

C Programming

# List of Linux Syscalls

2 years ago • Robert Oliver

In this guide you'll find a full list of Linux syscalls along with their definition, parameters, and commonly used flags.

You can combine multiple flags by using a logical AND and passing the result to the argument in question.

**Some notes about this guide:**

---

**MY LATEST VIDEOS**

00:24 / 02:20

- 
- Calls that have been long deprecated or removed have been omitted.
  - Items pertaining to outdated or infrequently used architectures (i.e. MIPS, PowerPC) are generally omitted.
  - Structures are defined only once. If a struct is mentioned and cannot be found in the syscall, please search the document for its definition.

Source materials include man pages, kernel source, and kernel development headers.

## Table of Contents

- [List of Linux Syscalls](#)
- [Table of Contents](#)
  - [read](#)
  - [write](#)
  - [open](#)

- [open flags](#)
- [close](#)
- [stat](#)
- [fstat](#)
- [lstat](#)
- [poll](#)
- [lseek](#)
  - whence flags
- [mmap](#)
  - prot flags
  - flags
- [mprotect](#)
  - prot flags
- [munmap](#)
- [brk](#)
- [rt\\_sigaction](#)
- [rt\\_sigprocmask](#)
  - how flags
- [rt\\_sigreturn](#)
- [ioctl](#)
- [pread64](#)
- [pwrite64](#)
- [readv](#)
- [writev](#)

- [access](#)
- [pipe](#)
- [select](#)
- [sched\\_yield](#)
- [mremap](#)
  - [flags](#)
- [msync](#)
  - [flags](#)
- [mincore](#)
- [madvise](#)
  - [advice](#)
- [shmget](#)
  - [shmflg](#)
- [shmat](#)
  - [shmflg](#)
- [shmctl](#)
  - [cmd](#)
- [dup](#)
- [dup2](#)
- [pause](#)
- [nanosleep](#)
- [getitimer](#)
  - [which timers](#)
- [alarm](#)

- [setitimer](#)
- [getpid](#)
- [sendfile](#)
- [socket](#)
  - [domain flags](#)
  - [type flags](#)
- [connect](#)
- [accept](#)
- [sendto](#)
  - [flags](#)
- [recvfrom](#)
  - [flags](#)
- [sendmsg](#)
- [recvmsg](#)
- [shutdown](#)
  - [how](#)
- [bind](#)
- [listen](#)
- [getsockname](#)
- [getpeername](#)
- [socketpair](#)
- [setsockopt](#)
- [getsockopt](#)
- [clone](#)

- [flags](#)
- [fork](#)
- [vfork](#)
- [execve](#)
- [exit](#)
- [wait4](#)
  - [options](#)
- [kill](#)
- [getppid](#)
- [uname](#)
- [semget](#)
- [semop](#)
- [semctl](#)
  - [cmd](#)
- [shmdt](#)
- [msgget](#)
- [msgsnd](#)
  - [msgflg](#)
- [msgrcv](#)
- [msgctl](#)
  - [cmd](#)
- [fcntl](#)
  - [cmd](#)
  - [flock](#)

- [f\\_owner\\_ex](#)
- [flock](#)
  - [operation](#)
- [fsync](#)
- [fdatasync](#)
- [truncate](#)
- [ftruncate](#)
- [getdents](#)
  - [types](#)
- [getcwd](#)
- [chdir](#)
- [fchdir](#)
- [rename](#)
- [mkdir](#)
- [rmdir](#)
- [creat](#)
- [link](#)
- [unlink](#)
- [symlink](#)
- [readlink](#)
- [chmod](#)
- [fchmod](#)
- [chown](#)
- [fchown](#)

- [lchown](#)
- [umask](#)
- [gettimeofday](#)
- [getrlimit](#)
  - [resource flags](#)
- [getrusage](#)
  - [who target](#)
- [sysinfo](#)
- [times](#)
- [ptrace](#)
  - [request flags](#)
- [getuid](#)
- [syslog](#)
  - [type flag](#)
- [getgid](#)
- [setuid](#)
- [setgid](#)
- [geteuid](#)
- [getegid](#)
- [setpgid](#)
- [getppid](#)
- [getpgrp](#)
- [setsid](#)
- [setreuid](#)

- [setregid](#)
- [getgroups](#)
- [setgroups](#)
- [setresuid](#)
- [setresgid](#)
- [getresuid](#)
- [getresgid](#)
- [getpgid](#)
- [setfsuid](#)
- [setfsgid](#)
- [getsid](#)
- [capget](#)
- [capset](#)
- [rt\\_sigpending](#)
- [rt\\_sigtimedwait](#)
- [rt\\_sigqueueinfo](#)
- [rt\\_sigsuspend](#)
- [sigaltstack](#)
- [utime](#)
- [mknod](#)
- [uselib](#)
- [personality](#)
- [ustat](#)
- [statfs](#)

- [fstatfs](#)
- [sysfs](#)
- [getpriority](#)
  - [which](#)
- [setpriority](#)
- [sched\\_setparam](#)
- [sched\\_getparam](#)
- [sched\\_setscheduler](#)
  - [policy](#)
- [sched\\_getscheduler](#)
- [sched\\_get\\_priority\\_max](#)
- [sched\\_get\\_priority\\_min](#)
- [sched\\_rr\\_get\\_interval](#)
- [mlock](#)
- [munlock](#)
- [mlockall](#)
  - [flags](#)
- [munlockall](#)
- [vhangup](#)
- [modify\\_ldt](#)
- [pivot\\_root](#)
- [prctl](#)
  - [option](#)
- [arch\\_prctl](#)

- adjtimex
- setrlimit
- chroot
- sync
- acct
- settimeofday
- mount
  - mountflags
- umount2
  - flags
- swapon
  - swapflags
- swapoff
- reboot
  - arg
- sethostname
- setdomainname
- iopl
- ioperm
- init\_module
- delete\_module
  - flags
- quotactl
  - cmd

- [gettid](#)
- [readahead](#)
- [setxattr](#)
- [Isetxattr](#)
- [fsetxattr](#)
- [getxattr](#)
- [Igetxattr](#)
- [fgetxattr](#)
- [listxattr](#)
- [llistxattr](#)
- [flistxattr](#)
- [removexattr](#)
- [Iremovexattr](#)
- [fremovexattr](#)
- [tkill](#)
- [time](#)
- [futex](#)
  - [op](#)
- [sched\\_setaffinity](#)
- [sched\\_getaffinity](#)
- [set\\_thread\\_area](#)
- [io\\_setup](#)
- [io\\_destroy](#)
- [io\\_getevents](#)

- [io\\_submit](#)
- [io\\_cancel](#)
- [get\\_thread\\_area](#)
- [lookup\\_dcookie](#)
- [epoll\\_create](#)
- [getdents64](#)
- [set\\_tid\\_address](#)
- [restart\\_syscall](#)
- [semtimedop](#)
- [fadvise64](#)
  - [advice](#)
- [timer\\_create](#)
  - [clockid](#)
- [timer\\_settime](#)
- [timer\\_gettime](#)
- [timer\\_getoverrun](#)
- [timer\\_delete](#)
- [clock\\_settime](#)
- [clock\\_gettime](#)
- [clock\\_getres](#)
- [clock\\_nanosleep](#)
- [exit\\_group](#)
- [epoll\\_wait](#)
- [epoll\\_ctl](#)

- [op](#)
- [tgkill](#)
- [utimes](#)
- [mbind](#)
  - [mode](#)
- [set\\_mempolicy](#)
- [get\\_mempolicy](#)
  - [flags](#)
- [mq\\_open](#)
  - [oflag](#)
- [mq\\_unlink](#)
- [mq\\_timedsend](#)
- [mq\\_timedreceive](#)
- [mq\\_notify](#)
- [kexec\\_load](#)
  - [flags](#)
- [waitid](#)
  - [options](#)
- [add\\_key](#)
  - [keyring](#)
- [request\\_key](#)
- [keyctl](#)
  - [cmd](#)
- [ioprio\\_set](#)

- [which](#)
- [ioprio\\_get](#)
- [inotify\\_init](#)
- [inotify\\_add\\_watch](#)
- [inotify\\_rm\\_watch](#)
- [migrate\\_pages](#)
- [openat](#)
- [mkdirat](#)
- [mknodat](#)
- [fchownat](#)
- [unlinkat](#)
- [renameat](#)
- [linkat](#)
- [symlinkat](#)
- [readlinkat](#)
- [fchmodat](#)
- [faccessat](#)
- [pselect6](#)
- [ppoll](#)
- [unshare](#)
  - [flags](#)
- [set\\_robust\\_list](#)
- [get\\_robust\\_list](#)
- [splice](#)

- flags
- tee
- sync\_file\_range
  - flags
- vmsplice
- move\_pages
  - flags
- utimensat
- epoll\_pwait
- signalfd
- timerfd\_create
- eventfd
  - flags
- fallocate
  - mode
- timerfd\_settime
- timerfd\_gettime
- accept4
- signalfd4
- eventfd2
- epoll\_create1
- dup3
- pipe2
- inotify\_init1

- [preadv](#)
- [pwritev](#)
- [rt\\_tgsigqueueinfo](#)
- [perf\\_event\\_open](#)
  - [flags](#)
- [recvmsg](#)
- [fanotify\\_init](#)
  - [flags](#)
  - [event\\_f\\_flags](#)
- [fanotify\\_mark](#)
  - [dirfd](#)
  - [flags](#)
- [name\\_to\\_handle\\_at](#)
- [open\\_by\\_handle\\_at](#)
- [syncfs](#)
- [sendmmsg](#)
- [setns](#)
  - [nsflag](#)
- [getcpu](#)
- [process\\_vm\\_readv](#)
- [process\\_vm\\_writev](#)
- [kcmp](#)
  - [type flags](#)
- [finit\\_module](#)

- [flags](#)

## read

Reads from a specified file using a file descriptor. Before using this call, you must first obtain a file descriptor using the `opensyscall`. Returns bytes read successfully.

```
ssize_t read(int fd, void *buf, size_t count)
```

- fd – file descriptor
- buf – pointer to the buffer to fill with read contents
- count – number of bytes to read

## write

Writes to a specified file using a file descriptor. Before using this call, you must first obtain a file descriptor using the `open` syscall. Returns bytes written successfully.

```
ssize_t write(int fd, const void *buf, size_t count)
```

- fd – file descriptor
- buf – pointer to the buffer to write
- count – number of bytes to write

## open

Opens or creates a file, depending on the flags passed to the call. Returns an integer with the file descriptor.

```
int open(const char *pathname, int flags, mode_t mode)
```

- `pathname` – pointer to a buffer containing the full path and filename
- `flags` – integer with operation flags (see below)
- `mode` – (optional) defines the permissions mode if file is to be created

## open flags

- `O_APPEND` – append to existing file
- `O_ASYNC` – use signal-driven IO
- `O_CLOEXEC` – use close-on-exec (avoid race conditions and lock contentions)
- `O_CREAT` – create file if it doesn't exist
- `O_DIRECT` – bypass cache (slower)
- `O_DIRECTORY` – fail if pathname isn't a directory
- `O_DSYNC` – ensure output is sent to hardware and metadata written before return
- `O_EXCL` – ensure creation of file
- `O_LARGEFILE` – allows use of file sizes represented by `off64_t`
- `O_NOATIME` – do not increment access time upon open
- `O_NOCTTY` – if pathname is a terminal device, don't become controlling terminal
- `O_NOFOLLOW` – fail if pathname is symbolic link
- `O_NONBLOCK` – if possible, open file with non-blocking IO
- `O_NDELAY` – same as `O_NONBLOCK`
- `O_PATH` – open descriptor for obtaining permissions and status of a file but does not allow read/write operations
- `O_SYNC` – wait for IO to complete before returning
- `O_TMPFILE` – create an unnamed, unreachable (via any other open call) temporary file
- `O_TRUNC` – if file exists, overwrite it (careful!)

## close

Close a file descriptor. After successful execution, it can no longer be used to reference the file.

```
int close(int fd)
• fd – file descriptor to close
```

## stat

Returns information about a file in a structure named stat.

```
int stat(const char *path, struct stat *buf);
• path – pointer to the name of the file
• buf – pointer to the structure to receive file information
```

On success, the buf structure is filled with the following data:

```
struct stat {
    dev_t      st_dev;      /* device ID of device with file */
    ino_t      st_ino;      /* inode */
    mode_t     st_mode;     /* permission mode */
    nlink_t    st_nlink;    /* number of hard links to file */
    uid_t      st_uid;      /* owner user ID */
    gid_t      st_gid;      /* owner group ID */
    dev_t      st_rdev;     /* device ID (only if device file) */
    off_t      st_size;     /* total size (bytes) */
    blksize_t  st_blksize;  /* blocksize for I/O */
```

```
blkcnt_t st_blocks; /* number of 512 byte blocks allocated */  
time_t st_atime; /* last access time */  
time_t st_mtime; /* last modification time */  
time_t st_ctime; /* last status change time */  
};
```

## fstat

Works exactly like the stat syscall except a file descriptor (`fd`) is provided instead of a path.

```
int fstat(int fd, struct stat *buf);
```

- `fd` – file descriptor
- `buf` – pointer to stat buffer (described in stat syscall)

Return data in `buf` is identical to the stat call.

## lstat

Works exactly like the stat syscall, but if the file in question is a symbolic link, information on the link is returned rather than its target.

```
int lstat(const char *path, struct stat *buf);
```

- `path` – full path to file
- `buf` – pointer to stat buffer (described in stat syscall)

Return data in `buf` is identical to the stat call.

## poll

Wait for an event to occur on the specified file descriptor.

```
int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

- `fds` – pointer to an array of `pollfd` structures (described below)
- `nfds` – number of `pollfd` items in the `fds` array
- `timeout` – Sets the number of milliseconds the syscall should block (negative forces `poll` to return immediately)

```
struct pollfd {  
    int fd;          /* file descriptor */  
    short events;   /* events requested for polling */  
    short revents;  /* events that occurred during polling */  
};
```

## lseek

This syscall repositions the read/write offset of the associated file descriptor. Useful for setting the position to a specific location to read or write starting from that offset.

```
off_t lseek(int fd, off_t offset, int whence)
```

- `fd` – file descriptor
- `offset` – offset to read/write from
- `whence` – specifies offset relation and seek behavior

## whence flags

- SEEK\_SET – offset is the absolute offset position in the file
- SEEK\_CUR – offset is the current offset location plus offset
- SEEK\_END – offset is the file size plus offset
- SEEK\_DATA – set offset to next location greater or equal to offset that contains data
- SEEK\_HOLE – set offset to next hole in file great or equal to offset

Returns resulting offset in bytes from the start of the file.

## mmap

Maps files or devices into memory.

```
void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset)
```

- addr – location hint for mapping location in memory, otherwise, if NULL, kernel assigns address
- length – length of the mapping
- prot – specifies memory protection of the mapping
- flags – control visibility of mapping with other processes
- fd – file descriptor
- offset – file offset

Returns a pointer to the mapped file in memory.

## prot flags

- PROT\_EXEC – allows execution of mapped pages
- PROT\_READ – allows reading of mapped pages
- PROT\_WRITE – allows mapped pages to be written
- PROT\_NONE – prevents access of mapped pages

## flags

- MAP\_SHARED – allows other processes to use this mapping
- MAP\_SHARED\_VALIDATE – same as MAP\_SHARED but ensures all flags are valid
- MAP\_PRIVATE – prevents other processes from using this mapping
- MAP\_32BIT – tells the kernel to locate mapping in the first 2 GB of RAM
- MAP\_ANONYMOUS – lets the mapping not be backed by any file (thus ignoring fd  
)
- MAP\_FIXED – treats `addr` argument as an actual address and not a hint
- MAP\_FIXED\_NOREPLACE – same as MAP\_FIXED but prevents clobbering existing mapped ranges
- MAP\_GROWSDOWN – tells the kernel to expand mapping downward in RAM (useful for stacks)
- MAP\_HUGETB – forces use of huge pages in mapping
- MAP\_HUGE\_1MB – use with MAP\_HUGETB to set 1 MB pages
- MAP\_HUGE\_2MB – use with MAP\_HUGETB to set 2 MB pages
- MAP\_LOCKED – maps the region to be locked (similar behavior to `mlock`)
- MAP\_NONBLOCK – prevents read-ahead for this mapping
- MAP\_NORESERVE – prevents allocation of swap space for this mapping
- MAP\_POPULATE – tells the kernel to populate page tables for this mapping (causing read-ahead)

- MAP\_STACK – tells the kernel to allocate address suitable for use in a stack
- MAP\_UNINITIALIZED – prevents clearing of anonymous pages

## mprotect

Sets or adjusts protection on a region of memory.

```
int mprotect(void *addr, size_t len, int prot)
```

- addr – pointer to region in memory
- prot – protection flag

Returns zero when successful.

## prot flags

- PROT\_NONE – prevents access to memory
- PROT\_READ – allows reading of memory
- PROT\_EXEC – allows execution of memory
- PROT\_WRITE – allows memory to be modified
- PROT\_SEM – allows memory to be used in atomic operations
- PROT\_GROWSUP – sets protection mode upward (for architectures that have stack that grows upward)
- PROT\_GROWSDOWN – sets protection mode downward (useful for stack memory)

## munmap

Unmaps mapped files or devices.

```
int munmap(void *addr, size_t len)
• addr – pointer to mapped address
• len – size of mapping
```

Returns zero when successful.

## brk

Allows for altering the program break that defines end of process's data segment.

```
int brk(void *addr)
• addr – new program break address pointer
```

Returns zero when successful.

## rt\_sigaction

Change action taken when process receives a specific signal (except SIGKILL and SIGSTOP).

```
int rt_sigaction(int signum, const struct sigaction *act, struct sigaction *oldact)
• signum – signal number
• act – structure for the new action
• oldact – structure for the old action
```

```
struct sigaction {
    void (*sa_handler)(int);
```

```
void      (*sa_sigaction)(int, siginfo_t *, void *);  
sigset_t  sa_mask;  
int       sa_flags;  
void      (*sa_restorer)(void);  
};  
  
siginfo_t {  
    int      si_signo;          /* signal number */  
    int      si_errno;          /* errno value */  
    int      si_code;           /* signal code */  
    int      si_trapno;         /* trap that caused hardware signal (unusual on most archit  
    pid_t    si_pid;            /* sending PID */  
    uid_t    si_uid;            /* real UID of sending program */  
    int      si_status;          /* exit value or signal */  
    clock_t  si_utime;          /* user time consumed */  
    clock_t  si_stime;          /* system time consumed */  
    sigval_t si_value;          /* signal value */  
    int      si_int;             /* POSIX.1b signal */  
    void    *si_ptr;             /* POSIX.1b signal */  
    int      si_overrun;         /* count of timer overrun */  
    int      si_timerid;         /* timer ID */  
    void    *si_addr;            /* memory location that generated fault */  
    long    si_band;             /* band event */  
    int      si_fd;              /* file descriptor */  
    short   si_addr_lsb;         /* LSB of address */  
    void    *si_lower;            /* lower bound when address violation occurred */  
    void    *si_upper;            /* upper bound when address violation occurred */  
    int      si_pkey;             /* protection key on PTE causing fault */  
    void    *si_call_addr;         /* address of system call instruction */  
    int      si_syscall;          /* number of attempted syscall */
```

```
unsigned int si_arch; /* arch of attempted syscall */  
}
```



## rt\_sigprocmask

Retreive and/or set the signal mask of the thread.

```
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset)
```

- how – flag to determine call behavior
- set – new signal mask (NULL to leave unchanged)
- oldset – previous signal mask

Returns zero upon success.

### how flags

- SIG\_BLOCK – set mask to block according to set
- SIG\_UNBLOCK – set mask to allow according to set
- SIG\_SETMASK – set mask to set

## rt\_sigreturn

Return from signal handler and clean the stack frame.

```
int sigreturn(unsigned long __unused)
```

## ioctl

Set parameters of device files.

```
int ioctl(int d, int request, ...)
```

- d – open file descriptor the device file
- request – request code
- ... – untyped pointer

Returns zero upon success in most cases.

## pread64

Read from file or device starting at a specific offset.

```
ssize_t pread64(int fd, void *buf, size_t count, off_t offset)
```

- fd – file descriptor
- buf – pointer to read buffer
- count – bytes to read

- offset – offset to read from

Returns bytes read.

## pwrite64

Write to file or device starting at a specific offset.

```
ssize_t pwrite64(int fd, void *buf, size_t count, off_t offset)
```

- fd – file descriptor
- buf – pointer to buffer
- count – bytes to write
- offset – offset to start writing

Returns bytes written.

## readv

Read from file or device into multiple buffers.

```
ssize_t readv(int fd, const struct iovec *iov, int iovcnt)
```

- **fd** – file descriptor
- **iov** – pointer to iovec structure
- **iovcnt** – number of buffers (described by iovec)

```
struct iovec {  
    void *iov_base; /* Starting address */  
    size_t iov_len; /* Number of bytes to transfer */  
};
```

Returns bytes read.

## writenv

Write to file or device from multiple buffers.

```
ssize_t writenv(int fd, const struct iovec *iov, int iovcnt)
```

- **fd** – file descriptor
- **iov** – pointer to iovec structure
- **iovcnt** – number of buffers (described by iovec)

```
struct iovec {  
    void *iov_base; /* Starting address */
```

```
    size_t iov_len;      /* Number of bytes to transfer */  
};
```

Returns bytes written.

## access

Check permissions of current user for a specified file or device.

```
int access(const char *pathname, int mode)  
• pathname – file or device  
• mode – permissions check to perform
```

Returns zero on success.

## pipe

Create a pipe.

```
int pipe(int pipefd[2])
```

- pipefd – array of file descriptors with two ends of the pipe

Returns zero on success.

## select

Wait for file descriptors to become ready for I/O.

```
int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,  
struct timeval *timeout)
```

- nfds – number of file descriptors to monitor (add 1)
- readfds – fixed buffer with list of file descriptors to wait for read access
- writefds – fixed buffer with list of file descriptors to wait for write access
- exceptfds – fixed buffer with list of file descriptors to wait for exceptional conditions
- timeout – timeval structure with time to wait before returning

```
typedef struct fd_set {  
    u_int fd_count;  
    SOCKET fd_array[FD_SETSIZE];  
};
```

```
struct timeval {  
    long tv_sec;           /* seconds */  
    long tv_usec;          /* microseconds */  
};
```

Returns number of file descriptors, or zero if timeout occurs.

## **sched\_yield**

Yield CPU time back to the kernel or other processes.

```
int sched_yield(void)
```

Returns zero on success.

## mremap

Shrink or enlarge a memory region, possibly moving it in the process.

```
void *mremap(void *old_address, size_t old_size, size_t new_size, int flags, ... /* void  
*new_address */)
```

- `old_address` – pointer to the old address to remap
- `old_size` – size of old memory region
- `new_size` – size of new memory region
- `flags` – define additional behavior

## flags

- `MREMAP_MAYMOVE` – allow the kernel to move the region if there isn't enough room (default)
- `MREMAP_FIXED` – move the mapping (must also specify `MREMAP_MAYMOVE`)

## msync

Synchronize a memory-mapped file previously mapped with `mmap`.

```
int msync(void *addr, size_t length, int flags)
```

- `addr` – address of memory mapped file
- `length` – length of memory mapping
- `flags` – define additional behavior

## flags

- MS\_ASYNC – schedule sync but return immediately
- MS\_SYNC – wait until sync is complete
- MS\_INVALIDATE – invalidate other mappings of same file

Returns zero on success.

## mincore

Check if pages are in memory.

```
int mincore(void *addr, size_t length, unsigned char *vec)
```

- addr – address of memory to check
- length – length of memory segment
- vec – pointer to array sized to (length+PAGE\_SIZE-1) / PAGE\_SIZE that is clear if page is in memory

Returns zero, but `vec` must be referenced for presence of pages in memory.

## madvise

Advise kernel on how to use a given memory segment.

```
int madvise(void *addr, size_t length, int advice)
```

- `addr` – address of memory
- `length` – length of segment
- `advice` – advice flag

## advice

- `MADV_NORMAL` – no advice (default)
- `MADV_RANDOM` – pages can be in random order (read-ahead performance may be hampered)
- `MADV_SEQUENTIAL` – pages should be in sequential order
- `MADV_WILLNEED` – will need pages soon (hinting to kernel to schedule read-ahead)
- `MADV_DONTNEED` – do not need anytime soon (discourages read-ahead)

## shmget

Allocate System V shared memory segment.

```
int shmget(key_t key, size_t size, int shmflg)
```

- `key` – an identifier for the memory segment
- `size` – length of memory segment
- `shmflg` – behavior modifier flag

## shmflg

- `IPC_CREAT` – create a new segment
- `IPC_EXCL` – ensure creation happens, else call will fail
- `SHM_HUGETLB` – use huge pages when allocating segment
- `SHM_HUGE_1GB` – use 1 GB hugetlb size
- `SHM_HUGE_2M` – use 2 MB hugetlb size
- `SHM_NORESERVE` – do not reserve swap space for this segment

## shmat

Attach shared memory segment to calling process's memory space.

```
void *shmat(int shmid, const void *shmaddr, int shmflg)
```

- `shmid` – shared memory segment id
- `shmaddr` – shared memory segment address
- `shmflg` – define additional behavior

## shmflg

- SHM\_RDONLY – attach segment as read-only
- SHM\_REMAP – replace exiting mapping

## shmctl

Get or set control details on shared memory segment.

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf)
```

- shmid – shared memory segment id
- cmd – command flag
- buf – shmid\_ds structure buffer for return or set parameters

```
struct shmid_ds {  
    struct ipc_perm shm_perm; /* Ownership and permissions */  
    size_t         shm_segsz; /* Size of shared segment (bytes) */
```

```
time_t          shm_atime;    /* Last attach time */
time_t          shm_dtime;    /* Last detach time */
time_t          shm_ctime;    /* Last change time */
pid_t           shm_cpid;    /* PID of shared segment creator */
pid_t           shm_lpid;    /* PID of last shmat(2)/shmdt(2) syscall */
shmatt_t        shm_nattach; /* Number of current attaches */

};

struct ipc_perm {
    key_t          __key;      /* Key provided to shmget */
    uid_t          uid;        /* Effective UID of owner */
    gid_t          gid;        /* Effective GID of owner */
    uid_t          cuid;       /* Effective UID of creator */
    gid_t          cgid;       /* Effective GID of creator */
    unsigned short mode;     /* Permissions and SHM_DEST + SHM_LOCKED flags */
    unsigned short __seq;    /* Sequence */
};
```

Successful IPC\_INFO or SHM\_INFO syscalls return index of highest used entry in the kernel's array of shared memory segments. Successful SHM\_STAT syscalls return id of memory segment provided in shmid. Everything else returns zero upon success.

## cmd

- IPC\_STAT – get shared memory segment info and place in buffer
- IPC\_SET – set shared memory segment parameters defined in buffer
- IPC\_RMID – mark shared memory segment to be removed

## dup

Duplicate file descriptor.

```
int dup(int oldfd)
• oldfd – file descriptor to copy
```

Returns new file descriptor.

## dup2

Same as `dup` except `dup2` uses file descriptor number specified in `newfd`.

```
int dup2(int oldfd, int newfd)
• oldfd – file descriptor to copy
• newfd – new file descriptor
```

## pause

Wait for a signal, then return.

```
int pause(void)
```

Returns -1 when signal received.

## nanosleep

Same as sleep but with time specified in nanoseconds.

```
int nanosleep(const struct timespec *req, struct timespec *rem)
```

- req – pointer to syscall argument structure
- rem – pointer to structure with remaining time if interrupted by signal

```
struct timespec {  
    time_t tv_sec;           /* time in seconds */
```

```
    long    tv_nsec;      /* time in nanoseconds */  
};
```

Returns zero upon successful sleep, otherwise time elapsed is copied into `rem` structure.

## getitimer

Get value from an interval timer.

```
int getitimer(int which, struct itimerval *curr_value)  
• which – which kind of timer  
• curr_value – pointer to itimerval structure with argument details
```

```
struct itimerval {  
    struct timeval it_interval; /* Interval for periodic timer */  
    struct timeval it_value;   /* Time until next expiration */  
};
```

Returns zero on success.

## which timers

- ITIMER\_REAL – timer uses real time
- ITIMER\_VIRTUAL – timer uses user-mode CPU execution time
- ITIMER\_PROF – timer uses both user and system CPU execution time

## alarm

Set an alarm for delivery of signal SIGALRM.

```
unsigned int alarm(unsigned int seconds)
```

- seconds – send SIGALRM in x seconds

Returns number of seconds remaining until a previously set alarm will trigger, or zero if no alarm was previously set.

## setitimer

Create or destroy alarm specified by `which`.

```
int setitimer(int which, const struct itimerval *new_value, struct itimerval *old_value)
```

- `which` – which kind of timer
- `new_value` – pointer to `itimerval` structure with new timer details
- `old_value` – if not null, pointer to `itimerval` structure with previous timer details

```
struct itimerval {  
    struct timeval it_interval; /* Interval for periodic timer */  
    struct timeval it_value;   /* Time until next expiration */  
};
```

Returns zero on success.

## getpid

Get PID of current process.

```
pid_t getpid(void)
```

Returns the PID of the process.

## sendfile

Transfer data between two files or devices.

```
ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count)
```

- `out_fd` – file descriptor for destination
- `in_fd` – file descriptor for source
- `offset` – position to begin read
- `count` – bytes to copy

Returns bytes written.

## socket

Create an endpoint for network communication.

```
int socket(int domain, int type, int protocol)
```

- `domain` – flag specifying type of socket
- `type` – flag specifying socket specifics

- protocol – flag specifying protocol for communication

## domain flags

- AF\_UNIX – Local communication
- AF\_LOCAL – Same as AF\_UNIX
- AF\_INET – IPv4 Internet protocol
- AF\_AX25 – Amateur radio AX.25 protocol
- AF\_IPXIPX – Novell protocols
- AF\_APPLETALK – AppleTalk
- AF\_X25 – ITU-T X.25 / ISO-8208 protocol
- AF\_INET6 – IPv6 Internet protocol
- AF\_DECnet – DECnet protocol sockets
- AF\_KEY – IPsec management protocol
- AF\_NETLINK – Kernel user interface device
- AF\_PACKET – Low-level packet interface
- AF\_RDS – Reliable Datagram Sockets (RDS)
- AF\_PPPOX – Generic PPP transport layer for L2 tunnels (L2TP, PPPoE, etc.)
- AF\_LLC – Logical link control (IEEE 802.2 LLC)
- AF\_IB – InfiniBand native addressing
- AF MPLS – Multiprotocol Label Switching
- AF\_CAN – Controller Area Network automotive bus protocol
- AF\_TIPC – TIPC (cluster domain sockets)
- AF\_BLUETOOTH – Bluetooth low-level socket protocol
- AF\_ALG – Interface to kernel cryptography API

- AF\_VSOCK – VSOCK protocol for hypervisor-guest communication (VMWare, etc.)
- AF\_KCMKCM – Kernel connection multiplexor interface
- AF\_XDPXDP – Express data path interface

## type flags

- SOCK\_STREAM – sequenced, reliable byte streams
- SOCK\_DGRAM – datagrams (connectionless and unreliable messages, fixed maximum length)
- SOCK\_SEQPACKET – sequenced, reliable transmission for datagrams
- SOCK\_RAW – raw network protocol access
- SOCK\_RDM – reliable datagram layer with possible out-of-order transmission
- SOCK\_NONBLOCK – socket is non-blocking (avoid extra calls to fcntl)
- SOCK\_CLOEXEC – set close-on-exec flag

Returns file descriptor on success.

## connect

Connect to a socket.

```
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen)
```

- **sockfd** – socket file descriptor
- **addr** – pointer to socket address
- **addrlen** – size of address

Returns zero on success.

## accept

Accept connection on socket.

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)
```

- **sockfd** – socket file descriptor
- **addr** – pointer to socket address
- **addrlen** – size of address

Returns file descriptor of accepted socket on success.

## sendto

Send message on a socket.

```
send(int sockfd, const void *buf, size_t len, int flags)
```

- sockfd – socket file descriptor
- buf – buffer with message to send
- len – length of message
- flags – additional parameters

## flags

- MSG\_CONFIRM – informs link layer a reply has been received
- MSG\_DONTROUTE – do not use gateway in transmission of packet
- MSG\_DONTWAIT – perform non-blocking operation
- MSG\_EOR – end of record
- MSG\_MORE – more data to send
- MSG\_NOSIGNAL – do not generate SIGPIPE signal if peer closed connection
- MSG\_OOB – sends out-of-band data on supported sockets and protocols

## recvfrom

Receive message from socket.

```
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr
*src_addr, socklen_t *addrlen)
```

- sockfd – socket file descriptor
- buf – buffer to receive message
- size – size of buffer
- flags – additional parameters
- src\_addr – pointer to source address
- addrlen – length of source address.

## flags

- MSG\_CMSG\_CLOEXEC – set close-on-exec flag for socket file descriptor
- MSG\_DONTWAIT – perform operation in a non-blocking manner
- MSG\_ERRQUEUE – queued errors should be received in socket error queue

Returns bytes received successfully.

## sendmsg

Similar to the `sendto` syscall but allows sending additional data via the `msg` argument.

```
ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flags)
```

- `sockfd` – socket file descriptor
- `msg` – pointer to `msghdr` structure with message to send (with headers)
- `flags` – same as `sendto` syscall

```
struct msghdr {  
    void        *msg_name;      /* optional address */  
    socklen_t    msg_namelen;   /* address size */  
    struct iovec *msg iov;     /* scatter/gather array */  
    size_t       msg iovlen;    /* number of array elements in msg iov */  
    void        *msg control;   /* ancillary data */  
    size_t       msg controllen; /* ancillary data length */  
    int         msg flags;     /* flags on received message */  
};
```

## recvmsg

Receive message from socket.

```
ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags)
```

- sockfd – socket file descriptor
- msg – pointer to msghdr structure (defined in sendmsg above) to receive
- flags – define additional behavior (see sendto syscall)

## shutdown

Shut down full-duplex connection of a socket.

```
int shutdown(int sockfd, int how)
```

- sockfd – socket file descriptor
- how – flags defining additional behavior

Returns zero on success.

## how

- SHUT\_RD – prevent further receptions
- SHUT\_WR – prevent further transmissions
- SHUT\_RDWR – prevent further reception and transmission

## bind

Bind name to a socket.

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)
```

- **sockfd** – socket file descriptor
- **addr** – pointer to sockaddr structure with socket address
- **addrlen** – length of address

```
struct sockaddr {  
    sa_family_t sa_family;  
    char        sa_data[14];  
}
```

Returns zero on success.

## listen

Listen on a socket for connections.

```
int listen(int sockfd, int backlog)
```

- sockfd – socket file descriptor
- backlog – maximum length for pending connection queue

Returns zero on success.

## getsockname

Get socket name.

```
int getsockname(int sockfd, struct sockaddr *addr, socklen_t *addrlen)
```

- sockfd – socket file descriptor
- addr – pointer to buffer where socket name will be returned
- addrlen – length of buffer

Returns zero on success.

## getpeername

Get the name of the connected peer socket.

```
int getpeername(int sockfd, struct sockaddr *addr, socklen_t *addrlen)
```

- sockfd – socket file descriptor
- addr – pointer to buffer where peer name will be returned
- addrlen – length of buffer

Returns zero on success.

## socketpair

Create pair of sockets already connected.

```
int socketpair(int domain, int type, int protocol, int sv[2])
```

Arguments are identical to `socket` syscall except fourth argument (`sv`) is an integer array that is filled with the two socket descriptors.

Returns zero on success.

## setsockopt

Set options on a socket.

```
int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen)
```

- `sockfd` – socket file descriptor
- `optname` – option to set
- `optval` – pointer to the value of the option
- `optlen` – length of option

Returns zero on success.

## getsockopt

Get current options of a socket.

```
int getsockopt(int sockfd, int level, int optname, void *optval, socklen_t *optlen)
```

- **sockfd** – socket file descriptor
- **optname** – option to get
- **optval** – pointer to receive option value
- **optlen** – length of option

Returns zero on success.

## clone

Create child process.

```
int clone(int (*fn)(void *), void *stack, int flags, void *arg, ...  
/* pid_t *parent_tid, void *tls, pid_t *child_tid */)
```

- **fd** – pointer to initial execution address
- **stack** – pointer to child process's stack
- **flag** – define behavior of clone syscall
- **arg** – pointer to arguments for child process

## flags

- **CLONE\_CHILD\_CLEARTID** – clear id of child thread at location referenced by **child\_tid**
- **CLONE\_CHILD\_SETTID** – store id of child thread at location referenced by **child\_tid**
- **CLONE\_FILES** – parent and child process share same file descriptors

- CLONE\_FS – parent and child process share same filesystem information
- CLONE\_IO – child process shares I/O context with parent
- CLONE\_NEWCGROUP – child is created in new cgroup namespace
- CLONE\_NEWPIC – child process created in new IPC namespace
- CLONE\_NEWWNET – create child in new network namespace
- CLONE\_NEWNS – create child in new mount namespace
- CLONE\_NEWPID – create child in new PID namespace
- CLONE\_NEWUSER – create child in new user namespace
- CLONE\_NEWUTS – create child process in new UTS namespace
- CLONE\_PARENT – child is clone of the calling process
- CLONE\_PARENT\_SETTID – store id of child thread at location referenced by parent\_tid
- CLONE\_PID – child process is created with same PID as parent
- CLONE\_PIDFD – PID file descriptor of child process is placed in parent's memory
- CLONE\_PTRACE – if parent process is traced, trace child as well
- CLONE\_SETTLS – thread local storage (TLS) descriptor is set to TLS
- CLONE\_SIGHAND – parent and child share signal handlers
- CLONE\_SYSVSEM – child and parent share same System V semaphore adjustment values
- CLONE\_THREAD – child is created in same thread group as parent
- CLONE\_UNTRACED – if parent is traced, child is not traced
- CLONE\_VFORK – parent process is suspended until child calls execve or \_exit
- CLONE\_VM – parent and child run in same memory space

## fork

Create child process.

```
pid_t fork(void)
```

Returns PID of child process.

## vfork

Create child process without copying page tables of parent process.

```
pid_t vfork(void)
```

Returns PID of child process.

## execve

Execute a program.

```
int execve(const char *pathname, char *const argv[], char *const envp[])
```

- pathname – path to program to run
- argv – pointer to array of arguments for program
- envp – pointer to array of strings (in key=value format) for the environment

Does not return on success, returns -1 on error.

## exit

Terminate calling process.

```
void _exit(int status)
```

- status – status code to return to parent

Does not return a value.

## wait4

Wait for a process to change state.

```
pid_t wait4(pid_t pid, int *wstatus, int options, struct rusage *rusage)
```

- pid – PID of process
- wstatus – Status to wait for

- options – options flags for call
- rusage – pointer to structure with usage about child process filled on return

Returns PID of terminated child.

## options

- WNOHANG – return if no child exited
- WUNTRACED – return if child stops (but not traced with ptrace)
- WCONTINUED – return if stopped child resumed with SIGCONT
- WIFEXITED – return if child terminates normally
- WEXITSTATUS – return exit status of child
- WIFSIGNALED – return true if child was terminated with signal
- WTERMSIG – return number of signal that caused child to terminate
- WCOREDUMP – return true if child core dumped
- IFSTOPPED – return true if child was stopped by signal
- WSTOPSIG – returns signal number that caused child to stop
- WIFCONTINUED – return true if child was resumed with SIGCONT

## kill

Send a signal to process.

```
int kill(pid_t pid, int sig)
```

- pid – PID of process
- sig – number of signal to send to process

Return zero on success.

## getppid

Get PID of parent's calling process.

```
pid_t getppid(void)
```

Returns the PID of parent of calling process.

## uname

Get information about the kernel.

```
int uname(struct utsname *buf)
```

- buf – pointer to `utsname` structure to receive information

Return zero on success.

```
struct utsname {  
    char sysname[];      /* OS name (i.e. "Linux") */  
    char nodename[];     /* node name */  
    char release[];      /* OS release (i.e. "4.1.0") */  
    char version[];      /* OS version */  
    char machine[];      /* hardware identifier */  
#ifdef _GNU_SOURCE  
    char domainname[];   /* NIS or YP domain name */  
#endif  
};
```

## semget

Get System V semaphore set identifier.

```
int semget(key_t key, int nsems, int semflg)  
• key – key of identifier to retrieve  
• nsems – number of semaphores per set
```

- `semflg` – semaphore flags

Returns value of semaphore set identifier.

## semop

Perform operation on specified semampore(s).

```
int semop(int semid, struct sembuf *sops, size_t nsops)
```

- `semid` – id of semaphore
- `sops` – pointer to `sembuf` structure for operations
- `nsops` – number of operations

```
struct sembuf {  
    ushort  sem_num;          /* semaphore index in array */  
    short   sem_op;           /* semaphore operation */
```

```
short    sem_flg;           /* flags for operation */  
};
```

Return zero on success.

## semctl

Perform control operation on semaphore.

```
int semctl(int semid, int semnum, int cmd, ...)
```

- `semid` – semaphore set id
- `semnum` – number of semaphore in set
- `cmd` – operation to perform

Optional fourth argument is a `semun` structure:

```
union semun {  
    int             val;    /* SETVAL value */  
    struct semid_ds *buf;   /* buffer for IPC_STAT, IPC_SET */  
    unsigned short  *array; /* array for GETALL, SETALL */  
    struct seminfo   *__buf; /* buffer for IPC_INFO */  
};
```

Returns non-negative value corresponding to `cmd` flag on success, or -1 on error.

## cmd

- `IPC_STAT` – copy information from kernel associated with `semid` into `semid_ds` referenced by `arg.buf`
- `IPC_SET` – write values of `semid_ds` structure referenced by `arg.buf`
- `IPC_RMID` – remove semaphore set
- `IPC_INFO` – get information about system semaphore limits info `seminfo` structure
- `SEM_INFO` – return `seminfo` structure with same info as `IPC_INFO` except some fields are returned with info about resources consumed by semaphores
- `SEM_STAT` – return `semid_ds` structure like `IPC_STAT` but `semid` argument is index into kernel's semaphore array
- `SEM_STAT_ANY` – return `seminfo` structure with same info as `SEM_STAT` but `sem_perm.mode` isn't checked for read permission
- `GETALL` – return `semval` for all semaphores in set specified by `semid` into `arg.array`
- `GETNCNT` – return value of `semncnt` for the semaphore of the set indexed by `semnum`
- `GETPID` – return value of `sempid` for the semaphore of the set indexed by `semnum`

- GETVAL – return value of `semval` for the semaphore of the set indexed by `semnum`
- GETZCNT – return value of `semzcnt` for the semaphore of the set indexed by `semnum`
- SETALL – set `semval` for all the semaphores set using `arg.array`
- SETVAL – set value of `semval` to `arg.val` for the semaphore of the set indexed by `semnum`

## shmdt

Detach shared memory segment referenced by `shmaddr`.

```
int shmdt(const void *shmaddr)
```

- `shmaddr` – address of shared memory segment to detach

Return zero on success.

## msgget

Get System V message queue identifier.

```
int msgget(key_t key, int msgflg)
```

- `key` – message queue identifier
- `msgflg` – if `IPC_CREAT` and `IPC_EXCL` are specified and queue exists for key, then `msgget` fails with return error set to `EEXIST`

Return message queue identifier.

## msgsnd

Send a message to a System V message queue.

```
int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg)
```

- `msqid` – message queue id
- `msgp` – pointer to `msgbuf` Structure
- `msgsz` – size of `msgbuf` structure
- `msgflg` – flags defining specific behavior

```
struct msgbuf {  
    long mtype;          /* msg type, must be greater than zero */  
    char mtext[1];       /* msg text */  
};
```

Returns zero on success or otherwise modified by `msgflg`.

## msgflg

- IPC\_NOWAIT – return immediately if no message of requested type in queue
- MSG\_EXCEPT – use with `msgtyp > 0` to read first message in queue with type different from `msgtyp`
- MSG\_NOERROR – truncate message text if longer than `msgsiz` bytes

## msgrcv

Receive message from a system V message queue.

```
ssize_t msgrcv(int msqid, void *msgp, size_t msgsiz, long mmsgtyp, int msgflg)
```

- `msqid` – message queue id
- `msgp` – pointer to `msgbuf` structure
- `msgsiz` – size of `msgbuf` structure
- `mmsgtyp` – read first msg if 0, read first msg of `mmsgtyp` if > 0, or if negative, read first msg in queue with type less or equal to absolute value of `mmsgtyp`
- `msgflg` – flags defining specific behavior

```
struct msgbuf {  
    long mtype;          /* msg type, must be greater than zero */  
    char mtext[1];       /* msg text */  
};
```

Returns zero on success or otherwise modified by `msgflg`.

## msgctl

System V message control.

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf)
```

- `msqid` – message queue id
- `cmd` – command to execute
- `buf` – pointer to buffer styled in `msqid_ds`

```
struct msqid_ds {  
    struct ipc_perm msg_perm; /* ownership/permissions */  
    time_t        msg_stime; /* last msgsnd(2) time */  
    time_t        msg_rtime; /* last msgrecv(2) time */  
    time_t        msg_ctime; /* last change time */  
    unsigned long __msg_cbytes; /* bytes in queue */  
    msgqnum_t     msg_qnum; /* messages in queue */  
    msglen_t      msg_qbytes; /* max bytes allowed in queue */  
    pid_t         msg_lspid; /* PID of last msgsnd(2) */  
    pid_t         msg_lrpid; /* PID of last msgrecv(2) */  
};
```

```
struct msginfo {  
    int msgpool; /* kb of buffer pool used */  
    int msgmap; /* max # of entries in message map */  
    int msgmax; /* max # of bytes per single message */  
    int msgmnb; /* max # of bytes in the queue */  
    int msgmni; /* max # of message queues */  
    int msgssz; /* message segment size */  
    int msqtql; /* max # of messages on queues */  
    unsigned short int msgseg; /* max # of segments unused in kernel */  
};
```

Returns zero on success or modified return value based on cmd.

## cmd

- IPC\_STAT – copy data structure from kernel by `msqid` into `msqid_ds` structure referenced by `buf`
- IPC\_SET – update `msqid_ds` structure referenced by `buf` to kernel, updating its `msg_ctime`
- IPC\_RMID – remove message queue
- IPC\_INFO – returns information about message queue limits into `msginfo` structure referenced by `buf`
- MSG\_INFO – same as IPC\_INFO except `msginfo` structure is filled with usage vs. max usage statistics
- MSG\_STAT – same as IPC\_STAT except `msqid` is a pointer into kernel's internal array

## fcntl

Manipulate a file descriptor.

```
int fcntl(int fd, int cmd, ... /* arg */ )
```

- `fd` – file descriptor
- `cmd` – cmd flag
- `/* arg */` – additional parameters based on `cmd`

Return value varies based on `cmd` flags.

## cmd

Parameters in () is the optional /\* arg \*/ with specified type.

- `F_DUPFD` – find lowest numbered file descriptor greater or equal to (`int`) and duplicate it, returning new file descriptor
- `F_DUPFD_CLOEXEC` – same as `F_DUPFD` but sets close-on-exec flag
- `F_GETFD` – return file descriptor flags
- `F_SETFD` – set file descriptor flags based on (`int`)
- `F_GETFL` – get file access mode
- `F_SETFL` – set file access mode based on (`int`)
- `F_GETLK` – get record locks on file (pointer to `struct flock`)
- `F_SETLK` – set lock on file (pointer to `struct flock`)
- `F_SETLKW` – set lock on file with wait (pointer to `struct flock`)

- F\_GETOWN – return process id receiving SIGIO and SIGURG
- F\_SETOWN – set process id to receive SIGIO and SIGURG (int)
- F\_GETOWN\_EX – return file descriptor owner settings (struct f\_owner\_ex \*)
- F\_SETOWN\_EX – direct IO signals on file descriptor (struct f\_owner\_ex \*)
- F\_GETSIG – return signal sent when IO is available
- F\_SETSIG – set signal sent when IO is available (int)
- F\_SETLEASE – obtain lease on file descriptor (int), where arg is F\_RDLCK, F\_WRLCK, and F\_UNLCK
- F\_GETLEASE – get current lease on file descriptor (F\_RDLCK, F\_WRLCK, or F\_UNLCK are returned)
- F\_NOTIFY – notify when dir referenced by file descriptor changes (int) (DN\_ACCESS, DN\_MODIFY, DN\_CREATE, DN\_DELETE, DN\_RENAME, DN\_ATTRIB are returned)
- F\_SETPipe\_SZ – change size of pipe referenced by file descriptor to (int) bytes
- F\_GETPIPE\_SZ – get size of pipe referenced by file descriptor

## flock

```
struct flock {  
    ...  
    short l_type;      /* lock type: F_RDLCK, F_WRLCK, or F_UNLCK */  
    short l_whence;    /* interpret l_start with SEEK_SET, SEEK_CUR, or SEEK_END */  
    off_t l_start;     /* offset for lock */  
    off_t l_len;       /* bytes to lock */  
    pid_t l_pid;       /* PID of blocking process (F_GETLK only) */  
    ...  
};
```

## f\_owner\_ex

```
struct f_owner_ex {  
    int type;  
    pid_t pid;  
};
```

## flock

Apply or remove advisory lock on open file

```
int flock(int fd, int operation)  
• fd – file descriptor  
• operation – operaton flag
```

Returns zero on success.

## operation

- `LOCK_SH` – place shared lock
- `LOCK_EX` – place exclusive lock
- `LOCK_UN` – remove existing lock

## fsync

Sync file's data and metadata in memory to disk, flushing all write buffers and completes pending I/O.

```
int fsync(int fd)
  • fd – file descriptor
```

Returns zero on success.

## fdatasync

Sync file's data (but not metadata, unless needed) to disk.

```
int fdatasync(int fd)
  • fd – file descriptor
```

Returns zero on success.

## truncate

Truncate file to a certain length.

```
int truncate(const char *path, off_t length)
```

- path – pointer to path of file
- length – length to truncate to

Returns zero on success.

## ftruncate

Truncate file descriptor to a certain length.

```
int ftruncate(int fd, off_t length)
```

- fd – file descriptor
- length – length to truncate to

Returns zero on success.

## getdents

Get directory entries from a specified file descriptor.

```
int getdents(unsigned int fd, struct linux_dirent *dirp, unsigned int count)
```

- fd – file descriptor of directory
- dirp – pointer to `linux_dirent` structure to receive return values
- count – size of `dirp` buffer

Returns bytes read on success.

```
struct linux_dirent {  
    unsigned long d_ino;      /* number of inode */  
    unsigned long d_off;      /* offset to next linux_dirent */  
    unsigned short d_reclen;  /* length of this linux_dirent */  
    char          d_name[];   /* filename (null terminated) */  
    char          pad;        /* padding byte */  
    char          d_type;     /* type of file (see types below) */  
};
```

## types

- DT\_BLK – block device
- DT\_CHR – char device
- DT\_DIR – directory
- DT\_FIFO – FIFO named pipe
- DT\_LNK – symlink
- DT\_REG – regular file
- DT\_SOCK – UNIX socket
- DT\_UNKNOWN – unknown

## getcwd

Get current working directory

```
char *getcwd(char *buf, size_t size)
```

- buf – pointer to buffer to receive path
- size – size of buf

Returns pointer to string containing current working directory.

## chdir

Change the current directory.

```
int chdir(const char *path)
• path – pointer to string with name of path
```

Returns zero on success.

## fchdir

Change to the current directory specified by supplied file descriptor.

```
int fchdir(int fd)
• fd – file descriptor
```

Returns zero on success.

## rename

Rename or move a file.

```
int rename(const char *oldpath, const char *newpath)
• oldpath – pointer to string with old path/name
• newpath – pointer to string with new path/name
```

Returns zero on success.

## mkdir

Make a directory.

```
int mkdir(const char *pathname, mode_t mode)
```

- pathname – pointer to string with directory name
- mode – file system permissions mode

Returns zero on success.

## rmdir

Remove a directory.

```
int rmdir(const char *pathname)
```

- pathname – pointer to string with directory name

Returns zero on success.

## creat

Create a file or device.

```
int creat(const char *pathname, mode_t mode)
```

- pathname – pointer to string with file or device name
- mode – file system permissions mode

Returns a file descriptor on success.

## link

Creates a hard link for a file.

```
int link(const char *oldpath, const char *newpath)
```

- oldpath – pointer to string with old filename
- newpath – pointer to string with new filename

Returns zero on success.

## unlink

Remove a file.

```
int unlink(const char *pathname)
```

- pathname – pointer to string with path name

Return zero on success.

## symlink

Create a symlink.

```
int symlink(const char *oldpath, const char *newpath)
```

- oldpath – pointer to string with old path name
- newpath – pointer to string with new path name

Return zero on success.

## readlink

Return name of a symbolic link.

```
ssize_t readlink(const char *path, char *buf, size_t bufsiz)
```

- `path` – pointer to string with symlink name
- `buf` – pointer to buffer with result
- `bufsiz` – size of buffer for result

Returns number of bytes placed in `buf`.

## chmod

Set permission on a file or device.

```
int chmod(const char *path, mode_t mode)
```

- `path` – pointer to string with name of file or device
- `mode` – new permissions mode

Returns zero on success.

## fchmod

Same as `chmod` but sets permissions on file or device referenced by file descriptor.

```
int fchmod(int fd, mode_t mode)
```

- `fd` – file descriptor
- `mode` – new permissions mode

Returns zero on success.

## chown

Change owner of file or device.

```
int chown(const char *path, uid_t owner, gid_t group)
• path – pointer to string with name of file or device
• owner – new owner of file or device
• group – new group of file or device
```

Returns zero on success.

## fchown

Same as `chown` but sets owner and group on a file or device referenced by file descriptor.

```
int fchown(int fd, uid_t owner, gid_t group)
• fd – file descriptor
• owner – new owner
• group – new group
```

Returns zero on success.

## lchown

Same as `chown` but doesn't reference symlinks.

```
int lchown(const char *path, uid_t owner, gid_t group)
• path – pointer to string with name of file or device
• owner – new owner
• group – new group
```

Returns zero on success.

## umask

Sets the mask used to create new files.

```
mode_t umask(mode_t mask)
• mask – mask for new files
```

System call will always succeed and returns previous mask.

## gettimeofday

```
int gettimeofday(struct timeval *tv, struct timezone *tz)
• tv – pointer to timeval structure to retrieve time
• tz – pointer to timezone structure to receive time zone
```

```
struct timeval {  
    time_t      tv_sec;      /* seconds */  
    suseconds_t tv_usec;    /* microseconds */  
};  
  
struct timezone {  
    int tz_minuteswest;    /* minutes west of GMT */  
    int tz_dsttime;        /* DST correction type */  
};
```

Returns zero on success.

## getrlimit

Get current resource limits.

```
int getrlimit(int resource, struct rlimit *rlim)
```

- **resource** – resource flag
- **rlim** – pointer to rlimit structure

```
struct rlimit {  
    rlim_t rlim_cur;  /* soft limit */  
    rlim_t rlim_max; /* hard limit */  
};
```

Returns zero on success and fills `rlim` structure with results.

## resource flags

- `RLIMIT_AS` – max size of process virtual memory
- `RLIMIT_CORE` – max size of core file
- `RLIMIT_CPU` – max CPU time, in seconds
- `RLIMIT_DATA` – max size of process's data segment
- `RLIMIT_FSIZE` – max size of files that process is allowed to create
- `RLIMIT_LOCKS` – max `flock` and `fcntl` leases allowed
- `RLIMIT_MEMLOCK` – max bytes of RAM allowed to be locked
- `RLIMIT_MSGQUEUE` – max size of POSIX message queues
- `RLIMIT_NICE` – max nice value
- `RLIMIT_NOFILE` – max number of files allowed to be opened plus one
- `RLIMIT_NPROC` – max number of processes or threads
- `RLIMIT_RSS` – max resident set pages
- `RLIMIT_RTPRIO` – real-time priority ceiling
- `RLIMIT_RTTIME` – limit in microseconds of real-time CPU scheduling
- `RLIMIT_SIGPENDING` – max number of queued signals
- `RLIMIT_STACK` – max size of process stack

## getrusage

Obtain resource usage.

```
int getrusage(int who, struct rusage *usage)
```

- who – target flag
- usage – pointer to rusage structure

```
struct rusage {  
    struct timeval ru_utime; /* used user CPU time */  
    struct timeval ru_stime; /* used system CPU time */  
    long    ru_maxrss;      /* maximum RSS */  
    long    ru_ixrss;       /* shared memory size */  
    long    ru_idrss;       /* unshared data size */  
    long    ru_isrss;       /* unshared stack size */  
    long    ru_minflt;      /* soft page faults */  
    long    ru_majflt;      /* hard page faults */  
    long    ru_nswap;        /* swaps */  
    long    ru_inblock;      /* block input operations */  
    long    ru_oublock;      /* block output operations */  
    long    ru_msgrnd;      /* sent # of IPC messages */  
    long    ru_msgrcv;      /* received # IPC messages */  
    long    ru_nssignals;    /* number of signals received */  
    long    ru_nvcs;         /* voluntary context switches */  
    long    ru_nivcs;        /* involuntary context switches */  
};
```

Returns zero on success.

## who target

- RUSAGE\_SELF – get usage statistics for calling process
- RUSAGE\_CHILDREN – get usage statistics for all children of calling process

- RUSAGE\_THREAD – get usage statistics for calling thread

## sysinfo

Return information about the system.

```
int sysinfo(struct sysinfo *info)
```

- info – pointer to sysinfo structure

```
struct sysinfo {  
    long uptime;          /* seconds since boot */  
    unsigned long loads[3]; /* 1/5/15 minute load avg */  
    unsigned long totalram; /* total usable memory size */  
    unsigned long freeram; /* available memory */  
    unsigned long sharedram; /* shared memory amount */  
    unsigned long bufferram; /* buffer memory usage */  
    unsigned long totalswap; /* swap space size */  
    unsigned long freeswap; /* swap space available */  
    unsigned short procs; /* total number of current processes */  
    unsigned long totalhigh; /* total high memory size */  
    unsigned long freehigh; /* available high memory size */  
    unsigned int mem_unit; /* memory unit size in bytes */  
    char _f[20-2*sizeof(long)-sizeof(int)]; /* padding to 64 bytes */  
};
```

Returns zero on success and places system information in sysinfo structure.

## times

Get process times.

```
clock_t times(struct tms *buf)
• buf – pointer to tms structure
```

```
struct tms {
    clock_t tms_utime; /* user time */
    clock_t tms_stime; /* system time */
    clock_t tms_cutime; /* children user time */
    clock_t tms_cstime; /* children system time */
};
```

Returns clock ticks since arbitrary point in past and may overflow. tms structure is filled with values.

## ptrace

## Trace a process.

```
long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data)
```

- `request` – determine type of trace to perform
- `pid` – process id to trace
- `addr` – pointer to buffer for certain response values
- `data` – pointer to buffer used in certain types of traces

Returns zero on request, placing trace data into `addr` and/or `data`, depending on trace details in request flags.

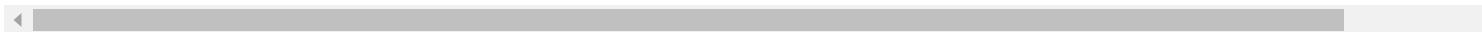
## request flags

- `PTRACE_TRACEME` – indicate process traced by parent
- `PTRACE_PEEKTEXT` and `PTRACE_PEEKDATA` – read word at `addr` and return as result of call
- `PTRACE_PEEKUSER` – read word at `addr` in `USER` area of the traced process's memory

- `PTRACE_POKETEXT` and `PTRACE_POKEDATA` – copy data into `addr` in traced process's memory
- `PTRACE_POKEUSER` – copy data into `addr` in the traced process's `USER` area in memory
- `PTRACE_GETREGS` – copy traced program's general registers into `data`
- `PTRACE_GETFPREGS` – copy traced program's floating-point registers into `data`
- `PTRACE_GETREGSET` – read traced program's registers in architecture-agnostic way
- `PTRACE_SETREGS` – modify traced program's general registers
- `PTRACE_SETFPREGS` – modify traced program's floating-point registers
- `PTRACE_SETREGSET` – modify traced program's registers (architecture-agnostic)
- `PTRACE_GETSIGINFO` – get info about signal that caused stop into `siginfo_t` structure
- `PTRACE_SETSIGINFO` – set signal info by copying `siginfo_t` structure from `data` into traced program
- `PTRACE_PEEKSIGINFO` – get `siginfo_t` structures without removing queued signals
- `PTRACE_GETSIGMASK` – copy mask of blocked signals into `data` which will be a `sigset_t` structure
- `PTRACE_SETSIGMASK` – change blocked signals mask to value in `data` which should be a `sigset_t` structure
- `PTRACE_SETOPTIONS` – set options from `data`, where `data` is a bit mask of the following options:
  - `PTRACE_O_EXITKILL` – send `SIGKILL` to traced program if tracing program exists
  - `PTRACE_O_TRACECLONE` – stop traced program at next `clone` syscall and start tracing new process
  - `PTRACE_O_TRACEEXEC` – stop traced program at next `execve` syscall
  - `PTRACE_O_TRACEEXIT` – stop the traced program at exit
  - `PTRACE_O_TRACEFORK` – stop traced program at next `fork` and start tracing forked process
  - `PTRACE_O_TRACESYSGOOD` – set bit 7 in signal number (`SIGTRAP|0x80`) when sending system call traps
  - `PTRACE_O_TRACEVFORK` – stop traced program at next `vfork` and start tracing new process

- `PTRACE_O_TRACEVFORKDONE` – stop traced program after next `vfork`
- `PTRACE_O_TRACESECCOMP` – stop traced program when `seccomp` rule is triggered
- `PTRACE_O_SUSPEND_SECCOMP` – suspend traced program's seccomp protections
- `PTRACE_GETEVENTMSG` – get message about most recent `ptrace` event and put in `data` of tracing program
- `PTRACE_CONT` – restart traced process that was stopped and if `data` is not zero, send number of signal to it
- `PTRACE_SYSCALL` and `PTRACE_SIGNALSTEP` – restart traced process that was stopped but stop at entry or exit of next syscall
- `PTRACE_SYSEMU` – continue, then stop on entry for next syscall (but don't execute it)
- `PTRACE_SYSEMU_SINGLESTEP` – same as `PTRACE_SYSEMU` but single step if instruction isn't a syscall
- `PTRACE_LISTEN` – restart traced program but prevent from executing (similar to `SIGSTOP`)
- `PTRACE_INTERRUPT` – stop the traced program
- `PTRACE_ATTACH` – attach to process `pid`
- `PTRACE_SEIZE` attach to process `pid` but do not stop process
- `PTRACE_SECCOMP_GET_FILTER` – allows for dump of traced program's classic BPF filters, where `addr` is the index of filter and `data` is pointer to structure `sock_filter`
- `PTRACE_DETACH` – detach then restart stopped traced program
- `PTRACE_GET_THREAD_AREA` – reads TLS entry into GDT with index specified by `addr`, placing copy `struct user_desc` at `data`
- `PTRACE_SET_THREAD_AREA` – sets TLS entry into GTD with index specified by `addr`, assigning it `struct user_desc` at `data`
- `PTRACE_GET_SYSCALL_INFO` – get information about syscall that caused stop and place struct `ptrace_syscall_info` into `data`, where `addr` is size of buffer

```
struct ptrace_peeksiginfo_args {  
    u64 off;      /* queue position to start copying signals */  
    u32 flags;    /* PTRACE_PEEKSIGINFO_SHARED or 0 */  
    s32 nr;       /* # of signals to copy */  
};  
  
struct ptrace_syscall_info {  
    __u8 op;          /* type of syscall stop */  
    __u32 arch;        /* AUDIT_ARCH_* value */  
    __u64 instruction_pointer; /* CPU instruction pointer */  
    __u64 stack_pointer; /* CPU stack pointer */  
    union {  
        struct {  
            __u64 nr;           /* syscall number */  
            __u64 args[6];      /* syscall arguments */  
        } entry;  
        struct {  
            __s64 rval;         /* syscall return value */  
            __u8 is_error;     /* syscall error flag */  
        } exit;  
        struct {  
            __u64 nr;           /* syscall number */  
            __u64 args[6];      /* syscall arguments */  
            __u32 ret_data;     /* SECCOMP_RET_DATA part of SECCOMP_RET_TRACE return val */  
        } seccomp;  
    };  
};
```



## getuid

Get UID of calling process.

```
uid_t getuid(void)
```

Returns the UID. Always succeeds.

## syslog

Read or clear kernel message buffer.

```
int syslog(int type, char *bufp, int len)
```

- **type** – function to perform
- **bufp** – pointer to buffer (used for reading)
- **len** – length of buffer

Returns bytes read, available to read, total size of kernel buffer, or 0, depending on type flag.

## type flag

- SYSLOG\_ACTION\_READ – read `len` bytes of kernel message log into `bufp`, returns number of bytes read
- SYSLOG\_ACTION\_READ\_ALL – read entire kernel message log into `bufp`, reading last `len` bytes from kernel, returning bytes read
- SYSLOG\_ACTION\_READ\_CLEAR – read, then clear kernel message log into `bufp`, up to `len` bytes, returning bytes read
- SYSLOG\_ACTION\_CLEAR – clear the kernel message log buffer, returns zero on success
- SYSLOG\_ACTION\_CONSOLE\_OFF – prevents kernel messages being sent to the console
- SYSLOG\_ACTION\_CONSOLE\_ON – enables kernel messages being sent to the console
- SYSLOG\_ACTION\_CONSOLE\_LEVEL – sets the log level of messages (values 1 to 8 via `len`) to allow message filtering
- SYSLOG\_ACTION\_SIZE\_UNREAD – returns number of bytes available for reading in kernel message log
- SYSLOG\_ACTION\_SIZE\_BUFFER – returns size of kernel message buffer

## getgid

Get GID of calling process.

```
gid_t getgid(void)
```

Returns the GID. Always succeeds.

## setuid

Set UID of calling process.

```
int setuid(uid_t uid)
```

- uid – new UID

Returns zero on success.

## setgid

Set GID of calling process.

```
int setgid(gid_t gid)
```

- gid – new GID

Returns zero on success.

## geteuid

Get effective UID of calling process.

```
uid_t geteuid(void)
```

Returns the effective UID. Always succeeds.

## getegid

Get effective GID of calling process.

```
gid_t getegid(void)
```

Returns the effective GID. Always succeeds.

## setpgid

Set process group ID of a process.

```
int setpgid(pid_t pid, pid_t pgid)
```

- pid – process ID
- pgid – process group ID

Returns zero on success.

## getppid

Get process group ID of a process.

```
pid_t getpgid(pid_t pid)
```

- pid – process ID

Returns process group ID.

## getpgrp

Get process group ID of calling process.

```
pid_t getpgrp(void)
```

Return process group ID.

## setsid

Create session if calling process isn't leader of a process group.

```
pid_t setsid(void)
```

Returns created session ID.

## setreuid

Set both real and effective UID for calling process.

```
int setreuid(uid_t ruid, uid_t euid)
```

- **ruid** – the real UID
- **euid** – the effective UID

Returns zero on success.

## setregid

Set both real and effective GID for calling process.

```
int setregid(gid_t rgid, gid_t egid)
```

- `rgid` – the real GID
- `egid` – the effective GID

Returns zero on success.

## getgroups

Get a list of supplementary group IDs for calling process.

```
int getgroups(int size, gid_t list[])
```

- `size` – size of array `list`
- `list` – array of `gid_t` to retrieve list

Returns number of supplementary group IDs retrieved into `list`.

## setgroups

Set list of supplementary group IDs for calling process.

```
int setgroups(size_t size, const gid_t *list)
```

- `size` – size of array `list`
- `list` – array of `gid_t` to set list

Returns zero on success.

## setresuid

Sets real, effective, and saved UID.

```
int setresuid(uid_t ruid, uid_t euid, uid_t suid)
```

- `ruid` – the real UID
- `euid` – the effective UID
- `suid` – the saved UID

Returns zero on success.

## setresgid

Sets real, effective, and saved GID.

```
int setresgid(gid_t rgid, gid_t egid, gid_t sgid)
• rgid – the real GID
• egid – the effective GID
• sgid – the saved GID
```

Returns zero on success.

## getresuid

Get the real, effective, and saved UID.

```
int getresuid(uid_t *ruid, uid_t *euid, uid_t *suid)
• ruid – the real UID
• euid – the effective UID
• suid – the saved UID
```

Returns zero on success.

## getresgid

Get the real, effective, and saved GID.

```
int getresuid(gid_t *rgid, gid_t *egid, gid_t *sgid)
```

- `rgid` – the real GID
- `egid` – the effective GID
- `sgid` – the saved GID

Returns zero on success.

## getpgid

Get process group ID of a process.

```
pid_t getpgid(pid_t pid)
```

- `pid` – process ID

Returns process group ID.

## setfsuid

Set UID for filesystem checks.

```
int setfsuid(uid_t fsuid)
```

Always returns previous filesystem UID.

## setfsgid

Set GID for filesystem checks.

```
int setfsgid(uid_t fsgid)
```

Always returns previous filesystem GID.

## getsid

Get session ID.

```
pid_t getsid(pid_t pid)
```

Returns session ID.

## capget

Get capabilities of a thread.

```
int capget(cap_user_header_t hdrp, cap_user_data_t datap)
```

- `hdrp` – capability header structure
- `datap` – capability data structure

```
typedef struct __user_cap_header_struct {  
    __u32 version;  
    int pid;  
} *cap_user_header_t;
```

```
typedef struct __user_cap_data_struct {  
    __u32 effective;  
    __u32 permitted;  
    __u32 inheritable;  
} *cap_user_data_t;
```

Returns zero on success.

## capset

Set capabilities of a thread.

```
int capset(cap_user_header_t hdrp, const cap_user_data_t datap)
```

- `hdrp` – capability header structure
- `datap` – capability data structure

```
typedef struct __user_cap_header_struct {  
    __u32 version;  
    int pid;  
} *cap_user_header_t;
```

```
typedef struct __user_cap_data_struct {  
    __u32 effective;  
    __u32 permitted;  
    __u32 inheritable;  
} *cap_user_data_t;
```

Returns zero on success.

## rt\_sigpending

Return signal set that are pending delivery to calling process or thread.

```
int sigpending(sigset_t *set)
```

- `set` – pointer to `sigset_t` structure to retrieve mask of signals.

## rt\_sigtimedwait

Suspend execution (until `timeout`) of calling process or thread until a signal referenced in `set` is pending.

```
int sigtimedwait(const sigset_t *set, siginfo_t *info, const struct timespec *timeout)
```

- `set` – pointer to `sigset_t` structure to define signals to wait for
- `info` – if not null, pointer to `siginfo_t` structure with info about signal

- `timeout` – a `timespec` structure setting a maximum time to wait before resuming execution

```
struct timespec {  
    long    tv_sec;          /* time in seconds */  
    long    tv_nsec;         /* time in nanoseconds */  
}
```

## rt\_sigqueueinfo

Queue a signal.

```
int rt_sigqueueinfo(pid_t tgid, int sig, siginfo_t *info)
```

- `tgid` – thread group id
- `sig` – signal to send
- `info` – pointer to structure `siginfo_t`

Returns zero on success.

## rt\_sigsuspend

Wait for a signal.

```
int sigsuspend(const sigset_t *mask)
```

- `mask` – pointer to `sigset_t` structure (defined in `sigaction`)

Always returns with -1.

## sigaltstack

Set/get signal stack context.

```
int sigaltstack(const stack_t *ss, stack_t *oss)
```

- ss – pointer to stack\_t structure representing new signal stack
- oss – pointer to stack\_t structure used for getting information on current signal stack

```
typedef struct {  
    void  *ss_sp;      /* stack base address */  
    int   ss_flags;   /* flags */  
    size_t ss_size;   /* bytes in stack */  
} stack_t;
```

Returns zero on success.

## utime

Change the last access and modification time of a file.

```
int utime(const char *filename, const struct utimbuf *times)
```

- `filename` – pointer to string with filename
- `times` – pointer to structure `utimbuf` structure

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

Returns zero on success.

## mknod

Create a special file (usually used for device files).

```
int mknod(const char *pathname, mode_t mode, dev_t dev)
```

- `pathname` – pointer to string with full path of file to create
- `mode` – permissions and type of file
- `dev` – device number

Returns zero on success.

## uselib

Load a shared library.

```
int uselib(const char *library)
```

- library – pointer to string with full path of library file

Return zero on success.

## personality

Set process execution domain (personality)

```
int personality(unsigned long persona)
```

- persona – domain of persona

Returns previous persona on success unless persona is set to 0xFFFFFFFF.

## ustat

Get filesystem statistics

```
int ustat(dev_t dev, struct ustat *ubuf)
• dev – number of device with mounted filesystem
• ubuf – pointer to ustat structure for return values
```

```
struct ustat {
    daddr_t f_tfree;      /* free blocks */
    ino_t   f_tinode;     /* free inodes */
    char    f_fname[6];   /* filesystem name */
    char    f_fpack[6];   /* filesystem pack name */
};
```

Returns zero on success and ustat structure referenced by ubuf is filled with statistics.

## statfs

Get filesystem statistics.

```
int statfs(const char *path, struct statfs *buf)
```

- path – pointer to string with filename of any file on the mounted filesystem
- buf – pointer to statfs structure

```
struct statfs {  
    __SWORD_TYPE    f_type;      /* filesystem type */  
    __SWORD_TYPE    f_bsize;     /* optimal transfer block size */  
    fsblkcnt_t      f_blocks;    /* total blocks */  
    fsblkcnt_t      f_bfree;     /* free blocks */  
    fsblkcnt_t      f_bavail;    /* free blocks available to unprivileged user */  
    fsfilcnt_t      f_files;     /* total file nodes */  
    fsfilcnt_t      f_ffree;     /* free file nodes */  
    fsid_t          f_fsid;      /* filesystem id */  
    __SWORD_TYPE    f_namelen;   /* maximum length of filenames */  
    __SWORD_TYPE    f_frsize;    /* fragment size */  
    __SWORD_TYPE    f_spare[5];  
};
```

Returns zero on success.

## fstatfs

Works just like statfs except provides filesystem statistics on via a file descriptor.

```
int fstatfs(int fd, struct statfs *buf)
```

- fd – file descriptor
- buf – pointer to statfs structure

Returns zero on success.

## sysfs

Get filesystem type information.

```
int sysfs(int option, const char *fsname)
int sysfs(int option, unsigned int fs_index, char *buf)
int sysfs(int option)
```

- option – when set to 3, return number of filesystem types in kernel, or can be 1 or 2 as indicated below

- `fsname` – pointer to string with name of filesystem (set option to 1)
- `fs_index` – index into null-terminated filesystem identifier string written to buffer at `buf` (set option to 2)
- `buf` – pointer to buffer

Returns filesystem index when `option` is 1, zero for 2, and number of filesystem types in kernel for 3.

## getpriority

Get priority of a process.

```
int getpriority(int which, int who)
```

- `which` – flag determining which priority to get
- `who` – PID of process

Returns priority of specified process.

## which

- PRIO\_PROCESS – process
  - \* PRIO\_PGRP – process group
- PRIO\_USER – user ID

## setpriority

Set priority of a process.

```
int setpriority(int which, int who, int prio)
• which – flag determining which priority to set
• who – PID of process
• prio – priority value (-20 to 19)
```

Returns zero on success.

## **sched\_setparam**

Set scheduling parameters of a process.

```
int sched_setparam(pid_t pid, const struct sched_param *param)
```

- pid – PID of process
- param – pointer to sched\_param structure

Returns zero on success.

## **sched\_getparam**

```
int sched_getparam(pid_t pid, struct sched_param *param)
```

- pid – PID of process
- param – pointer to sched\_param structure

Returns zero on success.

## **sched\_setscheduler**

Set scheduling parameters for a process.

```
int sched_setscheduler(pid_t pid, int policy, const struct sched_param *param)
```

- pid – PID of process
- policy – policy flag
- param – pointer to sched\_param structure

Returns zero on success.

## policy

- SCHED\_OTHER – standard round-robin time sharing policy
- SCHED\_FIFO – first-in-first-out scheduling policy
- SCHED\_BATCH – executes processes in a batch-style schedule
- SCHED\_IDLE – denotes a process be set for low priority (background)

## sched\_getscheduler

Get scheduling parameters for a process.

```
int sched_getscheduler(pid_t pid)
    • pid – PID of process
```

Returns policy flag (see sched\_setscheduler).

## **sched\_get\_priority\_max**

Get static priority maximum.

```
int sched_get_priority_max(int policy)
• policy – policy flag (see sched_setscheduler)
```

Returns maximum priority value for provided policy.

## **sched\_get\_priority\_min**

Get static priority minimum.

```
int sched_get_priority_min(int policy)
• policy – policy flag (see sched_setscheduler)
```

Returns minimum priority value for provided policy.

## sched\_rr\_get\_interval

Get SCHED\_RR interval for a process.

```
int sched_rr_get_interval(pid_t pid, struct timespec *tp)
```

- pid – PID of process
- tp – pointer to timespec structure

Returns zero on success and fills tp with intervals for pid if SCHED\_RR is the scheduling policy.

## mlock

Lock all or part of calling process's memory.

```
int mlock(const void *addr, size_t len)
```

- addr – pointer to start of address space
- len – length of address space to lock

Returns zero on success.

## munlock

Unlock all or part of calling process's memory.

```
int munlock(const void *addr, size_t len)
• addr – pointer to start of address space
• len – length of address space to unlock
```

Returns zero on success.

## mlockall

Lock all address space of calling process's memory.

```
int mlockall(int flags)
• flags – flags defining additional behavior
```

## flags

- MCL\_CURRENT – lock all pages as of time of calling this syscall
- MCL\_FUTURE – lock all pages that are mapped to this process in the future
- MCL\_ONFAULT – mark all current (or future, along with MCL\_FUTURE) when they are page faulted

## munlockall

Unlock all address space of calling process's memory.

```
int munlockall(void)
```

Returns zero on success.

## vhangup

Send a "hangup" signal to the current terminal.

```
int vhangup(void)
```

Returns zero on success.

## modify\_ldt

Read or write to the local descriptor table for a process

```
int modify_ldt(int func, void *ptr, unsigned long bytecount)
```

- func – 0 for read, 1 for write
- ptr – pointer to LDT
- bytecount – bytes to read, or for write, size of user\_desc structure

```
struct user_desc {  
    unsigned int entry_number;  
    unsigned int base_addr;  
    unsigned int limit;  
    unsigned int seg_32bit:1;  
    unsigned int contents:2;  
    unsigned int read_exec_only:1;  
    unsigned int limit_in_pages:1;  
    unsigned int seg_not_present:1;  
    unsigned int useable:1;  
};
```

Returns bytes read or zero for success when writing.

## pivot\_root

Change root mount.

```
int pivot_root(const char *new_root, const char *put_old)
• new_root – pointer to string with path to new mount
• put_old – pointer to string with path for old mount
```

Returns zero on success.

## prctl

```
int prctl(int option, unsigned long arg2, unsigned long arg3, unsigned long arg4,
unsigned long arg5)
```

- option – specify operation flag
- arg2, arg3, arg4, and arg5 – variables used depending on option, see option flags

## option

- PR\_CAP\_AMBIENT – read/change ambient capability of calling thread referencing value in arg2, in regards to:
  - PR\_CAP\_AMBIENT\_RAISE – capability in arg3 is added to ambient set
  - PR\_CAP\_AMBIENT\_LOWER – capability in arg3 is removed from ambient set
  - PR\_CAP\_AMBIENT\_IS\_SET – returns 1 if capability in arg3 is in the ambient set, 0 if not
  - PR\_CAP\_AMBIENT\_CLEAR\_ALL – remove all capabilities from ambient set, set arg3 to 0
- PR\_CAPBSET\_READ – return 1 if capability specified in arg2 is in calling thread's capability bounding set, 0 if not
- PR\_CAPBSET\_DROP – if calling thread has CAP\_SETPCAP capability in user namespace, drop capability in arg2 from capability bounding set for calling process
- PR\_SET\_CHILD\_SUBREAPER – if arg2 is not zero, set "child subreaper" attribute for calling process, if arg2 is zero, unset

- PR\_GET\_CHILD\_SUBREAPER – return "child subreaper" setting of calling process in location pointed to by arg2
- PR\_SET\_DUMPABLE – set state of dumpable flag via arg2
- PR\_GET\_DUMPABLE – return current dumpable flag for calling process
- PR\_SET\_ENDIAN – set endian-ness of calling process to arg2 via PR\_ENDIAN\_BIG, PR\_ENDIAN\_LITTLE, or PR\_ENDIAN\_PPC\_LITTLE
- PR\_GET\_ENDIAN – return endian-ness of calling process to location pointed by arg2
- PR\_SET\_KEEPcaps – set state of calling process's "keep capabilities" flag via arg2
- PR\_GET\_KEEPcaps – return current state of calling process's "keep capabilities" flag
- PR\_MCE\_KILL – set machine check memory corruption kill policy for calling process via arg2
- PR\_MCE\_KILL\_GET – return current per-process machine check kill policy
- PR\_SET\_MM – modify kernel memory map descriptor fields of calling process, where arg2 is one of the following options and arg3 is the new value to set
  - PR\_SET\_MM\_START\_CODE – set address above which program text can run
  - PR\_SET\_MM\_END\_CODE – set address below which program text can run
  - PR\_SET\_MM\_START\_DATA – set address above which initialized and uninitialized data are placed
  - PR\_SET\_MM\_END\_DATA – set address below which initialized and uninitialized data are placed
  - PR\_SET\_MM\_START\_STACK – set start address of stack
  - PR\_SET\_MM\_START\_BRK – set address above which program heap can be expanded with brk
  - PR\_SET\_MM\_BRK – set current brk value
  - PR\_SET\_MM\_ARG\_START – set address above which command line is placed
  - PR\_SET\_MM\_ARG\_END – set address below which command line is placed
  - PR\_SET\_MM\_ENV\_START – set address above which environment is placed

- PR\_SET\_MM\_ENV\_END – set address below which environment is placed
- PR\_SET\_MM\_AUXV – set new aux vector, with arg3 providing new address and arg4 containing size of vector
- PR\_SET\_MM\_EXE\_FILE – Supersede /proc/pid/exe symlink with a new one pointing to file descriptor in arg3
- PR\_SET\_MM\_MAP – provide one-shot access to all addresses by passing struct prctl\_mm\_map pointer in arg3 with size in arg4
- PR\_SET\_MM\_MAP\_SIZE – returns size of prctl\_mm\_map structure, where arg4 is pointer to unsigned int
- PR\_MPX\_ENABLE\_MANAGEMENT – enable kernel management of memory protection extensions
- PR\_MPX\_DISABLE\_MANAGEMENT – disable kernel management of memory protection extensions
- PR\_SET\_NAME – set name of calling process to null-terminated string pointed to by arg2
- PR\_GET\_NAME – get name of calling process in null-terminated string into buffer sized to 16 bytes referenced by pointer in arg2
- PR\_SET\_NO\_NEW\_PRIVS – set calling process no\_new\_privs attribute to value in arg2
- PR\_GET\_NO\_NEW\_PRIVS – return value of no\_new\_privs for calling process
- PR\_SET\_PDEATHSIG – set parent-death signal of calling process to arg2
- PR\_GET\_PDEATHSIG – return value of parent-death signal into arg2
- PR\_SET\_SECCOMP – set "seccomp" mode for calling process via arg2
- PR\_GET\_SECCOMP – get "seccomp" mode of calling process
- PR\_SET\_SECUREBITS – set "securebits" flags of calling thread to value in arg2
- PR\_GET\_SECUREBITS – return "securebits" flags of calling process
- PR\_GET\_SPECULATION\_CTRL – return state of speculation misfeature specified in arg2
- PR\_SET\_SPECULATION\_CTRL – set state of speculation misfeature specified in arg2

- PR\_SET\_THP\_DISABLE – set state of "THP disable" flag for calling process
- PR\_TASK\_PERF\_EVENTS\_DISABLE – disable all performance counters for calling process
- PR\_TASK\_PERF\_EVENTS\_ENABLE – enable performance counters for calling process
- PR\_GET\_THP\_DISABLE – return current setting of "THP disable" flag
- PR\_GET\_TID\_ADDRESS – return `clear_child_tid` address set by `set_tid_address`
- PR\_SET\_TIMERSLACK – sets current timer slack value for calling process
- PR\_GET\_TIMERSLACK – return current timer slack value for calling process
- PR\_SET\_TIMING – set statistical process timing or accurate timestamp-based process timing by flag in `arg2` (`PR_TIMING_STATISTICAL` or `PR_TIMING_TIMESTAMP`)
- PR\_GET\_TIMING – return process timing method in use
- PR\_SET\_TSC – set state of flag determining if timestamp counter can be read by process in `arg2` (`PR_TSC_ENABLE` or `PR_TSC_SIGSEGV`)
- PR\_GET\_TSC – return state of flag determining whether timestamp counter can be read in location pointed by `arg2`

Returns zero on success or value specified in `option` flag.

## arch\_prctl

Set architecture-specific thread state.

```
int arch_prctl(int code, unsigned long addr)
```

- `code` – defines additional behavior
- `addr` or `*addr` – address, or pointer in the case of "get" operations
- `ARCH_SET_FS` – set 64-bit base for FS register to `addr`
- `ARCH_GET_FS` – return 64-bit base value for FS register of current process in memory referenced by `addr`
- `ARCH_SET_GS` – set 64-bit base address for GS register to `addr`
- `ARCH_GET_GS` – return 64-bit base value for GS register of current process in memory referenced by `addr`

Returns zero on success.

## adjtimex

Tunes kernel clock.

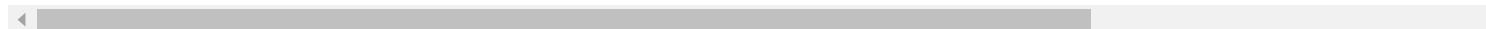
```
int adjtimex(struct timex *buf)
```

- `buf` – pointer to buffer with `timex` structure

```
struct timex {  
    int modes;           /* mode selector */
```

```
long offset;           /* time offset in nanoseconds if STA_NANO flag set, otherwise
long freq;            /* frequency offset */
long maxerror;        /* max error in microseconds */
long esterror;        /* est. error in microseconds */
int status;           /* clock command / status */
long constant;        /* PLL (phase-locked loop) time constant */
long precision;       /* clock precision in microseconds, read-only */
long tolerance;       /* clock frequency tolerance, read-only */
struct timeval time;  /* current time (read-only, except ADJ_SETOFFSET) */
long tick;             /* microseconds between clock ticks */
long ppsfreq;          /* PPS (pulse per second) frequency, read-only */
long jitter;           /* PPS jitter, read-only, in nanoseconds if STA_NANO flag set */
int shift;              /* PPS interval duration in seconds, read-only */
long stabil;           /* PPS stability, read-only */
long jitcnt;           /* PPS count of jitter limit exceeded events, read-only */
long calcnt;           /* PPS count of calibration intervals, read-only */
long errcnt;           /* PPS count of calibration errors, read-only */
long stbcnt;           /* PPS count of stability limit exceeded events, read-only */
int tai;                /* TAI offset set by previous ADJ_TAI operations, in seconds
/* padding bytes to allow future expansion */

};
```



Return clock state, either TIME\_OK, TIME\_INS, TIME\_DEL, TIME\_OOP, TIME\_WAIT, or TIME\_ERROR.

## setrlimit

Set resource limits.

```
int setrlimit(int resource, const struct rlimit *rlim)
• resource – type of resource to set (see getrlimit for list)
• rlim – pointer to rlimit structure
```

```
struct rlimit {
    rlim_t rlim_cur; /* soft limit */
    rlim_t rlim_max; /* hard limit */
};
```

Returns zero on success.

## chroot

Change root directory.

```
int chroot(const char *path)
• path – pointer to string containing path to new mount
```

Returns zero on success.

## sync

Flush filesystem caches to disk.

```
void sync(void)
```

Returns zero on success.

## acct

Toggle process accounting.

```
int acct(const char *filename)
```

- filename – pointer to string with existing file

Returns zero on success.

## settimeofday

Set the time of day.

```
int gettimeofday(const struct timeval *tv, const struct timezone *tz)
```

- tv – pointer to `timeval` structure of new time (see `gettimeofday` for structure)
- tz – pointer to `timezone` structure (see `gettimeofday` for structure)

Returns zero on success.

## mount

Mount a file system.

```
int mount(const char *source, const char *target, const char *filesystemtype,  
unsigned long mountflags, const void *data)
```

- source – pointer to string containing device path
- target – pointer to string containing mount target path
- filesystemtype – pointer to filesystem type (see `/proc/filesystems` for supported filesystems)
- mountflags – flags or mount options
- data – usually a comma-separated list of options understood by the filesystem type

Returns zero on success.

## mountflags

- MS\_BIND – perform bind mount, making file or subtree visible at another point within file system
- MS\_DIRSYNC – make dir changes synchronous
- MS\_MANDLOCK – allow mandatory locking
- MS\_MOVE – move subtree, source specifies existing mount point and target specifies new location
- MS\_NOATIME – don't update access time
- MS\_NODEV – don't allow access to special files
- MS\_NODIRATIME – don't update access times for directories
- MS\_NOEXEC – don't allow programs to be executed
- MS\_NOSUID – don't honor SUID or SGID bits when running programs
- MS\_RDONLY – mount read-only
- MS\_RELATIME – update last access time if current value of atime is less or equal to mtime or ctime
- MS\_REMOUNT – remount existing mount
- MS\_SILENT – suppress display of printk() warning messages in kernel log
- MS\_STRICTATIME – always update atime when accessed
- MS\_SYNCHRONOUS – make write synchronous

## umount2

## Unmount a filesystem.

```
int umount2(const char *target, int flags)
• target – pointer to string with filesystem to umount
• flags – additional options
```

Returns zero on success.

### flags

- MNT\_FORCE – force unmount even if busy, which can cause data loss
- MNT\_DETACH – perform lazy unmount and make mount point unavailable for new access, then actually unmount when mount isn't busy
- MNT\_EXPIRE – mark mount point as expired
- UMOOUNT\_NOFOLLOW – do not dereference target if symlink

### swapon

Start swapping to specified device.

```
int swapon(const char *path, int swapflags)
• path – pointer to string with path to device
• swapflags – flags for additional options
```

Returns zero on success.

## swapflags

- SWAP\_FLAG\_PREFER – new swap area will have a higher priority than the default priority level
- SWAP\_FLAG\_DISCARD – discard or trim freed swap pages (for SSDs)

## swapoff

Stop swapping to specified device.

```
int swapoff(const char *path)
```

- path – pointer to string with path to device

Returns zero on success.

## reboot

Reboot the system.

```
int reboot(int magic, int magic2, int cmd, void *arg)
```

- magic – must be set to `LINUX_REBOOT_MAGIC1` or `LINUX_REBOOT_MAGIC2A` for this call to work
- magic2 – must be set to `LINUX_REBOOT_MAGIC2` or `LINUX_REBOOT_MAGIC2C` for this call to work
- arg – pointer to additional argument flag

Does not return on success, returns -1 on failure.

## arg

- `LINUX_REBOOT_CMD_CAD_OFF` – `CTRL+ALT+DELETE` is disabled, and `CTRL+ALT+DELETE` will send `SIGINT` to `init`
- `LINUX_REBOOT_CMD_CAD_ON` – `CTRL+ALT+DELETE` enabled
- `LINUX_REBOOT_CMD_HALT` – halt system and display "System halted."
- `LINUX_REBOOT_CMD_KEXEC` – execute a previously loaded kernel with `kexec_load`, requires `CONFIG_KEXEC` in kernel
- `LINUX_REBOOT_CMD_POWER_OFF` – power down system
- `LINUX_REBOOT_CMD_RESTART` – restart system and display "Restarting system."
- `LINUX_REBOOT_CMD_RESTART2` – restart system and display "Restarting system with command aq%saq."

## sethostname

Set hostname of machine.

```
int sethostname(const char *name, size_t len)
• name – pointer to string with new name
• len – length of new name
```

Returns zero on success.

## setdomainname

Set NIS domain name.

```
int setdomainname(const char *name, size_t len)
```

- **name** – pointer to string with new name
- **len** – length of new name

Return zero on success.

## iopl

Change I/O privilage level

```
int iopl(int level)
```

- **level** – new privilege level

Returns zero on success.

## ioperm

Set I/O permissions.

```
int ioperm(unsigned long from, unsigned long num, int turn_on)
```

- `from` – starting port address
- `num` – number of bits
- `turn_on` – zero or non-zero denotes enabled or disabled

Returns zero on success.

## init\_module

Load module into kernel with module file specified by file descriptor.

```
int init_module(void *module_image, unsigned long len, const char *param_values)
```

- `module_image` – pointer to buffer with binary image of module to load
- `len` – size of buffer
- `param_values` – pointer to string with parameters for kernel

Returns zero on success.

## delete\_module

Unload a kernel module.

```
int delete_module(const char *name, int flags)
```

- name – pointer to string with name of module
- flags – modify behavior of unload

Return zero on success.

## flags

- O\_NONBLOCK – immediately return from syscall
- O\_NONBLOCK | O\_TRUNC – unload module immediately even if reference count is not zero

## quotactl

Change disk quotas.

```
int quotactl(int cmd, const char *special, int id, caddr_t addr)
```

- cmd – command flag
- special – pointer to string with path to mounted block device
- id – user or group ID
- addr – address of data structure, optional to some cmd flags

## cmd

- Q\_QUOTAON – turn on quotas for filesystem referenced by special, with id specifying quota format to use:
  - QFMT\_VFS\_OLD – original format
  - QFMT\_VFS\_V0 – standard VFS v0 format
  - QFMT\_VFS\_V1 – format with support for 32-bit UIDs and GIDs
- Q\_QUOTAOFF – turn off quotas for filesystem referenced by special
- Q\_GETQUOTA – get quota limits and usage for a user or group id, referenced by id, where addr is pointer to dqblk structure
- Q\_GETNEXTQUOTA – same as Q\_GETQUOTA but returns info for next id greater or equal to id that has quota set, where addr points to nextdqblk structure
- Q\_SETQUOTA – set quota info for user or group id, using dqblk structure referenced by addr
- Q\_GETINFO – get info about quotafile, where addr points to dqinfo structure
- Q\_SETINFO – set information about quotafile, where addr points to dqinfo structure
- Q\_GETFMT – get quota format used on filesystem referenced by special, where addr points to 4 byte buffer where format number will be stored
- Q\_SYNC – update on-disk copy of quota usage for filesystem
- Q\_GETSTATS – get statistics about quota subsystem, where addr points to a dqstats structure

- Q\_XQUOTAON – enable quotas for an XFS filesystem
- Q\_XQUOTAOFF – disable quotas on an XFS filesystem
- Q\_XGETQUOTA – on XFS filesystems, get disk quota limits and usage for user id specified by `id`, where `addr` points to `fs_disk_quota` structure
- Q\_XGETNEXTQUOTA – same as Q\_XGETQUOTA but returns `fs_disk_quota` referenced by `addr` for next id greater or equal than `id` that has quota set
- Q\_XSETQLIM – on XFS filesystems, set disk quota for UID, where `addr` references pointer to `fs_disk_quota` structure
- Q\_XGETQSTAT – returns XFS specific quota info in `fs_quota_stat` referenced by `addr`
- Q\_XGETQSTATV – returns XFS specific quota info in `fs_quota_statv` referenced by `addr`
- Q\_XQUOTARM – on XFS filesystems, free disk space used by quotas, where `addr` references unsigned int value containing flags (same as `d_flags` field of `fs_disk_quota` structure)

```
struct dqblk {  
    uint64_t dqb_bhardlimit; /* absolute limit on quota blocks alloc */  
    uint64_t dqb_bsoftlimit; /* preferred limit on quota blocks */  
    uint64_t dqb_curspace; /* current space used in bytes */  
    uint64_t dqb_ihardlimit; /* max number of allocated inodes */  
    uint64_t dqb_isoftlimit; /* preferred inode limit */  
    uint64_t dqb_curinodes; /* current allocated inodes */  
    uint64_t dqb_bttime; /* time limit for excessive use over quota */  
    uint64_t dqb_itime; /* time limit for excessive files */  
    uint32_t dqb_valid; /* bit mask of QIF_* constants */  
};
```

```
struct nextdqblk {  
    uint64_t dqb_bhardlimit;  
    uint64_t dqb_bsoftlimit;  
    uint64_t dqb_curspace;  
    uint64_t dqb_ihardlimit;  
    uint64_t dqb_isoftlimit;  
    uint64_t dqb_curinodes;  
    uint64_t dqb_btime;  
    uint64_t dqb_itime;  
    uint32_t dqb_valid;  
    uint32_t dqb_id;  
};
```

```
struct dqinfo {  
    uint64_t dqi_bgrace; /* time before soft limit becomes hard limit */  
    uint64_t dqi_igrace; /* time before soft inode limit becomes hard limit */  
    uint32_t dqi_flags; /* flags for quotafile */  
    uint32_t dqi_valid;  
};
```

```
struct fs_disk_quota {  
    int8_t d_version; /* version of structure */  
    int8_t d_flags; /* XFS_{USER,PROJ,GROUP}_QUOTA */  
    uint16_t d_fieldmask; /* field specifier */  
    uint32_t d_id; /* project, UID, or GID */  
    uint64_t d_blk_hardlimit; /* absolute limit on disk blocks */  
    uint64_t d_blk_softlimit; /* preferred limit on disk blocks */
```

```
    uint64_t d_ino_hardlimit; /* max # allocated inodes */
    uint64_t d_ino_softlimit; /* preferred inode limit */
    uint64_t d_bcount; /* # disk blocks owned by user */
    uint64_t d_icount; /* # inodes owned by user */
    int32_t d_itimer; /* zero if within inode limits */
    int32_t d_btimer; /* as above for disk blocks */
    uint16_t d_iwarns; /* # warnings issued regarding # of inodes */
    uint16_t d_bwarns; /* # warnings issued regarding disk blocks */
    int32_t d_padding2; /* padding */
    uint64_t d_rtb_hardlimit; /* absolute limit on realtime disk blocks */
    uint64_t d_rtb_softlimit; /* preferred limit on realtime disk blocks */
    uint64_t d_rtbcnt; /* # realtime blocks owned */
    int32_t d_rtbtimer; /* as above, but for realtime disk blocks */
    uint16_t d_rtbwarns; /* # warnings issued regarding realtime disk blocks */
    int16_t d_padding3; /* padding */
    char d_padding4[8]; /* padding */
};


```

```
struct fs_quota_stat {
    int8_t qs_version; /* version for future changes */
    uint16_t qs_flags; /* XFS_QUOTA_{U,P,G}DQ_{ACCT,ENFD} */
    int8_t qs_pad; /* padding */
    struct fs_qfilestat qs_uquota; /* user quota storage info */
    struct fs_qfilestat qs_gquota; /* group quota storage info */
    uint32_t qs_incoredq; /* number of dqots in core */
    int32_t qs_btimelimit; /* limit for blocks timer */
    int32_t qs_itimelimit; /* limit for inodes timer */
    int32_t qs_rtbtimelimit; /* limit for realtime blocks timer */
    uint16_t qs_bwarnlimit; /* limit for # of warnings */
    uint16_t qs_iwarnlimit; /* limit for # of warnings */
};
```

```
struct fs_qfilestatv {  
    uint64_t qfs_ino;          /* inode number */  
    uint64_t qfs_nblk;         /* number of BBs (512-byte blocks) */  
    uint32_t qfs_extents;      /* number of extents */  
    uint32_t qfs_pad;          /* pad for 8-byte alignment */  
};  
  
struct fs_quota_statv {  
    int8_t   qs_version;       /* version for future changes */  
    uint8_t   qs_pad1;          /* pad for 16-bit alignment */  
    uint16_t  qs_flags;         /* XFS_QUOTA_.* flags */  
    uint32_t  qs_incoredqs;     /* number of dquots incore */  
    struct fs_qfilestatv qs_uquota; /* user quota info */  
    struct fs_qfilestatv qs_gquota; /* group quota info */  
    struct fs_qfilestatv qs_pquota; /* project quota info */  
    int32_t   qs_btimelimit;    /* limit for blocks timer */  
    int32_t   qs_itimelimit;    /* limit for inodes timer */  
    int32_t   qs_rtbtimelimit;  /* limit for realtime blocks timer */  
    uint16_t  qs_bwarnlimit;    /* limit for # of warnings */  
    uint16_t  qs_iwarnlimit;    /* limit for # of warnings */  
    uint64_t  qs_pad2[8];        /* padding */  
};
```

Returns zero on success.

## gettid

Get thread ID.

```
pid_t gettid(void)
```

Returns thread ID of calling process.

## readahead

Read file into page cache.

```
ssize_t readahead(int fd, off64_t offset, size_t count)
```

- **fd** – file descriptor of file to read ahead
- **offset** – offset from start of file to read
- **count** – number of bytes to read

Returns zero on success.

## setxattr

Set extended attribute value.

```
int setxattr(const char *path, const char *name, const void *value,  
size_t size, int flags)
```

- path – pointer to string with filename
- name – pointer to string with attribute name
- value – pointer to string with attribute value
- size – size of value
- flags – set to XATTR\_CREATE to create attribute, XATTR\_REPLACE to replace

Returns zero on success.

## lsetxattr

Set extended attribute value of symbolic link.

```
int lsetxattr(const char *path, const char *name, const void *value,  
size_t size, int flags)
```

- path – pointer to string with symlink
- name – pointer to string with attribute name
- value – pointer to string with attribute value
- size – size of value
- flags – set to XATTR\_CREATE to create attribute, XATTR\_REPLACE to replace

Returns zero on success.

## fsetxattr

Set extended attribute value of file referenced by file descriptor.

```
int fsetxattr(int fd, const char *name, const void *value, size_t size, int flags)
```

- fd – file descriptor of file in question
- name – pointer to string with attribute name
- value – pointer to string with attribute value
- size – size of value
- flags – set to XATTR\_CREATE to create attribute, XATTR\_REPLACE to replace

Returns zero on success.

## getxattr

Get extended attribute value.

```
ssize_t getxattr(const char *path, const char *name, void *value, size_t size)
```

- `path` – pointer to string with filename
- `name` – pointer to string with attribute name
- `value` – pointer to string with attribute value
- `size` – size of value

Returns size of extended attribute value.

## lgetxattr

Get extended attribute value from symlink.

```
ssize_t lgetxattr(const char *path, const char *name, void *value, size_t size)
```

- `path` – pointer to string with symlink
- `name` – pointer to string with attribute name
- `value` – pointer to string with attribute value
- `size` – size of value

Returns size of extended attribute value.

## fgetxattr

Get extended attribute value from file referenced by file descriptor.

```
ssize_t fgetxattr(int fd, const char *name, void *value, size_t size)
```

- `fd` – file descriptor of file in question
- `name` – pointer to string with attribute name
- `value` – pointer to string with attribute value
- `size` – size of `value`

Returns size of extended attribute value.

## listxattr

List extended attribute names.

```
ssize_t listxattr(const char *path, char *list, size_t size)
```

- path – pointer to string with filename
- list – pointer to list of attribute names
- size – size of list buffer

Returns size of name list.

## llistxattr

List extended attribute names for a symlink.

```
ssize_t llistxattr(const char *path, char *list, size_t size)
```

- path – pointer to string with symlink
- list – pointer to list of attribute names
- size – size of list buffer

Returns size of name list.

## flistxattr

List extended attribute names for file referenced by file descriptor.

```
ssize_t flistxattr(int fd, char *list, size_t size)
```

- `fd` – file descriptor of file in question
- `list` – pointer to list of attribute names
- `size` – size of list buffer

Returns size of name list.

## removexattr

Remove an extended attribute.

```
int removexattr(const char *path, const char *name)
```

- `path` – pointer to string with filename
- `name` – pointer to string with name of attribute to remove

Returns zero on success.

## **lremovexattr**

Remove an extended attribute of a symlink.

```
int lremovexattr(const char *path, const char *name)
```

- path – pointer to string with filename
- name – pointer to string with name of attribute to remove

Returns zero on success.

## **fremovexattr**

Remove an extended attribute of a file referenced by a file descriptor.

```
int fremovexattr(int fd, const char *name)
```

- fd – file descriptor of file in question
- name – pointer to string with name of attribute to remove

Returns zero on success.

## tkill

Send a signal to a thread.

```
int tkill(int tid, int sig)
• tid – thread id
• sig – signal to send
```

Returns zero on success.

## time

Get time in seconds.

```
time_t time(time_t *t)
• t – if not NULL, return value is also stored in referenced memory address
```

Returns time (in seconds) since UNIX Epoch.

## futex

Fast user-space locking.

```
int futex(int *uaddr, int op, int val, const struct timespec *timeout,
          int *uaddr2, int val3)
```

- `uaddr` – pointer to address of value to monitor for change
- `op` – operation flag
- `timeout` – pointer to `timespec` structure with timeout
- `uaddr2` – pointer to integer used for some operations
- `val3` – additional argument in some operations

Return value depends on operation detailed above.

## op

- `FUTEX_WAIT` – atomically verifies that `uaddr` still contains value `val` and sleeps awaiting `FUTEX_WAKE` on this address
- `FUTEX_WAKE` – wakes at most `val` processes waiting on futex address
- `FUTEX_REQUEUE` – wakes up `val` processes and requeues all waiters on futex at address `uaddr2`
- `FUTEX_CMP_REQUEUE` – similar to `FUTEX_REQUEUE` but first checks if location `uaddr` contains value of `val3`

## sched\_setaffinity

Set process CPU affinity mask.

```
int sched_setaffinity(pid_t pid, size_t cpusetsize, cpu_set_t *mask)
```

- pid – PID of process
- cpusetsize – length of data at `mask`
- mask – pointer to mask

Returns zero on success.

## sched\_getaffinity

Get process CPU affinity mask.

```
int sched_getaffinity(pid_t pid, size_t cpusetsize, cpu_set_t *mask)
```

- pid – PID of process
- cpusetsize – length of data at `mask`

- `mask` – pointer to mask

Returns zero on success with mask placed in memory referenced by `mask`.

## **set\_thread\_area**

Set thread local storage area.

```
int set_thread_area(struct user_desc *u_info)
• u_info – pointer to user_desc structure
```

Returns zero on success.

## **io\_setup**

Create async I/O context.

```
int io_setup(unsigned nr_events, aio_context_t *ctx_idp)
```

- `nr_events` – total number of events to receive
- `ctx_idp` – pointer reference to created handle

Returns zero on success.

## io\_destroy

Destroy async I/O context.

```
int io_destroy(aio_context_t ctx_id)
```

- `ctx_id` – ID of context to destroy

Returns zero on success.

## io\_getevents

Read async I/O events from queue.

```
int io_getevents(aio_context_t ctx_id, long min_nr, long nr, struct io_event
*eventsstruct, timespec *timeout)
```

- `ctx_id` – AIO context ID
- `min_nr` – minimum number of events to read
- `nr` – number of events to read
- `eventsstruct` – pointer to `io_event` structure
- `timeout` – pointer to `timespec` timeout structure

Returns number of events read, or zero if no events are available or are less than `min_nr`.

## io\_submit

Submit async I/O blocks for processing.

```
int io_submit(aio_context_t ctx_id, long nrstruct, iocb *iocbpp)
```

- `ctx_id` – AIO context ID
- `nrstruct` – number of structures
- `iocbpp` – pointer to `iocb` structure

Returns number of `iocb` submitted.

## io\_cancel

Cancel previously submitted async I/O operation.

```
int io_cancel(aio_context_t ctx_id, struct iocb *iocb, struct io_event *result)
```

- `ctx_id` – AIO context ID
- `iocb` – pointer to `iocb` structure
- `result` – pointer to `io_event` structure

Returns zero on success and copies event to memory referenced by `result`.

## get\_thread\_area

Get a thread local storage area.

```
int get_thread_area(struct user_desc *u_info)
```

- `u_info` – pointer to `user_desc` structure to receive data

Returns zero on success.

## lookup\_dcookie

Return directory entry's path.

```
int lookup_dcookie(u64 cookie, char *buffer, size_t len)
```

- cookie – unique identifier of a directory entry
- buffer – pointer to buffer with full path of directory entry
- len – length of buffer

Returns bytes written to `buffer` with path string.

## epoll\_create

Open epoll file descriptor.

```
int epoll_create(int size)
```

- size – ignored, but must be greater than 0

Returns file descriptor.

## getdents64

Get directory entries.

```
int getdents(unsigned int fd, struct linux_dirent *dirp, unsigned int count)
```

- fd – file descriptor of directory

- `dirp` – pointer to `linux_dirent` structure for results
- `count` – size of the `dirp` buffer

```
struct linux_dirent {  
    unsigned long d_ino;      /* inode number */  
    unsigned long d_off;      /* offset to next linux_dirent */  
    unsigned short d_reclen;  /* length of this linux_dirent */  
    char          d_name[];   /* null-terminated filename */  
    char          pad;        /* zero padding byte */  
    char          d_type;     /* file type */  
}
```

Returns bytes read, and at end of directory returns zero.

## set\_tid\_address

Set pointer to thread ID.

```
long set_tid_address(int *tidptr)  
    • tidptr – pointer to thread ID
```

Returns PID of calling process.

## restart\_syscall

Restart a syscall.

```
long sys_restart_syscall(void)
```

Returns value of system call it restarts.

## semtimedop

Same as the `semop` syscall except if calling thread would sleep, duration is limited to timeout.

```
int semtimedop(int semid, struct sembuf *sops, unsigned nsops, struct timespec *timeout)
```

- `semid` – id of semaphore
- `sops` – pointer to `sembuf` structure for operations
- `nsops` – number of operations
- `timeout` – timeout for calling thread, and upon return from syscall time elapsed placed in structure

Returns zero on success.

## fadvise64

Predeclare access pattern for file data to allow kernel to optimize I/O operations.

```
int posix_fadvise(int fd, off_t offset, off_t len, int advice)
```

- `fd` – file descriptor of file in question
- `offset` – offset that access will begin
- `len` – length of anticipated access, or 0 to end of file
- `advice` – advice to give kernel

Returns zero on success.

## advice

- POSIX\_FADV\_NORMAL – application has no specific advice
- POSIX\_FADV\_SEQUENTIAL – application expects to access data sequentially
- POSIX\_FADV\_RANDOM – data will be accessed randomly
- POSIX\_FADV\_NOREUSE – data will be accessed only once
- POSIX\_FADV\_WILLNEED – data will be needed in near future
- POSIX\_FADV\_DONTNEED – data will not be needed in near future

## timer\_create

Create POSIX per-process timer.

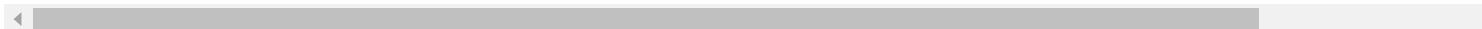
```
int timer_create(clockid_t clockid, struct sigevent *sevp, timer_t *timerid)
```

- `clockid` – type of clock to use
- `sevp` – pointer to `sigevent` structure explaining how caller will be notified when timer expires
- `timerid` – pointer to buffer that will receive timer ID

Returns zero on success.

```
union sigval {  
    int     sival_int;  
    void   *sival_ptr;  
};
```

```
struct sigevent {  
    int           sigev_notify; /* method of notification */  
    int           sigev_signo; /* notification signal */  
    union sigval sigev_value; /* data to pass with notification */  
    void         (*sigev_notify_function) (union sigval); /* Function used for thread not  
    void         *sigev_notify_attributes; /* attributes for notification thread */  
    pid_t        sigev_notify_thread_id; /* id of thread to signal */  
};
```



## clockid

- CLOCK\_REALTIME – settable system wide real time clock
- CLOCK\_MONOTONIC – nonsettable monotonically increasing clock measuring time from unspecified point in past
- CLOCK\_PROCESS\_CPUTIME\_ID – clock measuring CPU time consumed by the calling process and its threads
- CLOCK\_THREAD\_CPUTIME\_ID – clock measuring CPU time consumed by calling thread

## timer\_settime

Arm or disarm POSIX per-process timer.

```
int timer_settime(timer_t timerid, int flags, const struct itimerspec *new_value,  
                  struct itimerspec *old_value)  
• timerid – id of timer  
• flags – specify TIMER_ABSTIME to process new_value->it_value as an absolute value  
• new_value – pointer to itimerspec structure defining new initial and new interval for timer
```

- `old_value` – pointer to structure to receive previous timer details

```
struct itimerspec {  
    struct timespec it_interval; /* interval */  
    struct timespec it_value;   /* expiration */  
};
```

Returns zero on success.

## timer\_gettime

Returns time until next expiration from POSIX per-process timer.

```
int timer_gettime(timer_t timerid, struct itimerspec *curr_value)
```

- `timerid` – id of timer
- `curr_value` – pointer to `itimerspec` structure where current timer values are returned

Returns zero on success.

## timer\_getoverrun

Get overrun count on a POSIX per-process timer.

```
int timer_getoverrun(timer_t timerid)
```

- `timerid` – id of timer

Returns overrun count of specified timer.

## timer\_delete

Delete POSIX per-process timer.

```
int timer_delete(timer_t timerid)
```

- `timerid` – id of timer

Returns zero on success.

## clock\_settime

Set specified clock.

```
int clock_settime(clockid_t clk_id, const struct timespec *tp)
```

- `clk_id` – clock id
- `tp` – pointer to `timespec` structure with clock details

Returns zero on success.

## clock\_gettime

Get time from specified clock.

```
int clock_gettime(clockid_t clk_id, struct timespec *tp)
```

- `clk_id` – clock id
- `tp` – pointer to `timespec` structure returned with clock details

Returns zero on success.

## clock\_getres

Obtain resolution of specified clock.

```
int clock_getres(clockid_t clk_id, struct timespec *res)
```

- `clk_id` – clock id
- `res` – pointer to `timespec` structure returned with details

Returns zero on success.

## clock\_nanosleep

High-resolution sleep with specifiable clock.

```
int clock_nanosleep(clockid_t clock_id, int flags, const struct timespec  
*request, struct timespec *remain)
```

- `clock_id` – type of clock to use
- `flags` – specify `TIMER_ABSTIME` to process request as an absolute value
- `remain` – pointer to `timespec` structure to receive remaining time on sleep

Returns zero after sleep interval.

## exit\_group

Exit all threads in a process.

```
void exit_group(int status)
• status – status code to return
```

Does not return.

## epoll\_wait

Wait for I/O event on epoll file descriptor.

```
int epoll_wait(int epfd, struct epoll_event *events, int maxevents, int timeout)
• epfd – epoll file descriptor
• events – pointer to epoll_event structure with events available to calling process
• maxevents – maximum number of events, must be greater than zero
• timeout – timeout in milliseconds
```

```
typedef union epoll_data {
    void    *ptr;
    int      fd;
    uint32_t u32;
    uint64_t u64;
} epoll_data_t;
```

```
struct epoll_event {
    uint32_t   events;    /* epoll events */
    epoll_data_t data;    /* user data variable */
};
```

Returns number of file descriptors ready for requested I/O or zero if timeout occurred before any were available.

## epoll\_ctl

Control interface for epoll file descriptor.

```
int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event)
```

- `epfd` – epoll file descriptor
- `op` – operation flag
- `fd` – file descriptor for target file
- `event` – pointer to `epoll_event` structure with event, purpose altered by `op`

Returns zero on success.

## op

- `EPOLL_CTL_ADD` – add `fd` to interest list
- `EPOLL_CTL_MOD` – change settings associated with `fd` in interest list to new settings specified in `event`
- `EPOLL_CTL_DEL` – remove target file descriptor `fd` from interest list, with `event` argument ignored

## tgkill

Send signal to a thread.

```
int tgkill(int tgid, int tid, int sig)
```

- `tgid` – thread group id
- `tid` – thread id
- `sig` – signal to send

Returns zero on success.

## utimes

Change file last access and modification times.

```
int utimes(const char *filename, const struct timeval times[2])
```

- `filename` – pointer to string with file in question
- `times` – array of `timeval` structure where `times[0]` specifies new access time where `times[1]` specifies new modification time

Returns zero on success.

## mbind

Set NUMA memory policy on a memory range.

```
long mbind(void *addr, unsigned long len, int mode, const unsigned long  
*nodemask, unsigned long maxnode, unsigned flags)
```

- `addr` – pointer to starting memory address
- `len` – length of memory segment
- `mode` – NUMA mode
- `nodemask` – pointer to mask defining nodes that mode applies to

- `maxnode` – max number of bits for `nodemask`
- `flags` – set `MPOL_F_STATIC_NODES` to specify physical nodes, `MPOL_F_RELATIVE_NODES` to specify node ids relative to set allowed by threads current cpuset

Returns zero on success.

## mode

- `MPOL_DEFAULT` – remove any nondefault policy and restore default behavior
- `MPOL_BIND` – specify policy restricting memory allocation to node specified in `nodemask`
- `MPOL_INTERLEAVE` – specify page allocations be interleaved across set of nodes specified in `nodemask`
- `MPOL_PREFERRED` – set preferred node for allocation
- `MPOL_LOCAL` – mode specifies "local allocation" – memory is allocated on the node of the CPU that triggers allocation

## set\_mempolicy

Set default NUMA memory policy for thread and its offspring.

```
long set_mempolicy(int mode, const unsigned long *nodemask,  
                   unsigned long maxnode)
```

- `mode` – NUMA mode
- `nodemask` – pointer to mask defining node that mode applies to
- `maxnode` – max number of bits for `nodemask`

Return zero on success.

## get\_mempolicy

Get NUMA memory policy for thread and its offspring.

```
long get_mempolicy(int *mode, unsigned long *nodemask, unsigned long maxnode,
void *addr, unsigned long flags)
```

- mode – NUMA mode
- nodemask – pointer to mask defining node that mode applies to
- maxnode – max number of bits for nodemask
- addr – pointer to memory region
- flags – defines behavior of call

Return zero on success.

## flags

- MPOL\_F\_NODE or 0 (zero preferred) – get information about calling thread's default policy and store in nodemask buffer
- MPOL\_F\_MEMS\_ALLOWED – mode argument is ignored and subsequent calls return set of nodes thread is allowed to specify is returned in nodemask
- MPOL\_F\_ADDR – get information about policy for addr

## mq\_open

Creates a new or open existing POSIX message queue.

```
mqd_t mq_open(const char *name, int oflag)
mqd_t mq_open(const char *name, int oflag, mode_t mode, struct mq_attr *attr)
```

- name – pointer to string with name of queue
- oflag – define operation of call
- mode – permissions to place on queue
- attr – pointer to `mq_attr` structure to define parameters of queue

```
struct mq_attr {  
    long mq_flags;      /* flags (not used for mq_open) */  
    long mq_maxmsg;    /* max messages on queue */  
    long mq_msgsize;   /* max message size in bytes */  
    long mq_curmsgs;   /* messages currently in queue (not used for mq_open) */  
};
```

## oflag

- `O_RDONLY` – open queue to only receive messages
- `O_WRONLY` – open queue to send messages
- `O_RDWR` – open queue for both send and receive
- `O_CLOEXEC` – set close-on-exec flag for message queue descriptor
- `O_CREAT` – create message queue if it doesn't exist
- `O_EXCL` – if `O_CREAT` specified and queue already exists, fail with `EEXIST`
- `O_NONBLOCK` – open queue in nonblocking mode

## mq\_unlink

Remove message queue.

```
int mq_unlink(const char *name)
• name – pointer to string with queue name
```

Returns zero on success.

## mq\_timedsend

Send message to message queue.

```
int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned msg_prio,
const struct timespec *abs_timeout)
```

- mqdes – descriptor pointing to message queue
- msg\_ptr – pointer to message
- msg\_len – length of message
- msg\_prio – priority of message
- abs\_timeout – pointer to timespec structure defining timeout

Returns zero on success.

## mq\_timedreceive

Receive a message from a message queue.

```
ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned *msg_prio)
```

- mqdes – descriptor pointing to message queue
- msg\_ptr – pointer to buffer to receive message
- msg\_len – length of message

Return number of bytes in received message.

## mq\_notify

Register to receive notification when message is available in a message queue.

```
int mq_notify(mqd_t mqdes, const struct sigevent *sevp)
```

- `mqdes` – descriptor pointing to message queue
- `sevp` – pointer to `sigevent` structure

Returns zero on success.

## kexec\_load

Load new kernel for execution at a later time.

```
long kexec_load(unsigned long entry, unsigned long nr_segments, struct  
kexec_segment *segments, unsigned long flags)
```

- `entry` – entry address in kernel image
- `nr_segments` – number of segments referenced by `segments` pointer
- `segments` – pointer to `kexec_segment` structure defining kernel layout
- `flags` – modify behavior of call

```
struct kexec_segment {  
    void    *buf;          /* user space buffer */  
    size_t   bufsz;        /* user space buffer length */  
    void    *mem;          /* physical address of kernel */
```

```
    size_t memsz;      /* physical address length */  
};
```

Returns zero on success.

## flags

- KEXEC\_FILE\_UNLOAD – unload currently loaded kernel
- KEXEC\_FILE\_ON\_CRASH – load new kernel in memory region reserved for crash kernel
- KEXEC\_FILE\_NO\_INITRAMFS – specify that loading initrd/initramfs is optional

## waitid

Wait for change of state in process.

```
int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options)
```

- `idtype` – defines `id` scope, specifying `P_PID` for process id, `P_PGID` process group id, or `P_ALL` to wait for any child where `id` is ignored
- `id` – id of process or process group, defined by `idtype`
- `infop` – pointer to `siginfo_t` structure filled in by return
- `options` – modifies behavior of syscall

Returns zero on success.

## options

- `WNOHANG` – return immediately if no child has exited

- WUNTRACED – also return if child was stopped but not traced
- WCONTINUED – also return if stopped child has resumed via SIGCONT
- WIFEXITED – returns true if child was terminated normally
- WEXITSTATUS – returns exit status of child
- WIFSIGNALED – returns true if child process terminated by signal
- WTERMSIG – returns signal that caused child process to terminate
- WCOREDUMP – returns true if child produced core dump
- WIFSTOPPED – returns true if child process stopped by delivery of signal
- WSTOPSIG – returns number of signal that caused child to stop
- WIFCONTINUED – returns true if child process was resumed via SIGCONT
- WEXITED – wait for terminated children
- WSTOPPED – wait for stopped children via delivery of signal
- WCONTINUED – wait for previously stopped children that were resumed via SIGCONT
- WNOWAIT – leave child in waitable state

## add\_key

Add key to kernel's key management.

```
key_serial_t add_key(const char *type, const char *description, const void
*payload, size_t plen, key_serial_t keyring)
```

- type – pointer to string with type of key
- description – pointer to string with description of key
- payload – key to add
- plen – length of key
- keyring – serial number of keyring or special flag

Returns serial number of created key.

## keyring

- KEY\_SPEC\_THREAD\_KEYRING – specifies caller's thread-specific keyring
- KEY\_SPEC\_PROCESS\_KEYRING – specifies caller's process-specific keyring
- KEY\_SPEC\_SESSION\_KEYRING – specifies caller's session-specific keyring
- KEY\_SPEC\_USER\_KEYRING – specifies caller's UID-specific keyring
- KEY\_SPEC\_USER\_SESSION\_KEYRING – specifies caller's UID-session keyring

## request\_key

Request key from kernel's key management.

```
key_serial_t request_key(const char *type, const char *description,  
const char *callout_info, key_serial_t keyring)
```

- type – pointer to string with type of key
- description – pointer to string with description of key
- callout\_info – pointer to string set if key isn't found
- keyring – serial number of keyring or special flag

Returns serial number of key found on success.

## keyctl

Manipulate kernel's key management.

```
long keyctl(int cmd, ...)
```

- `cmd` – command flag modifying syscall behavior
- ... – additional arguments per `cmd` flag

Returns serial number of key found on success.

## cmd

- KEYCTL\_GET\_KEYRING\_ID – ask for keyring id
- KEYCTL\_JOIN\_SESSION\_KEYRING – join or start named session keyring
- KEYCTL\_UPDATE – update key
- KEYCTL\_REVOKE – revoke key
- KEYCTL\_CHOWN – set ownership of key
- KEYCTL\_SETPERM – set permissions on a key
- KEYCTL\_DESCRIBE – describe key
- KEYCTL\_CLEAR – clear contents of keyring
- KEYCTL\_LINK – link key into keyring
- KEYCTL\_UNLINK – unlink key from keyring
- KEYCTL\_SEARCH – search for key in keyring
- KEYCTL\_READ – read key or keyring's contents
- KEYCTL\_INSTANTIATE – instantiate partially constructed key
- KEYCTL\_NEGATE – negate partially constructed key
- KEYCTL\_SET\_REQKEY\_KEYRING – set default request-key keyring
- KEYCTL\_SET\_TIMEOUT – set timeout on a key
- KEYCTL\_ASSUME\_AUTHORITY – assume authority to instantiate key

## ioprio\_set

Set I/O scheduling class and priority.

```
int ioprio_set(int which, int who, int ioprio)
```

- `which` – flag specifying target of `who`
- `who` – id determined by `which` flag
- `ioprio` – bit mask specifying scheduling class and priority to assign to `who` process

Returns zero on success.

## which

- `IOPRIO_WHO_PROCESS` – `who` is process or thread id, or `0` to use calling thread
- `IOPRIO_WHO_PGRP` – `who` – is a process id identifying all members of a process group, or `0` to operate on process group where calling process is member
- `IOPRIO_WHO_USER` – `who` is UID identifying all processes that have a matching real UID

## ioprio\_get

Get I/O scheduling class and priority.

```
int ioprio_get(int which, int who)
```

- `which` – flag specifying target of `who`
- `who` – id determined by `which` flag

Return `ioprio` value of process with highest I/O priority of matching processes.

## inotify\_init

Initialize an inotify instance.

```
int inotify_init(void)
```

Returns file descriptor of new inotify event queue.

## inotify\_add\_watch

Add watch to an initialized inotify instance.

```
int inotify_add_watch(int fd, const char *pathname, uint32_t mask)
```

- `fd` – file descriptor referring to inotify instance with watch list to be modified
- `pathname` – pointer to string with path to monitor
- `mask` – mask of events to be monitored

Returns watch descriptor on success.

## inotify\_rm\_watch

Remove existing watch from inotify instance.

```
int inotify_rm_watch(int fd, int wd)
```

- `fd` – file descriptor associated with watch
- `wd` – watch descriptor

Returns zero on success.

## migrate\_pages

Move pages in process to another set of nodes.

```
long migrate_pages(int pid, unsigned long maxnode, const unsigned long
*old_nodes, const unsigned long *new_nodes)
```

- pid – PID of process in question
- maxnode – max nodes in old\_nodes and new\_nodes masks
- old\_nodes – pointer to mask of node numbers to move from
- new\_nodes – pointer to mask of node numbers to move to

Returns number of pages that couldn't be moved.

## openat

Open file relative to directory file descriptor.

```
int openat(int dirfd, const char *pathname, int flags)
int openat(int dirfd, const char *pathname, int flags, mode_t mode)
```

- dirfd – file descriptor of directory
- pathname – pointer to string with path name
- flags – see open syscall
- mode – see open syscall

Returns new file descriptor on success.

## mkdirat

Create directory relative to directory file descriptor.

```
int mkdirat(int dirfd, const char *pathname, mode_t mode)
```

- `dirfd` – file descriptor of directory
- `pathname` – pointer to string with path name
- `mode` – see `mkdir` syscall

Returns zero on success.

## mknodat

Create a special file relative to directory file descriptor.

```
int mknodat(int dirfd, const char *pathname, mode_t mode, dev_t dev)
```

- `dirfd` – file descriptor of directory
- `pathname` – pointer to string with path name
- `mode` – see `mknod` syscall
- `dev` – device number

Returns zero on success.

## fchownat

Change ownership of file relative to directory file descriptor.

```
int fchownat(int dirfd, const char *pathname, uid_t owner, gid_t group, int flags)
```

- `dirfd` – file descriptor of directory

- `pathname` – pointer to string with path name
- `owner` – user id (UID)
- `group` – group id (GID)
- `flags` – if `AT_SYMLINK_NOFOLLOW` is specified, do no dereference symlinks

## unlinkat

Delete name and possibly file it references.

```
int unlinkat(int dirfd, const char *pathname, int flags)
```

- `dirfd` – file descriptor of directory
- `pathname` – pointer to string with path name
- `flags` – see `unlink` or `rmdir`

Returns zero on success.

## renameat

Change name or location of file relative to directory file descriptor.

```
int renameat(int olddirfd, const char *oldpath, int newdirfd, const char *newpath)
```

- `olddirfd` – file descriptor of directory with source
- `oldpath` – pointer to string with path name to source
- `newdirfd` – file descriptor of directory with target
- `newpath` – pointer to string with path name to target

Returns zero on success.

## linkat

Create a hard link relative to directory file descriptor.

```
int linkat(int olddirfd, const char *oldpath, int newdirfd, const char *newpath, int flags)
```

- `olddirfd` – file descriptor of directory with source
- `oldpath` – pointer to string with path name to source
- `newdirfd` – file descriptor of directory with target
- `newpath` – pointer to string with path name to target
- `flags` – see link

Returns zero on success.

## symlinkat

Create a symbolic link relative to directory file descriptor.

```
int symlinkat(const char *target, int newdirfd, const char *linkpath)
```

- `target` – pointer to string with target
- `newdirfd` – file descriptor of directory with target
- `linkpath` – pointer to string with source

Returns zero on success.

## readlinkat

Read contents of symbolic link pathname relative to directory file descriptor.

```
ssize_t readlinkat(int dirfd, const char *pathname, char *buf, size_t bufsiz)
```

- `dirfd` – file descriptor relative to symlink
- `pathname` – pointer to string with symlink path
- `buf` – pointer to buffer receiving symlink pathname
- `bufsiz` – size of `buf`

Returns number of bytes placed into `buf` on success.

## fchmodat

Change permissions of file relative to a directory file descriptor.

```
int fchmodat(int dirfd, const char *pathname, mode_t mode, int flags)
```

- `dirfd` – file descriptor of directory
- `pathname` – pointer to string with file in question
- `mode` – permissions mask
- `flags` – see `chmod`

Returns zero on success.

## faccessat

Check user's permissions for a given file relative to a directory file descriptor.

```
int faccessat(int dirfd, const char *pathname, int mode, int flags)
```

- `dirfd` – file descriptor of directory
- `pathname` – pointer to string with file in question

- mode – specify check to perform
- flags – see access

Returns zero if permissions are granted.

## pselect6

Synchronous I/O multiplexing. Works just like `select` with a modified timeout and signal mask.

```
int pselect6(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,  
const struct timespec *timeout, const sigset_t *sigmask)
```

- nfds – number of file descriptors to monitor (add 1)
- readfds – fixed buffer with list of file descriptors to wait for read access
- writefds – fixed buffer with list of file descriptors to wait for write access
- exceptfds – fixed buffer with list of file descriptors to wait for exceptional conditions
- timeout – timeval structure with time to wait before returning
- sigmask – pointer to signal mask

Returns number of file descriptors contained in returned descriptor sets.

## ppoll

Wait for an event on a file descriptor like `poll` but allows for a signal to interrupt timeout.

```
int ppoll(struct pollfd *fds, nfds_t nfds, const struct timespec *timeout_ts,  
const sigset_t *sigmask)
```

- fds – pointer to an array of `pollfd` structures (described below)
- nfds – number of `pollfd` items in the `fds` array

- `timeout_ts` – sets the number of milliseconds the syscall should block (negative forces `poll` to return immediately)
- `sigmask` – signal mask

Returns number of structures having nonzero `revents` fields, or zero upon timeout.

## unshare

Disassociate parts of process execution context.

```
int unshare(int flags)
```

- `flags` – define behavior of call

## flags

- `CLONE_FILES` – unsquare file descriptor table so calling process no longer shares file descriptors with other processes
- `CLONE_FS` – unshare file system attributes so calling process no longer shares its root or current directory, or umask with other processes
- `CLONE_NEWIPC` – unshare System V IPC namespace so calling process has private copy of System V IPC namespace not shared with other processes
- `CLONE_NEWWNET` – unshare network namespace so calling process is moved to a new network namespace not shared with other processes
- `CLONE_NEWNS` – unsquare mount namespace
- `CLONE_NEWUTS` – unsquare UTS IPC namespace
- `CLONE_SYSVSEM` – unshare System V sempaphore undo values

## set\_robust\_list

Set list of robust futexes.

```
long set_robust_list(struct robust_list_head *head, size_t len)
```

- pid – thread/process id, or if 0 current process id is used
- head – pointer to location of list head
- len\_ptr – length of head\_ptr

Returns zero on success.

## get\_robust\_list

Get list of robust futexes.

```
long get_robust_list(int pid, struct robust_list_head **head_ptr, size_t *len_ptr)
```

- pid – thread/process id, or if 0 current process id is used
- head – pointer to location of list head
- len\_ptr – length of head\_ptr

Returns zero on success.

## splice

Splice data to/from a pipe.

```
splice(int fd_in, loff_t *off_in, int fd_out, loff_t *off_out, size_t len, unsigned int flags)
```

- fd\_in – file descriptor referring to a pipe for input

- `fd_out` – file descriptor referring to a pipe for output
- `off_in` – null if `fd_in` refers to a pipe, otherwise points to offset for read
- `off_out` – null if `fd_out` refers to a pipe, otherwise points to offset for write
- `len` – total bytes to transfer
- `flags` – defines additional behavior related to syscall

Returns number of bytes spliced to or from pipe.

## flags

- `SPLICE_F_MOVE` – try to move pages instead of copying
- `SPLICE_F_NONBLOCK` – try not to block I/O
- `SPLICE_F_MORE` – advise kernel that more data coming in subsequent splice
- `SPLICE_F_GIFT` – only for `vmsplice`, gift user pages to kernel

## tee

Duplicate pipe content.

```
tee(int fd_in, int fd_out, size_t len, unsigned int flags)
```

- `fd_in` – file descriptor referring to a pipe for input
- `fd_out` – file descriptor referring to a pipe for output
- `len` – total bytes to transfer
- `flags` – defines additional behavior related to syscall (see flags for `splice`)

Returns number of bytes duplicated between pipes.

## sync\_file\_range

Sync filesegment with disk.

```
int sync_file_range(int fd, off64_t offset, off64_t nbytes, nsigned int flags)
```

- fd – file descriptor of file in question
- offset – offset to begin sync
- nbytes – number of bytes to sync
- flags – defines additional behavior

Returns zero on success.

## flags

- SYNC\_FILE\_RANGE\_WAIT\_BEFORE – wait after write of all pages in range already submitted to device driver before performing any write
- SYNC\_FILE\_RANGE\_WRITE – write all dirty pages in range already not submitted for write
- SYNC\_FILE\_RANGE\_WAIT\_AFTER – wait after write of all pages in range before performing any write

## vmsplice

Splice user pages into pipe.

```
ssize_t vmsplice(int fd, const struct iovec *iov, unsigned long nr_segs, unsigned int flags)
```

- fd – file descriptor of pipe
- iov – pointer to array of iovec structures
- nr\_segs – ranges of user memory

- flags – defines additional behavior (see splice)

Return number of bytes transferred into pipe.

## move\_pages

Move pages of process to another node.

```
long move_pages(int pid, unsigned long count, void **pages, const int
*nodes, int *status, int flags)
```

- pid – process id
- pages – array of pointers to pages to move
- nodes – array of integers specifying location to move each page
- status – array of integers to receive status of each page
- flags – defines additional behavior

Returns zero on success.

## flags

- MPOL\_MF\_MOVE – move only pages in exclusive use
- MPOL\_MF\_MOVE\_ALL – pages shared between multiple processes can also be moved

## utimensat

Change timestamps with nanosecond precision.

```
int utimensat(int dirfd, const char *pathname, const struct timespec times[2], int flags)
```

- `dirfd` – directory file descriptor
- `pathname` – pointer to string with path of file
- `times` – array of timestamps, where `times[0]` is new last access time and `times[1]` is new last modification time
- `flags` – if `AT_SYMLINK_NOFOLLOW` specified, update timestamps on symlink

Returns zero on success.

## epoll\_pwait

Wait for I/O event on epoll file descriptor. Same as `epoll_wait` with a signal mask.

```
int epoll_pwait(int epfd, struct epoll_event *events, int maxevents, int timeout,
const sigset_t *sigmask)
```

- `epfd` – epoll file descriptor
- `events` – pointer to `epoll_event` structure with events available to calling process
- `maxevents` – maximum number of events, must be greater than zero
- `timeout` – timeout in milliseconds
- `sigmask` – signal mask to catch

Returns number of file descriptors ready for requested I/O or zero if timeout occurred before any were available.

## signalfd

Create file descriptor that can receive signals.

```
int signalfd(int fd, const sigset_t *mask, int flags)
```

- fd – if -1, create new file descriptor, otherwise use existing file descriptor
- mask – signal mask
- flags – set to SFD\_NONBLOCK to assign O\_NONBLOCK on new file descriptor, or SFD\_CLOEXEC to set FD\_CLOEXEC flag on new file descriptor

Returns file descriptor on success.

## timerfd\_create

Create timer that notifies a file descriptor.

```
int timerfd_create(int clockid, int flags)
```

- clockid – specify CLOCK\_REALTIME or CLOCK\_MONOTONIC
- flags – set to TFD\_NONBLOCK to assign O\_NONBLOCK on new file descriptor, or TFD\_CLOEXEC to set FD\_CLOEXEC flag on new file descriptor

Returns new file descriptor.

## eventfd

Create file descriptor for event notification.

```
int eventfd(unsigned int initval, int flags)
```

- initval – counter maintained by kernel
- flags – define additional behavior

Returns new eventfd file descriptor.

## flags

- EFD\_CLOEXEC – set close-on-exec flag on new file descriptor (FD\_CLOEXEC)
- EFD\_NONBLOCK – set O\_NONBLOCK on new file descriptor, saving extra call to fcntl to set this status
- EFD\_SEMAPHORE – perform semaphore-like semantics for reads from new file descriptor

## fallocate

Allocate file space.

```
int fallocate(int fd, int mode, off_t offset, off_t len)
```

- fd – file descriptor in question
- mode – defines behavior
- offset – starting range of allocation
- len – length of allocation

## mode

- FALLOC\_FL\_KEEP\_SIZE – do not change file size even if offset+len is greater than the original file size
- FALLOC\_FL\_PUNCH\_HOLE – deallocate space in specified range, zeroing blocks

## timerfd\_settime

Arms or disarms timer referenced by fd.

```
int timerfd_settime(int fd, int flags, const struct itimerspec *new_value,  
                    struct itimerspec *old_value)
```

- fd – file descriptor
- flags – set to 0 to start relative timer, or TFD\_TIMER\_ABSTIME to use absolute timer
- new\_value – pointer to itimerspec structure to set value
- old\_value – pointer to itimerspec structure to receive previous value after successful update

Returns zero on success.

## timerfd\_gettime

Get current setting of timer referenced by fd.

```
int timerfd_gettime(int fd, struct itimerspec *curr_value)
```

- fd – file descriptor
- curr\_value – pointer to itimerspec structure with current timer value

Returns zero on success.

## accept4

Same as accept syscall.

## signalfd4

Same as signalfd syscall.

## eventfd2

Same as `eventfd` without `flags` argument.

## **epoll\_create1**

Same as `epoll_create` without `flags` argument.

## **dup3**

Same as `dup2` except calling program can force close-on-exec flag to be set on new file descriptor.

## **pipe2**

Same as `pipe`.

## **inotify\_init1**

Same as `inotify_init` without `flags` argument.

## **preadv**

Same as `readv` but adds `offset` argument to mark start of input.

## **pwritev**

Same as `writev` but adds `offset` argument to mark start of output.

## **rt\_tgsigqueueinfo**

Not intended for application use. Instead, use `rt_sigqueue`.

## perf\_event\_open

Start performance monitoring.

```
int perf_event_open(struct perf_event_attr *attr, pid_t pid, int cpu, int group_fd,  
unsigned long flags)
```

- attr – pointer to `perf_event_attr` structure for additional configuration
- pid – process id
- cpu – cpu id
- group\_fd – create event groups
- flags – defines additional behavior options

```
struct perf_event_attr {  
    __u32      type;          /* event type */  
    __u32      size;          /* attribute structure size */  
    __u64      config;        /* type-specific configuration */  
  
    union {  
        __u64 sample_period;   /* sampling period */  
        __u64 sample_freq;     /* sampling frequency */  
    };  
  
    __u64      sample_type;   /* specify values included in sample */  
    __u64      read_format;   /* specify values returned in read */  
  
    __u64      disabled;      : 1,   /* off by default */  
    inherit:       : 1,   /* inherited by children */
```

```
pinned          : 1, /* must always be on PMU */
exclusive       : 1, /* only group on PMU */
exclude_user    : 1, /* don't count user */
exclude_kernel  : 1, /* don't count kernel */
exclude_hv      : 1, /* don't count hypervisor */
exclude_idle    : 1, /* don't count when idle */
mmap            : 1, /* include mmap data */
comm             : 1, /* include comm data */
freq             : 1, /* use freq, not period */
inherit_stat    : 1, /* per task counts */
enable_on_exec  : 1, /* next exec enables */
task             : 1, /* trace fork/exit */
watermark       : 1, /* wakeup_watermark */
precise_ip      : 2, /* skid constraint */
mmap_data       : 1, /* non-exec mmap data */
sample_id_all   : 1, /* sample_type all events */
exclude_host    : 1, /* don't count in host */
exclude_guest   : 1, /* don't count in guest */
exclude_callchain_kernel : 1, /* exclude kernel callchains */
exclude_callchain_user  : 1, /* exclude user callchains */
__reserved_1 : 41;

union {
    __u32 wakeup_events; /* every x events, wake up */
    __u32 wakeup_watermark; /* bytes before wakeup */
};

__u32 bp_type; /* breakpoint type */

union {
    __u64 bp_addr; /* address of breakpoint*/
    __u64 config1; /* extension of config */
};
```

```
union {
    __u64 bp_len; /* breakpoint length */
    __u64 config2; /* extension of config1 */
};

__u64 branch_sample_type; /* enum perf_branch_sample_type */
__u64 sample_regs_user; /* user regs to dump on samples */
__u32 sample_stack_user; /* stack size to dump on samples */
__u32 __reserved_2; /* align to u64 */

};
```

Returns new open file descriptor on success.

## flags

- `PERF_FLAG_FD_NO_GROUP` – allows creating event as part of event group without a leader
- `PERF_FLAG_FD_OUTPUT` – reroute output from event to group leader
- `PERF_FLAG_PID_CGROUP` – activate per-container full system monitoring

## recvmsg

Receive multiple messages on a socket using single syscall.

```
int recvmsg(int sockfd, struct msghdr *msgvec, unsigned int vlen, unsigned int flags,
           struct timespec *timeout)
```

- `sockfd` – socket file descriptor
- `msgvec` – pointer to array of `msghdr` structures
- `vlen` -size of `msgvec` array

- flags – specify flags from `recvmsg` or specify `MSG_WAITFORONE` to activate `MSG_DONTWAIT` after receipt of first message
- timeout – pointer to `timespec` structure specifying timeout

Returns number of messages received in `msgvec` on success.

## fanotify\_init

Create fanotify group.

```
int fanotify_init(unsigned int flags, unsigned int event_f_flags)
```

- flags – defines additional parameters
- event\_f\_flags – defines file status flags set on file descriptors created for fanotify events

Returns new file descriptor on success.

## flags

- `FAN_CLASS_PRE_CONTENT` – allow receipt of events notifying access or attempted access of a file before containing final content
- `FAN_CLASS_CONTENT` – allow receipt of events notifying access or attempted access of a file containing final content
- `FAN_REPORT_FID` – allow receipt of events containing info about filesystem related to an event
- `FAN_CLASS_NOTIF` – default value, allowing only for receipt of events notifying file access

## event\_f\_flags

- O\_RDONLY – read-only access
- O\_WRONLY – write-only access
- O\_RDWR – read/write access
- O\_LARGEFILE – support files exceeding 2 GB
- O\_CLOEXEC – enable close-on-exec flag for file descriptor

## fanotify\_mark

Add/remote/modify a fanotify mark on a file.

```
int fanotify_mark(int fanotify_fd, unsigned int flags, uint64_t mask,  
int dirfd, const char *pathname)
```

- fanotify\_fd – file descriptor from fanotify\_init
- flags – defines additional behavior
- mask – file mask
- dirfd – use depends on flags and pathname, see dirfd below

Returns zero on success.

## dirfd

- If pathname is NULL, dirfd is a file descriptor to be marked
- If pathname is NULL and dirfd is AT\_FDCWD then current working directory is marked
- If pathname is an absolute path, dirfd is ignored
- If pathname is a relative path and dirfd is not AT\_FDCWD, then pathname and dirfd define the file to be marked

- If `pathname` is a relative path and `dirfd` is `AT_FDCWD`, then `pathname` is used to determine file to be marked

## flags

- `FAN_MARK_ADD` – events in `mask` are added to mark or ignore mask
- `FAN_MARK_REMOVE` – events in `mask` are removed from mark or ignore mask
- `FAN_MARK_FLUSH` – remove all masks for filesystems, for mounts, or all marks for files and directories from fanotify group
- `FAN_MARK_DONT_FOLLOW` – if `pathname` is a symlink, mark link instead of file it refers
- `FAN_MARK_ONLYDIR` – if object marked is not a directory, then raise error
- `FAN_MARK_MOUNT` – mark mount point specified by `pathname`
- `FAN_MARK_FILESYSTEM` – mark filesystem specified by `pathname`
- `FAN_MARK_IGNORED_MASK` – events in `mask` will be added or removed from ignore mask
- `FAN_MARK_IGNORED_SURV_MODIFY` – ignore mask will outlast modify events
- `FAN_ACCESS` – create event when file or dir is accessed
- `FAN MODIFY` – create event when file is modified
- `FAN_CLOSE_WRITE` – create event when file that is writable is closed
- `FAN_CLOSE_NOWRITE` – create event when a file that is read-only or a directory is closed
- `FAN_OPEN` – create event when file or dir opened
- `FAN_OPEN_EXEC` – create event when file is opened to be executed
- `FAN_ATTRIB` – create event when file or dir metadata is changed
- `FAN_CREATE` – create event when file or dir is created in marked directory
- `FAN_DELETE` – create event when file or dir is deleted in marked directory
- `FAN_DELETE_SELF` – create event when marked file or dir is deleted

- FAN\_MOVED\_FROM – create event when file or dir is moved in a marked directory
- FAN\_MOVED\_TO – create event when file or dir has been moved to a marked directory
- FAN\_MOVE\_SELF – create event when marked file or directory is moved
- FAN\_Q\_OVERFLOW – create event when overflow of event queue occurs
- FAN\_OPEN\_PERM – create event when a process requests permission to open file or directory
- FAN\_OPEN\_EXEC\_PERM – create event when a process requests permission to open a file to execute
- FAN\_ACCESS\_PERM – create event when a process requests permission to read a file or directory
- FAN\_ONDIR – create events for directories themselves are accessed
- FAN\_EVENT\_ON\_CHILD – create events applying to the immediate children of marked directories

## name\_to\_handle\_at

Returns file handle and mount ID for file specified by `dirfd` and `pathname`.

```
int name_to_handle_at(int dirfd, const char *pathname, struct file_handle  
*handle, int *mount_id, int flags)
```

- `dirfd` – directory file descriptor
- `pathname` – pointer to string with full path to file
- `file_handle` – pointer to `file_handle` structure
- `mount_id` – pointer to filesystem mount containing `pathname`

Returns zero on success and `mount_id` is populated.

## open\_by\_handle\_at

Opens file corresponding to handle that is returned from `name_to_handle_at` syscall.

```
int open_by_handle_at(int mount_fd, struct file_handle *handle, int flags)
```

- `mount_fd` – file descriptor
- `handle` – pointer to `file_handle` structure
- `flags` – same flags for `open` syscall

```
struct file_handle {  
    unsigned int handle_bytes; /* size of f_handle (in/out) */  
    int handle_type; /* type of handle (out) */  
    unsigned char f_handle[0]; /* file id (sized by caller) (out) */  
};
```

Returns a file descriptor.

## syncfs

Flush filesystem cache specified by a file descriptor.

```
int syncfs(int fd)
```

- `fd` – file descriptor residing on disk to flush

Returns zero on success.

## sendmmsg

Send multiple messages via socket.

```
int sendmmsg(int sockfd, struct mmsghdr *msgvec, unsigned int vlen, int flags)
```

- `sockfd` – file descriptor specifying socket
- `msgvec` – pointer to `mmsghdr` structure
- `vlen` – number of messages to send
- `flags` – flags defining operation (same as `sendto` flags)

```
struct mmsghdr {  
    struct msghdr msg_hdr; /* header of message */  
    unsigned int msg_len; /* bytes to transmit */  
};
```

Returns number of messages sent from `msgvec`.

## setns

Reassociate a thread with namespace.

```
int setns(int fd, int nstype)  
• fd – file descriptor specifying a namespace  
• nstype – specify type of namespace (0 allows any namespace)
```

Returns zero on success.

## nsflag

- `CLONE_NEWCGROUP` – file descriptor must reference cgroup namespace

- CLONE\_NEWIPC – file descriptor must reference IPC namespace
- CLONE\_NEWWNET – file descriptor must reference network namespace
- CLONE\_NEWNS – file descriptor must reference a mount namespace
- CLONE\_NEWPID – file descriptor must reference descendant PID namespace
- CLONE\_NEWUSER – file descriptor must reference user namespace
- CLONE\_NEWUTS – file descriptor must reference UTS namespace

## getcpu

Return CPU/NUMA node for calling process or thread.

```
int getcpu(unsigned *cpu, unsigned *node, struct getcpu_cache *tcache)
```

- cpu – pointer to the CPU number
- node – pointer to the NUMA node number
- tcache – set to NULL (no longer used)

Returns zero on success.

## process\_vm\_readv

Copy data between a remote (another) process and the local (calling) process.

```
ssize_t process_vm_readv(pid_t pid, const struct iovec *local iov, unsigned long liovcnt,  
const struct iovec *remote iov, unsigned long riovcnt, unsigned long flags)
```

- pid – source process ID
- local iov – pointer to iovec structure with details about local address space
- liovcnt – number of elements in local iov

- `remote iov` – pointer to `iovec` structure with details about remote address space
- `riovcnt` – number of elements in `remote iov`
- `flags` – unused, set to 0

Returns number of bytes read.

## process\_vm\_writev

Copy data from the local (calling) process to a remote (another) process.

```
ssize_t process_vm_writev(pid_t pid, const struct iovec *local iov, unsigned long liovcnt,  
const struct iovec *remote iov, unsigned long riovcnt, unsigned long flags)
```

- `pid` – source process ID
- `local iov` – pointer to `iovec` structure with details about local address space
- `liovcnt` – number of elements in `local iov`
- `remote iov` – pointer to `iovec` structure with details about remote address space
- `riovcnt` – number of elements in `remote iov`
- `flags` – unused, set to zero

```
struct iovec {  
    void *iov_base; /* start address */  
    size_t iov_len; /* bytes to transfer */  
};
```

Returns number of bytes written.

## kcmp

Compare two processes to see if they share resources in the kernel.

```
int kcmp(pid_t pid1, pid_t pid2, int type, unsigned long idx1, unsigned long idx2)
```

- pid1 – the first process ID
- pid2 – the second process ID
- type – type of resource to compare
- idx1 – flag-specific resource index
- idx2 – flag-specific resource index

Returns zero if processes share the same resource.

## type flags

- KCMP\_FILE – check if file descriptors specified in idx1 and idx2 are shared by both processes
- KCMP\_FILES – check if the two processes share the same set of open file descriptors (idx1 and idx2 are not used)
- KCMP\_FS – check if the two processes share the same filesystem information (for example, the filesystem root, mode creation mask, working directory, etc.)
- KCMP\_I0 – check if processes share the same I/O context
- KCMP\_SIGHAND – check if processes share same table of signal dispositions
- KCMP\_SYSVSEM – check if processes share same semaphore undo operations
- KCMP\_VM – check if processes share same address space
- KCMP\_EPOLL\_TFD – check if file descriptor referenced in idx1 of process pid1 is present in epoll referenced by idx2 of process pid2, where idx2 is a structure kcmp\_epoll\_slot describing target

## file

```
struct kcmp_epoll_slot {  
    __u32efd;  
    __u32tfd;  
    __u64toff;  
};
```

## finit\_module

Load module into kernel with module file specified by file descriptor.

```
int finit_module(int fd, const char *param_values, int flags)  
• fd – file descriptor of kernel module file to load  
• param_values – pointer to string with parameters for kernel  
• flags – flags for module load
```

Returns zero on success.

## flags

- MODULE\_INIT\_IGNORE\_MODVERSIONS – ignore symbol version hashes
- MODULE\_INIT\_IGNORE\_VERMAGIC – ignore kernel version magic

#System Calls

## ABOUT THE AUTHOR

### Robert Oliver

Writer, System Admin, Full Stack Developer, Philosopher.

<https://rwo2.com>

[View all posts](#)



## RELATED LINUX HINT POSTS

[Bitwise Operator in C](#)

[Input Output Instructions in C](#)

[How to Check If a Number Is Even  
in C](#)

[File Opening Modes in C](#)

[Different Types of Function](#)

[Constant in C Language](#)

[Use of strcpy\(\), strcmp\(\), and  
strcat\(\)](#)

Linux Hint LLC, editor@linuxhint.com  
1309 S Mary Ave Suite 210, Sunnyvale, CA 94087

---

AN ELITE CAFEMEDIA PUBLISHER