wl	T	F	F	F	T

Offset is an unsigned offset from the beginning of a piece of shared memory to which a user wishes to attach. It should be some multiple of **BSIZE** bytes. If it is not, the *smget* will fail. *Size* is the unsigned size of the section of shared memory that the process wishes to attach. It also should be a multiple of **BSIZE** bytes. It may be rounded up by as much as **BSIZE** −1 bytes so that the entire section of shared memory the user requested access to is available. *Smget* will fail if the section of shared memory requested is not within the limits of the piece of shared

memory as it was created by *smcreat*. *Time_lmt* is the amount of time the user is willing to wait until a particular section of shared memory is free, when trying to do an *smget* with a mode of **SMWRITELOCK**. If *time_lmt* is 0, the *smget* will fail immediately if anyone else has any portion of the requested shared memory attached. If *time_lmt* is less than 0, the user is willing to wait indefinately for the section of shared memory to become free. If *time_lmt* is positive, then the user is willing to wait this many seconds for the section of shared memory to become available. If *smget* fails because the *time_lmt* was exceeded, the error **EBUSY** will be returned in *errno* and *vaddr* will be set to **NULL**. Whenever *smget* succeeds, *vaddr* will be some multiple of **BSIZE** bytes and is the pointer to the section of attached shared memory.

If *sm_des* is **SMDESNONE** (As defined in <sys/**shmem.h**>), the *smget* behaves like a memory allocator. An unnamed section of memory, **size** big, is attached to the user process. *Mode* is only meaningful if it is **SMWRITELOCK**. This mode will prevent a child process from inheriting the attached section of memory. In other cases the section of memory is readable and writable. Memory attached in this fashion is guaranteed to be zeroed. *Offset* is ignored. *Time_lmt* behaves in the normal fashion.

smput *Smput* releases an attached section of shared memory. *Vaddr* must match the value returned by *smget*.

A copy of **/usr/include/sys/shmem.h** is included here for reference.

```
/*      "mode" definitions to be used with "smget".                    */

#define                 SMREAD          0
#define                 SMWRITE         1
#define                 SMREADLOCK      2
#define                 SMWRITELOCK     3

/*      Shared memory descriptor to use when attaching                 */
/*      unnamed memory.                                                */

#define                 SMDESNONE   (-1)

typedef     short       smdes_t ;

#ifdef      KERNEL

/*      Structure maintaining the state of each page of                */
/*      shared memory.                                                 */

struct SM_pgstate
 {
        daddr_t sm_block ;          /* Block for this page
                                     * into the file attached
                                     * as shared memory.
                                     * Are stored in ascending
                                     * order for any file.
                                     */
        struct buf *sm_bufpt ;      /* Ptr to buffer header
                                     * for this page.
                                     */
        char sm_refcnt ;            /* Count of total users
                                     * attached to this page.
                                     */
        char sm_rlrefcnt ;          /* Count of users attached to
                                     * this page as read locked.
                                     */
```

```
                char sm_wrefcnt ;           /* Count of users attached
                                             * to this page for writing.
                                             */
                char sm_wlwant ;            /* Number of users wanting page
                                             * write locked.
                                             */
                char sm_rlwant ;            /* Number of users wanting page
                                             * read locked.
                                             */
                char sm_wwant ;             /* Count of users wanting page
                                             * for writing.
                                             */
                short sm_flags ;
                struct SM_pgstate *sm_next ;  /* Pointer to next page's
                                               * control structure.
                                               */
        } ;

        /*      Definitions for "sm_flags".                                  */

        #define         IS_OCCUPIED         1
        #define         IS_WRITELOCKED      2
        #define         IS_READLOCKED       4
        #define         WL_REQUEST          10
        #define         RL_REQUEST          20
        #define         W_REQUEST           40

        struct SM_pgptrs
         {
                struct SM_pgstate *s_current ;
                struct SM_pgstate *s_previous ;
         } ;
        #endif
```

FILES

/dev/shmem/*

SHARED MEMORY RULES

1) Shared memory descriptors are inherited across forks and executes.

2) Sections of attached shared memory are inherited across forks if they were not opened **SMWRITELOCK.** Writelocked sections are retained by the parent, but closed to the child, keeping them writelocked.

3) Attached sections of shared memory are not inherited across *exec's.*

4) If a *break* system call tries to expand memory into an attached section of shared memory, it will fail.

5) If some process tries to *open* a shared memory file while another process is waiting for an *smget* with **SMWRITELOCK** to succeed, the *open* will fail.

SEE ALSO

break(2), close(2), creat(2), open(2)

DIAGNOSTICS

From assembly code, the carry bit is set in the case of errors and *errno* set with an indication of the specific error. From C a −1 is returned from *smcreat, smclose, smopen,* and *smput,* and a **NULL** from *smget.*

IMPLEMENTATION CONSIDERATIONS

It is envisioned that a shared memory file will be a special file type. It will be implemented only under version 7 or later file systems. Each section of shared memory will require two inodes, a visible inode referenced in the UNIX file system, and an invisible inode, used only by

the shared memory routines to access the data when it is on the disk. This implementation will require that *check* understand this new file type and not remove the invisible inode during a check.

It should be noted that the implementation of *smget* and *smput* interact very nicely with the *MSG* implementation proposed by Dale DeJager. When passing large *no_copy* messages, memory must first be allocated and the final receiver must return it to the operating system. *Smget* and *smput* nicely serve the purpose of the routines *memget* and *memfree*.

When new shared memory is created initially, a control block will be allocated at the same time. In the control block will be an *SM_control* structure for each page of this section of shared memory. The structure contains two counts, the count of the total number of users attached to this page, and the number of users attached to this page for writing. It also contains five flags, **IS_WRITELOCKED,** meaning that the page is currently writelocked, **IS_READLOCKED,** meaning that the page is currently readlocked, **WL_REQUEST,** meaning that someone is requesting writelock permissions for this page, **RL_REQUEST,** meaning that someone is requesting readlock permission for this page, and **W_REQUEST,** meaning that someone is requesting write permission for this page. When a process requests a page for readlocked access, each page will be locked starting from the beginning, if that page has a 0 reference count for writers. If the refenerce count is something other than 0, the process will set the **RL_REQUEST** flag and sleep on the page until it is awakened and finds the writing reference count 0. When a process requests a page for writelocked access, each page will be locked starting from the beginning, if that page has a 0 reference count for writers and has the **IS_READLOCKED** and **IS_WRITELOCKED** flags off. If either condition isn't met, the process will sleep on the page until it awakens to find both conditions satisfied. If someone is requesting read access, they always succeed. If someone is requesting write access, they will succeed if the page isn't **IS_WRITELOCKED** or **IS_READLOCKED.** If they can't attach immediately they will set the **W_REQUEST** and sleep on the page until both conditions are satisfied before succeeding. In all cases, if the *time_lmt* expires during the wait for the pages to become available, the request will fail.

NAME

signal − catch or ignore signals

SYNOPSIS

```
#include <signal.h>

int (*signal (sig, func))( )
int sig;
(*func)( );
```

DESCRIPTION

A signal is generated by some abnormal event, initiated either by a user at a typewriter (quit, interrupt), by a program error (bus error, etc.), or by request of another program (kill). Normally, all signals (except death of a child and power fail) cause termination of the receiving process, but a *signal* call allows them either to be ignored or to cause an interrupt to a specified location. Here is the list of signals:

SIGHUP	1	hangup
SIGINT	2	interrupt
SIGQUIT	3*	quit
SIGILL	4*	illegal instruction (not reset when caught)
SIGTRAP	5*	trace trap (not reset when caught)
SIGIOT	6*	IOT instruction
SIGEMT	7*	EMT instruction
SIGFPE	8*	floating point exception
SIGKILL	9	kill (cannot be caught or ignored)
SIGBUS	10*	bus error
SIGSEGV	11*	segmentation violation
SIGSYS	12*	bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	catchable software termination signal
	16	unassigned
	17	unassigned
SIGCLD	18	death of a child
SIGPWR	19	power fail

The starred (*) signals in the list above cause a core image if not caught or ignored.

If *func* is **SIG_DFL**, the default action for signal *sig* is reinstated; this default is termination, sometimes with a core image. If *func* is **SIG_IGN**, the signal is ignored. Otherwise when the signal occurs *func* will be called with the signal number as argument. A return from the function will continue the process at the point it was interrupted. Except as indicated, a signal is reset to **SIG_DFL** after being caught. Thus if it is desired to catch every such signal, the catching routine must issue another *signal* call.

When a caught signal occurs during certain system calls, the call terminates prematurely. In particular this can occur during a *read* or *write*(2) on a slow device (like a typewriter; but not a file); and during *pause* or *wait*(2). When such a signal occurs, the saved user status is arranged in such a way that when return from the signal-catching takes place, it will appear that the system call returned an error status. The user's program may then, if it wishes, re-execute the call.

The value of *signal* is the previous (or initial) value of *func* for the particular signal.

After a *fork*(2) the child inherits all signals. *Exec*(2) resets all caught signals to default action.

Users should not use the signal numbers directly; instead, they should include the file **/usr/include/signal.h** as indicated above.

The default action for the death of a child signal is to ignore the signal. If *label* is odd, the signal is ignored and terminated child processes are automatically removed from the system — eliminating the necessity of doing a *wait*(2) for the terminated children.

For the power fail signal, the default action is to ignore it.

SEE ALSO

kill(1), kill(2), ptrace(2), setjmp(3C)

DIAGNOSTICS

The value **−1** is returned if the given signal is out of range.

BUGS

If a repeated signal arrives before the last one can be reset, there is no chance to catch it.

ASSEMBLER

(signal = 48.)
sys signal; sig; label
(old value in r0)

If *label* is 0, default action is reinstated. If *label* is odd, the signal is ignored. Any other even *label* specifies an address in the process where an interrupt is simulated. An RTI or RTT instruction will return from the interrupt.

NAME

sprofil — turn on/off system profiling

SYNOPSIS

#include <sys/sprof.h>

int sprofil (spcnt, numcnts, lowpc, intsize)
struct SPCNT spcnt;
unsigned int numcnts;
caddr_t lowpc;
unsigned int intsize;

DESCRIPTION

Calling *sprofil* with *spcnt* non-zero will initiate system profiling. If any other process is profiling, **EBUSY** is immediately returned. If *intsize* is 0, then the system will profile system routines, reserving a counter for *numcnts* global external text symbols. The (sorted) starting addresses for the system routines are provided by the user (usually from **/unix**).

If *intsize* is non-zero, the system will reserve a counter for every *intsize* group of bytes, starting at byte address *lowpc*, for a total of *numcnts* intervals.

If the size of the *spcnt* structure would overflow one PDP11/70 memory page (8192 bytes), then **EINVAL** is returned. Otherwise, the user's data space is locked in memory and the memory management information for the *spcnt* structure is saved in the kernel's *sysprof* structure.

If an independent clock is used (either a DEC KW11-K or a Digital Pathways TCU100 may be used), then that clock is started. When it interrupts (should be at level 7), or when the system clock routine is called, if no independent clock is used, the counter for the interrupted routine is incremented by 1. If the system was in user or idle mode, that is recorded instead.

The system increments the proper counter in user D space by temporarily changing kernel D space register 5 to point to the user page with the table of counters (hence the one page limit for the size of the *SPCNT* structure).

System profiling is stopped by sending a 0 in argument one. Normally, a user would do:

```
sprofil (spcnt, numcnts, lowpc, intsize);
sleep (seconds);
sprofil (0, 0, 0, 0);
```

and then report the results in some tabular form (see *sprof*(1M)).

The file **<sys/sprof.h>** including the prototype *SPCNT* structure, is as follows:

```
/*              @(#)sprof.h     3.2             */

/*
 *
 * Used by system profiling routines( sprofil.sincupc and sprof )
 *
 *
 */

#ifdef KERNEL
struct  pgreg {
                char *par;
                char *pdr;
};

struct sysprof {
```

```
                struct          SPCNT           *base;
                caddr_t lowpc;                  /* low pc word for i option */
                unsigned int numcnts;           /* number of counters in union array */
                unsigned int intsize;           /* size of i intervals or 0 for r opt */
                int pid;
                struct pgreg  newpg;
                struct pgreg  oldpg;
};

#endif
struct          NHIT {
                caddr_t         nloc;
                spcnt_t         nhits;
};

struct          SPCNT {
                long            b_urhits;
                long            b_syhits;
                long            b_idhits;
                union           {
                                /*
                                 *              "allocate" maximum possible size of counter buffers
                                 *              (they must fit entirely into one page)
                                 */
                                struct          NHIT            ropt[(8192 - 3*sizeof(long))/sizeof(struct NHIT)];
                                spcnt_t                         iopt[(8192 - 3*sizeof(long))/sizeof(spcnt_t)];
                }               u_ct;
};

#ifdef          IPROFCLK

/* independent profile clock kw11-k (A clock) */

#define         KW11K           (struct kw11ka *)0170404

struct          kw11ka {
                int             kw11ks;
                int             kw11kb;
};
#else
#ifdef          IPROFCLB

/* independent profile clock TCU-100 (battery clock) */

#define TCU100 (int *)0160774
#define TCURATE -48
                /*              rate of -33 should be 62.06/sec, is 120/sec for our clock
                                                -45             45.6    70.6
                                                -64             31      42.6
                                                -48             42.6    64
                                our clock may be dumb, but at least it's consistent
                 */

#endif
#endif
```

Notice that only the kernel gets the *sysprof* definition; the user can use the *SPCNT* structure definition.

SEE ALSO

sprof(1)

WARNINGS

If the data space in the kernel gets too big, the kernel D-space register 5 trick may not work.

If the system clock is used, any system routine in sync with the clock may appear invisible to system profiling.

ASSEMBLER

(syscb = 45. ; sprofil = 4.)
(struct spcnt in r0; a 4 in r1)
sys sprofil; numcnts; lowpc; intsize;

NAME

stat, fstat – get file status

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

stat (name, buf)
char *name;
struct stat *buf;

fstat (fildes, buf)
struct stat *buf;
```

DESCRIPTION

Stat obtains detailed information about a named file. *Fstat* obtains the same information about an open file known by the file descriptor from a successful *open, creat, dup* or *pipe*(2) call.

Name points to a null-terminated string naming a file; *buf* is the address of a buffer into which information is placed concerning the file. It is unnecessary to have any permissions at all with respect to the file, but all directories leading to the file must be readable. The structure pointed to by *buf* has the following structure. The defined types, *ino_t, off_t, time_t,* name various width integer values; *dev_t* encodes major and minor device numbers; their exact definitions are in the include file <sys/types.h> (see *types*(7)).

```
/*      @(#)stat.h      3.1             */
struct  stat
{
        dev_t       st_dev;
        ino_t       st_ino;
        int         st_mode;
        int         st_nlink;
        int         st_uid;
        int         st_gid;
        dev_t       st_rdev;
        off_t       st_size;
        time_t      st_atime;
        time_t      st_mtime;
        time_t      st_ctime;
};

#define     S_IFMT      0170000             /* type of file */
#define                 S_IFDIR     0040000 /* directory */
#define                 S_IFCHR     0020000 /* character special */
#define                 S_IFBLK     0060000 /* block special */
#define                 S_IFREG     0100000 /* regular */
#define                 S_IFMPC     0030000 /* multiplexed char special */
#define                 S_IFMPB     0070000 /* multiplexed block special */
#define     S_ISUID     0004000             /* set user id on execution */
#define     S_ISGID     0002000             /* set group id on execution */
#define     S_ISVTX     0001000             /* save swapped text even after use */
#define     S_IREAD     0000400             /* read permission, owner */
#define     S_IWRITE    0000200             /* write permission, owner */
#define     S_IEXEC     0000100             /* execute/search permission, owner */
```

When *fildes* is associated with a pipe, *fstat* reports an ordinary file with an i-node number, restricted permissions, and a not necessarily meaningful length.

SEE ALSO

ls(1), fs(5), types(7)

DIAGNOSTICS

Zero is returned if a status is available; –1 if the file cannot be found.

ASSEMBLER

(stat = 18.)
sys stat; name; buf

(fstat = 28.)
(file descriptor in r0)
sys fstat; buf

NAME

stat − get file status

SYNOPSIS

```
stat (name, buf)
char *name;
struct inode *buf;
```

DESCRIPTION

Name points to a null-terminated string naming a file; *buf* is the address of a 36(10) byte buffer into which information is placed concerning the file. It is unnecessary to have any permissions at all with respect to the file, but all directories leading to the file must be readable. After *stat*, *buf* has the following structure (starting offset given in bytes):

```
struct {
        char    minor;          /* +0: minor device of i-node */
        char    major;          /* +1: major device */
        int     inumber;        /* +2 */
        int     flags;          /* +4: see below */
        char    nlinks;         /* +6: number of links to file */
        char    uid;            /* +7: user ID of owner */
        char    gid;            /* +8: group ID of owner */
        char    size0;          /* +9: high byte of 24-bit size */
        int     size1;          /* +10: low word of 24-bit size */
        int     addr[8];/* +12: block numbers or device number */
        int     actime[2];      /* +28: time of last access */
        int     modtime[2];     /* +32: time of last modification */
};
```

The flags are as follows:

100000	i-node is allocated
060000	2-bit file type:
	000000 plain file
	040000 directory
	020000 character-type special file
	060000 block-type special file.
010000	large file
004000	set user-ID on execution
002000	set group-ID on execution
000400	read (owner)
000200	write (owner)
000100	execute (owner)
000070	read, write, execute (group)
000007	read, write, execute (others)

SEE ALSO

ls(1), fstat(2), fs(5)

DIAGNOSTICS

Error bit (c-bit) is set if the file cannot be found. From C, a −1 return indicates an error.

NAME

stime − set time

SYNOPSIS

stime (tbuf)
int tbuf[2];

DESCRIPTION

Stime sets the system's idea of the time and date. Time is measured in seconds from 0000 GMT Jan 1 1970. Only the super-user may use this call. Setting the time and date causes any sleeping or paused processes to be awakened (programs using *alarm* and the C interface to *sleep* are not disturbed).

SEE ALSO

date(1), pause(2), sleep(2), time(2), ctime(3C)

DIAGNOSTICS

Error bit (c-bit) set if user is not the super-user. From C, a −1 indicates an error.

ASSEMBLER

(stime = 25.)
(time in r0-r1)
sys stime

NAME

stty, gtty − set and retrieve terminal modes

SYNOPSIS

#include <sys/sgtty.h>

stty (fildes, arg)
struct SGBUF *arg;

gtty (fildes, arg)
struct SGBUF *arg;

DESCRIPTION

Stty and *gtty* are used to set and get various characteristics of a character device referred to by *fildes*. *Fildes* usually refers to a typewriter line but may also refer to certain special devices such as named pipes. The second argument, *arg*, should be a pointer to the SGTTY structure which is defined in the include file **<sys/sgtty.h>** . A copy of this header file is included here for reference:

```
/*              @(#)sgtty.h     3.2             */

/*
 * stty and gtty structure layouts
 *
 * All structures are 6 bytes.
 * For each given command, doing a stty
 * sets the information into the operating
 * system. Doing a gtty retrieves it.
 */


/*
 * Command 0 -- set modes and speeds.
 *              Wait for output to drain and flush any input.
 * Command 1 -- set modes and speeds.
 *              Don't wait or flush.
 */
#define         STTY_MODES  0
#define         STTY_NFMODES                1
struct SGBUF {
                char            sm_ispeed;      /* Input speed */
                char            sm_ospeed;      /* Output speed, data and stop bits */
                char            sm_cmd;                         /* Command = 0 or 1 */
                char            sm_fill;
                int             sm_modes;       /* See below */
};

/*
 * Modes
 */
#define         NCDELAY         0000001                 /* no carriage return delay */
#define         XTABS           0000002                 /* map tabs to spaces on output */
#define         LCASE           0000004                 /* upper case only terminal */
#define         ECHO            0000010                 /* echo all received chars */
#define         CRMOD           0000020                 /* map CR->LF;echo CR or LF as CR-LF*/
#define         RAW             0000040                 /* raw character input */
#define         ODDP            0000100                 /* odd parity rcvd/xmtd */
#define         EVENP           0000200                 /* even parity rcvd/xmtd */
#define         ANYP            0000300                 /* any parity mask */
#define         HDPLX           0000400                 /* Half duplex line */
#define         NOHUP           0001000                 /* don't drop DTR on last close */
```

```
#define     XCLUDE      0002000     /* disallow future opens */
#define     NOSLEEP     0004000     /* dont sleep if nothing is ready */
#define     NTDELAY     0010000     /* no tab delay flag */
#define     NLDELAY     0020000     /* no newline delay flag */
#define     TANDEM      0040000     /* xon/xoff enabled */
#define     STDTTY      0100000     /* non-std tty escapes and kills */

/*
 * Speeds
 */
#define     B0          0
#define     B50         1
#define     B75         2
#define     B110        3
#define     B134        4
#define     B150        5
#define     B200        6
#define     B300        7
#define     B600        8
#define     B1200       9
#define     B1800       10
#define     B2400       11
#define     B4800       12
#define     B9600       13
#define     EXTA        14
#define     EXTB        15

/*
 * Character length and stop bits.
 * Character length does not include parity or stop bits.
 * Ored with sm_ospeeed.
 */
#define     SETSTOP     0200        /* set to change stop or length bits */
#define     ONESTOP     0000
#define     TWOSTOP     0100        /* 1.5 stop bits at 75 baud */
#define     BITS5       0000
#define     BITS6       0020
#define     BITS7       0040
#define     BITS8       0060
#define     SLBITS      0160        /* Mask of stop and length bits */


/*
 * Command 2 -- set line
 * discipline of a line
 */
#define     STTY_LTYPE  2

/*
 * standard line discipline
 */
#define     STDLTYPE    0
struct {
            int         sl_fil1;
            char        sl_cmd;                 /* Command = 2 */
            char        sl_ltype;   /* Line discipline number = 0 */
            int         sl_fil2;
};

/*
 * line disciplines 1 and 2 reserved for
 * project specific line disciplines
 */
```

```
#define         PRJ1LTYPE       1
#define         PRJ2LTYPE       2

/*
 * transparent line discipline
 */
#define         TRSLTYPE        3
struct {
                char            ts_quanta;      /* Sleep quanta */
                char            ts_fil1;
                char            ts_cmd;                         /* Command = 2 */
                char            ts_ltype;       /* Line discipline number = 3 */
                char            ts_brk0;        /* First break character */
                char            ts_brk1;        /* Second break character */
};

/*
 * Half Duplex line discipline
 */
#define         HFLTYPE         4
struct {
                int             sl_fil1;
                char            sl_cmd;                         /* Command = 2 */
                char            sl_ltype;       /* Line discipline number = 4 */
                int             sl_fil2;
};

/*
 * Line disciplines 5 through 9 reserved for
 * future common line disciplines
 */
#define         RSV5LTYPE       5
#define         RSV6LTYPE       6
#define         RSV7LTYPE       7
#define         RSV8LTYPE       8
#define         RSV9LTYPE       9

/*
 * Command 3 -- set terminal type
 */
#define         STTY_TERM       3
struct {
                char            st_flgs;        /* terminal flags (see below) */
                char            st_fil1;
                char            st_cmd;                         /* Command = 3 */
                char            st_term;        /* Terminal type */
                int             st_fil2;
};

/*
 * Terminal types
 */
#define         TERM_NONE       0               /* tty */
#define         TERM_TEC        1               /* TEC Scope */
#define         TERM_V61        2               /* DEC VT61 */
#define         TERM_V10        3               /* DEC VT100 */
#define         TERM_TEX        4               /* Tektronix 4023 */
#define         TERM_D40        5               /* TTY Mod 40/1 */
#define         TERM_H45        6               /* Hewlitt-Packard 45 */
#define         TERM_D42        7               /* TTY Mod 40/2B */

/*
 * Terminal flags
```

```
 */
#define         TM_NONE                 0               /* use default flags */
#define         TM_SNL                  1               /* special newline flag */
#define         TM_ANL                  2               /* auto newline on column 80 */
#define         TM_LCF                  4               /* last col of last row special */
#define         TM_CECHO        010             /* echo terminal cursor control */
#define         TM_CINVIS       020             /* do not send esc sequences to user */
#define         TM_SET                          0200            /* must be on to set/reset flags */


/*
 * Command 4 -- set variable portion
 * of crt screen
 */
#define         STTY_SCREEN     4
struct {
                char            ss_crow;        /* cursor's row */
                                                        /* ignored on stty */
                char            ss_fil1;
                char            ss_cmd;                 /* Command = 4 */
                char            ss_vrow;        /* variable row */
                int             ss_fil2;
};


/*
 * Command 0377 -- enable spy
 */
#define         STTY_SPY        0377
struct {
                int             sy_fil1;
                char            sy_cmd;                 /* Command = 0377 */
                char            sy_scmd;        /* 0=>delete spy; 1=>initiate spy */
                int             sy_fil2;
};


/*
 * stty info for named pipes ONLY
 */
#define         STTY_NPIPE      0376
struct {
                int             sp_rflg;        /* read flag; 0 => nosleep */
                char            sp_cmd;                 /* Command = 0376 */
                char            sp_fil1;
                int             sp_wflg;        /* write flag; 0 => nosleep */
};
```

Notice that the format of the SGTTY structure may be different for various *stty*/*gtty* commands. The only byte which is always used is the command byte. This byte appears in all the structure definitions and must be filled in by the user before utilizing the *stty* or *gtty* system calls. The user should declare any *stty* or *gtty* structures using the structure tag-name **SGBUF**. Note, however, that references to the structure may be made using the **SGBUF** structure or any of the untagged structures defined above.

If the command byte is **STTY_MODES** or **STTY_NFMODES** the system call will set or get the input speed, output speed, number of data and stop bits, and the teletype modes. If an attempt is made to change the speed of a nonprogrammable device (e.g., DJ-11) or change the speed to a unsupported speed (e.g., **B4800** on a DC-11) the present speed is left unchanged.

Certain modes require further explaination:

LCASE Map upper case to lower case on input; map lower case to uppercase on output. Map | to \!; ‘ to \’; { to \(; } to \); ~ to \^; and map \<C> to upper case input where <C> is any upper case character.

RAW In raw mode, every character is passed immediately to the program without waiting for a full line to be typed. No input characters have special meaning. (e.g., The interrupt character (DEL) will not cause the program to be interrupted but will be sent to the program as a character.) **LCASE** and **CRMOD** will still cause input mapping. Output character processing is unaffected. If the transmitter has been stopped by the ESC key, setting **RAW** will release it. Note, however, that this can only be effective if the **STTY_NFMODES** command is utilized. Otherwise the program will wait for the ESC key to be depressed again. Input and output data width is eight bits, but the eighth bit may be a parity bit depending upon the setting of **ODDP** and **EVENP**.

ODD, EVENP

For the standard line discipline a character will be rejected unless its parity matches that expected. If both bits are set either parity is accepted and even parity is transmitted. If both bits are set and **RAW** is set the parity is visible to and supplied by the user on input and output. If neither bit is set no characters are accepted and even parity is transmitted.

HDPLX For those communications controllers with the capability, disable reception during transmission.

XCLUDE When set, no one may open the line. Cleared upon the last close.

NOSLEEP

Return a zero if a read is performed and no characters are present. Don't wait to flush output on *close* or *stty*. Don't wait for carrier in the first *read* or *write* after an *open* if carrier is not up. Normally a process will block when waiting for carrier to come up after an *open*. This roadblock will take place in the first *read* or *write* not the *open*.

STDTTY Change the erase character from # to _ and the delete line character from @ to $. In addition to CR and LF, wake up on / and !, and generate an interrupt upon reception of & or DEL.

It is also possible for the user to set the number of data and stop bits if the defaults are not satisfactory. The default is **TWOSTOP** at **B75** and **B110**, **ONESTOP** otherwise; and **BITS5** for **B75**, **BITS7** plus one even parity bit otherwise. In order to set these bits the **SETSTOP** bit must also be set.

Normally a *stty* will wait for output to flush before doing anything. This can be circumvented by using the command **STTY_NFMODES**.

The **STTY_LTYPE** command may be used to change the line discipline (protocol) used on a line. The normal CB-UNIX line discipline is **STDLTYPE**. Also commonly supported is the half duplex line discipline **HFLTYPE**, and the transparent line discipline **TRSLTYPE**. Different line disciplines expect different format in the *stty/ gtty* structure. **STLDTYPE** and **HFLTYPE** require no additional information.

TRSLTYPE is a line discipline that allows the user full eight bit transparency on input and output with or without parity. For this line discipline a *write* will perform no mapping. A read will return upon the occurrence of the first of three conditions as specified by the user:

1) The requested number of characters have arrived.

2) The number of seconds, *ts_quanta*, has elapsed.

3) A break character has arrived.

If *ts_quanta* is zero timing is disabled, otherwise *ts_quanta* is the maximum wait time in seconds. If *ts_brk0* and *ts_brk1* are both zero no break characters will awaken the process. If *ts_brk1* is 0377 then *ts_brk0* is taken as a single break character. Otherwise both break characters are assumed valid. **NCDELAY**, **XTABS**, **LCASE**, **ECHO**, **CRMOD**, **RAW**, **NTDELAY**, **NLDELAY**, and **STDTTY** have no meaning for this line discipline.

The **STTY_TERM** command is used to specify the type of CRT connected to a line. **TERM_NONE** is the standard, non-CRT, type. If a type other than **TERM_NONE** is specified input and output mapping will occur for the CRT language defined in the header file **<crtctl.h>**. In this case the ESC character takes on special meaning, escaping the subsequent characters on input and output. The terminal flags *st_flag* and modes *st_modes* are given a default set of values when a terminal type is set. The modes may be subsequently changed with a **STTY_MODES** command. The flags may be changed by setting the **TM_SET** bit when changing the terminal type and specifying the flag bits. The flag bits require further clarification:

TM_SNL Handle new lines specially if the terminal driver is so equipped.

TM_ANL Provide a carriage return and newline when writing beyond column eighty.

TM_LCF Immediately before placing a character in the last column and last row, delete the top line, print the character in the last column of the now second to last row, and then move the cursor to column one of the new last line. This function is required for terminal that move the cursor to "bad" places when printing in the last position.

TM_CECHO
Echo the control sequences such as cursor up when received.

TM_CINVIS
Do not pass the cursor control characters to the user program on input.

The **STTY_SCREEN** command is also used to set or get information about CRT terminals. It is used to set or get the variable row for split screen operation and to get the current row number of the cursor.

The **STTY_SPY** command will cause any output directed to the terminal specified by *fildes* to be copied to the controlling terminal of the program performing the *stty*. Only one spy operation may be active in the entire system at any time. The spy continues until explicitly turned off. Currently spy is only effective on lines using the **STDLTYPE** line discipline.

Finally, the **STTY_NPIPE** command can be used on named pipes to prevent *read*s or *write*s to named pipes from roadblocking. If *sp_rflg* is nonzero then a reader of the named pipe will roadblock when a *read* is performed with no data in the pipe, otherwise a zero is returned immediately. Similarly if *sp_wflg* is nonzero a *write* will roadblock if the pipe is full. When a named pipe is first opened *sp_rflg* is set to one and *sp_wflg* is zero.

Stty has been replaced by *ioctl*(2) in the new implementation of the library.

SEE ALSO

stty(1), ioctl(2)

ASSEMBLER

(stty = 31.)
(file descriptor in r0)
sys stty; arg

(gtty = 32.)
(file descriptor in r0)
sys gtty; arg

NAME

sync − update super-block

SYNOPSIS

sync ( )

DESCRIPTION

Sync causes all information in memory that should be on disk to be written out. This includes modified super blocks, modified i-nodes, and delayed block I/O.

It should be used by programs which examine a file system, for example *check*(1), *df*(1), etc.,.

SEE ALSO

sync(1), update(1)

ASSEMBLER

(sync = 36.; not in assembler)

sys sync

NAME

tell − get file offset

SYNOPSIS

long tell (file)
int file;

DESCRIPTION

Tell returns the current read/write pointer associated with the open file whose descriptor is specified as argument.

Tell is obsolete − use *lseek*(2) instead.

SEE ALSO

lseek(2)

DIAGNOSTICS

C-bit set or −1 returned for an unknown file descriptor.

ASSEMBLER

(tell = 40.)
(file descriptor in r0)
sys tell
(offset in r0-r1)

NAME

time − get date and time

SYNOPSIS

time (tvec)
int tvec[2];

DESCRIPTION

Time returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds. From *as*, the high order word is in the r0 register and the low order is in r1. From C, the user-supplied vector is filled in.

SEE ALSO

date(1), stime(2), ctime(3)

ASSEMBLER

(time = 13.)
sys time (time in r0, r1)

NAME

times − get process times

SYNOPSIS

long times (buffer)
struct tbuffer *buffer;

DESCRIPTION

Times fills the structure addressed by *buffer* with time-accounting information for the current process and for the terminated child processes of the current process. All times are in 1/60 seconds.

After the call, the buffer will appear as follows:

```
struct tbuffer {
        long    proc_user_time;
        long    proc_system_time;
        long    child_user_time;
        long    child_system_time;
};
```

The time for a child is the sum of its process time and its children's times.

The value returned by *times* is the elapsed time, in 60ths of a second, since a point in the past. This point does not vary from one invocation of *times* to another, but is otherwise arbitrary, so that while the value returned by a single call to *times* is not meaningful in itself, the difference between two calls can be used for accurate calculation of elapsed time.

SEE ALSO

time(1), time(2)

ASSEMBLER

(times = 43.)
sys times; buffer
(elapsed time in r0-r1)

NAME

ucore − enable/disable unique core dumping feature.

SYNOPSIS

int ucore (mode)
int mode ;

DESCRIPTION

Ucore turns on the unique core dumping feature if *mode* is non-zero. If *mode* is zero, the feature is disable. This is the default condition. The previous state is returned by *ucore*. When the feature is disabled, cores drop in **core**, while when it is enabled, they drop in **core.***nnnnn*, where *nnnnn* is the process id of the process which is dying.

SEE ALSO

ucore(1)

ASSEMBLER

(syscb = 45. ; ucore = 5.)
(ucore in R1)
sys syscb; mode
(R0 = previous state)

NAME

umask – set and get creation mask

SYNOPSIS

umask (mask)

DESCRIPTION

Umask sets the process inode creation mask to *mask* and returns the previous value of *mask*. The creation mask indicates automatic restrictions placed on the access permissions (read, write, execute) at the time of file creation (initial *creat* and *mknod*), that is, each bit set in the mask clears the corresponding permission mode bit.

For example, a mask of 0 would leave the modes unaltered. A mask of 022 would remove write permission for the group and others even if specified in the *creat* or *mknod* system call. A mask value of 077 would remove all group and others permissions.

SEE ALSO

mkdir(1), sh(1), mknod(1, 2), creat(2), chmod(2)

ASSEMBLER

(syscb = 45.; umask = 1)
(new mask in r0)
(umask in r1)
sys syscb; ..; ..
(old mask in r0)

NAME

umount − dismount file system

SYNOPSIS

umount (special)
char *special;

DESCRIPTION

Umount announces to the system that special file *special* is no longer to contain a removable file system. A *close* is issued to the pertinent device driver. The file associated with the special file reverts to its ordinary interpretation; see *mount*(2). Only the super-user may unmount a file system.

SEE ALSO

umount(1), mount(2)

DIAGNOSTICS

Error bit (c-bit) set if no file system was mounted on the special file, if there are still active files on the mounted file system, or if the user is not super-user. From C, a −1 return indicates an error.

ASSEMBLER

(umount = 22.)
sys umount; special

NAME

uname − get name of current UNIX system

SYNOPSIS

#include <sys/utsname.h>

int uname (name)
char *name;

DESCRIPTION

Uname stores in the structure pointed to by *name* information identifying the current UNIX system.

Uname uses the structure defined in **<sys/utsname.h>**:

```
/*              @(#)/usr/src/ucb/sys/utsname.h  3.1             */
struct utsname {
                char            sysname[9];
                char            nodename[9];
                char            release[9];
                char            version[9];
};
extern struct utsname utsname;
```

Uname returns in *sysname* a null-terminated character name of the current UNIX system. Similarly, *nodename* may contain the name that the system is known by on a communications network. *Release* and *version* further identify the operating system.

SEE ALSO

uname(1)

DIAGNOSTICS

The error bit (c-bit) is set if *name* can not be written. From C, a −1 return indicates an error.

ASSEMBLER

(utssys = 57.; uname = 0)
(pointer to name in r0)
sys utssys; uname

NAME

unlink − remove directory entry

SYNOPSIS

int unlink (name)
char *name;

DESCRIPTION

Name points to a null-terminated string. *Unlink* removes the entry for the file pointed to by *name* from its directory. If this entry was the last link to the file, the contents of the file are freed and the file is destroyed. If, however, the file was open in any process, the actual destruction is delayed until it is closed, even though the directory entry has disappeared.

SEE ALSO

rm(1), link(2)

DIAGNOSTICS

Zero is normally returned; −1 indicates that the file does not exist, that its directory cannot be written, or that the file contains pure procedure text that is currently in use. Write permission is not required on the file itself. Only the super-user may unlink a directory.

ASSEMBLER

(unlink = 10.)
sys unlink; name

NAME

utime — update times in file

SYNOPSIS

```
int utime (name, times)
char *name;
struct utimbuf *times;
```

DESCRIPTION

Utime is used to set both the access and modification times of a file. *Name* points to a null-terminated string naming a file, and *times* points to a structure containing two long integer time values:

```
struct utimbuf {
        long int actime;                /* access time */
        long int modtime;       /* modification time */
};
```

Only the owner of the file and the super-user may issue this call in this way.

Another way to use *utime* is to set *times* to NULL; in this case, the access and modification times of the file are set to the current time, and the user need only have write access to the file.

SEE ALSO

stat(2)

DIAGNOSTICS

The error bit (c-bit) is set if *name* does not exist, if permission is denied, or if the file system is read-only. From C, a −1 return indicates an error.

ASSEMBLER

(utime = 30.)

sys utime; file; timep

NAME

wait − wait for process to die

SYNOPSIS

wait (&status)
struct { char lobyte; char hibyte; } status;

DESCRIPTION

Wait causes its caller to delay until one of its child processes terminates. If any child has died since the last *wait*, return is immediate; if there are no children, return is immediate with the error bit set (resp. with a value of −1 returned). In the case of several children several *wait* calls are needed to learn of all the deaths.

If no error is indicated on return, the r1 high byte, i.e. *status.hibyte*, contains the low byte of the child process r0, i.e. the argument of *exit*, when it terminated. The r1 low byte, i.e. *status.lobyte*, contains the termination status of the process. See *signal*(2) for a list of termination statuses (signals); **0** status indicates normal termination. If the **0200** bit of the termination status is set, a core image of the process was produced by the system. Status **0177** is returned for a stopped process which has not terminated and can be restarted (see *ptrace*(2)).

On return, r0 contains the process ID of the dead child. From C, the process ID of the child is the returned value.

SEE ALSO

exit(2), fork(2), signal(2)

DIAGNOSTICS

The error bit (c-bit) on if no children not previously waited for. From C, a returned value of −1 indicates an error.

ASSEMBLER

(wait = 7.)
sys wait
(process id in r0)
(status in r1)

NAME

write − write on a file

SYNOPSIS

write (fildes, buffer, nbytes)
char *buffer;

DESCRIPTION

A file descriptor is a word returned from a successful *open*, *creat*, *dup*, or *pipe* call.

Buffer is the address of *nbytes* contiguous bytes which are written on the output file. The number of characters actually written is returned (in r0). It should be regarded as an error if this is not the same as requested.

Writes which are multiples of 512 characters long and begin on a 512-byte boundary are more efficient than any others.

SEE ALSO

creat(2), open(2), pipe(2), read(2)

DIAGNOSTICS

The error bit (c-bit) is set on an error: bad descriptor, buffer address, or count; physical I/O errors. From C, a returned value of −1 indicates an error.

ASSEMBLER

(write = 4.)
(file descriptor in r0)
sys write; buffer; nbytes
(byte count in r0)