SystemCalls

API Reference 7.0.1

Generated by Doxygen 1.5.8

Fri Sep 4 10:33:34 2020

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Module Documentation

2.1 specific thread system calls

Architecture specific system calls related to threading.

Functions

- int get_thread_area (UserVA l_user_desc)

  Get a thread local storage (TLS).
- int set_thread_area (UserVA l_user_desc)

  Set a thread local storage (TLS).

2.1.1 Function Documentation

2.1.1.1 int get_thread_area (UserVA *l_user_desc*)

Note:

Not supported on arm64.

Parameters:

← ***l_user_desc*** descriptor

Returns:

LINUX_EINVAL.

Note:

syscall 244 (32-bit)
Not supported in 64-bit

Parameters:

← ***l_user_desc*** descriptor

Returns:

0 on success, and errno on failure

2.1.1.2 int set_thread_area (UserVA *l_user_desc*)

Note:

Not supported on arm64.

Parameters:

← ***l_user_desc*** descriptor

Returns:

LINUX_EINVAL.

Note:

syscall 243 (32-bit)
Not supported in 64-bit

Parameters:

← ***l_user_desc*** descriptor

Returns:

0 on success, and errno on failure

2.2 Miscellaneous system calls

Miscellaneous System calls.

Functions

- int fork (void)

  Creates a new world.

- int getpriority (int which, int who)

  Gets the scheduling priority.

- int64 getrlimit (int resource, structr rlimit userRlim)

  get resource limits

- int64 getrusage (int who, struct rusage ∗userRUsage)

  get information about resource utilization

- int iopl (uint32 level)

  Change IOPL in eflags.

- int64 prctl (int option, unsigned long arg2, unsigned long arg3, unsigned long arg4, unsigned long arg5)

  Partially implements the prctl syscall.

- long prlimit64 (pid_t pid, int resource, struct rlimit64 ∗newRlim, struct rlimit64 ∗oldRlim)

  process resource limits as 64 bit values

- long ptrace (enum __ptrace_request req, pid_t pid, void ∗addr, void ∗data)

  Trace a userworld.

- int64 reboot (int magic1, int magic2, int flag, void ∗arg)

  Reboots the system.

- int64 setrlimit (int resource, struct rlimit ∗userRLimit)

  set resource limits

2.2.1 Function Documentation

2.2.1.1 int fork (void)

Note:

fork, syscall 2
This syscall is not fully supported. See limitations below.
Fork'ing world needs to be single-threaded.
FD state is not inherited (offset) not shared (offset, fcntl).
Timer state is not inherited.
Signal state is not inherited (actions/handlers are).

Some objects are not shared or inherited (sockets, shared mmaps).
Floating point state is not inherited.
Not supported for 64-bit apps.

Returns:

0 on success or ENOMEM

2.2.1.2 int getpriority (int *which*, int *who*)

Note:

getpriority, syscall 96 (32-bit) and 140 (64-bit)
This syscall is not supported (no-op).

Parameters:

← ***which***

← ***who***

Returns:

Appropriate error

2.2.1.3 int64 getrlimit (int *resource*, structr rlimit *userRlim*)

Note:

getrlimit, syscall 191(32-bit) and 97(64-bit)

Parameters:

← ***resource*** Specific resource that is being queried

→ ***userRlim*** Pointer to a results strct rlimit structure

Returns:

Returns 0 on success, -1 otherwise.

2.2.1.4 int64 getrusage (int *who*, struct rusage ∗ *userRUsage*)

Note:

getrusage, syscall 77(32-bit) and 98(64-bit)
This syscall is not fully supported. See limitations below
1. Only RUSAGE_SELF is supported.
2. Only the ru_utime field in the results structure is filled up.

Parameters:

← ***who*** Which resource to query, only RUSAGE_SELF is supported

← ***userRUsage*** results structure.

Returns:

Returns 0 on success, -1 otherwise.

2.2.1.5 int iopl (uint32 *level*)

Note:

Not supported on arm64.

Parameters:

← ***level*** I/O privilege level

Returns:

LINUX_EINVAL.

Note:

Only allowed if ESX is running in a VM

Parameters:

← ***level*** I/O privilege level

Returns:

Returns 0 on success, -1 otherwise.

2.2.1.6 int64 prctl (int *option*, unsigned long *arg2*, unsigned long *arg3*, unsigned long *arg4*, unsigned long *arg5*)

Note:

prctl, syscall 172
This syscall is not fully supported. See limitations below.

Supported options: PR_[GET|SET]_NAME PR_SET_SECCOMP PR_GET_FD_PATH UW_PRCTL_-SET_FSS_TIMEOUT UW_PRCTL_GET_FSS_TIMEOUT UW_PRCTL_GET_EXE_PATH UW_-PRCTL_GET_FD_PATH UW_PRCTL_REALPATH UW_PRCTL_SET_WORLD_OPID UW_PRCTL_-GET_WORLD_OPID_HASH UW_PRCTL_GET_WORLD_OPID_STR UW_PRCTL_GET_DATA_-START UW_PRCTL_SET_RES_GROUPID UW_PRCTL_GET_RES_GROUPID UW_PRCTL_GET_-RES_PARGROUPID UW_PRCTL_GET_RES_GROUPSTATE UW_PRCTL_GET_MAX_THREAD UW_PRCTL_GET_MAX_FD

Parameters:

← ***option***
← ***arg2***
← ***arg3***
← ***arg4***
← ***arg5***

Returns:

0 on success, or appropriate error otherwise

2.2.1.7 long prlimit64 (pid_t *pid*, int *resource*, struct rlimit64 ∗ *newRlim*, struct rlimit64 ∗ *oldRlim*)

Note:

prlimit, syscall 340 (32-bit) and 302 (64-bit)

Parameters:

← ***pid*** Specified process id
← ***resource*** Specified resource
← ***newRlim*** Pointer to the new rlimit
→ ***oldRlim*** Pointer to the old rlimit

Returns:

Returns 0 on success, -1 otherwise.

2.2.1.8 long ptrace (enum __ptrace_request *req*, pid_t *pid*, void ∗ *addr*, void ∗ *data*)

Note:

ptrace, syscall 26
This syscall is not fully supported. See limitations below.
PTRACE_SETFPREGS, PTRACE_SETSIGINFO, PTRACE_SETOPTIONS, PTRACE_GETEVENTMSG, PTRACE_SYSEMU_SINGLESTEP and PTRACE_SYSEMU are not supported.

Parameters:

← ***req***
← ***pid***
← ***addr***
→ ***data***

Returns:

0 on success, or appropriate error otherwise

2.2.1.9 int64 reboot (int *magic1*, int *magic2*, int *flag*, void $*$ *arg*)

Note:

reboot, syscall 88
This syscall is not fully supported. See limitations below.
LINUX_REBOOT_MAGIC2C is not supported.
LINUX_REBOOT_CMD_RESTART2, LINUX_REBOOT_CMD_CAD_ON and
LINUX_REBOOT_CMD_CAD_OFF are not supported.

Parameters:

← ***magic1***

← ***magic2***

← ***flag***

← ***arg***

Returns:

0 on success, or appropriate error otherwise

2.2.1.10 int64 setrlimit (int *resource*, struct rlimit $*$ *userRLimit*)

Note:

setrlimit, syscall 75(32-bit) and 160(64-bit)
This syscall is not fully supported. See limitations below
Only the following resources are supported. All others resources settings are ignored.
RLIMIT_STACK
RLIMT_NOFILE
RLIMIT_CORE

Parameters:

← ***resource*** Specific resource that is being set

← ***userRLimit*** Pointer to a the new value of the resource

Returns:

Returns 0 on success, -1 otherwise.

2.3 AIO related sys calls

Functions

- int io_cancel (UserAIO_ContextID ctx, UserVAConst iocb, UserVA event)

  Cancels the specified IO in the context.

- int io_destroy (UserAIO_ContextID ctx)

  Destroys the specified aysnc io ctx.

- int io_getevents (UserAIO_ContextID ctx, long min_num_requests, long num_requests, UserVA events, UserVA timeout)

  Gets any events associated with the async io ctx.

- int io_setup (uint32 nr_events, UserVA ctx)

  Create a context to be able to issue async io requests.

- int io_submit (UserAIO_ContextID ctx, long num_requests, UserVAConst cbs)

  Submits the specfied IOs as an async request.

2.3.1 Function Documentation

2.3.1.1 int io_cancel (UserAIO_ContextID *ctx*, UserVAConst *iocb*, UserVA *event*)

Note:

io_cancel, syscall 249 (32-bit) and 210 (64-bit).

Parameters:

← ***ctx*** an aync io context

← ***iocb*** the io to cancel

→ ***event*** an address to store the event at

Returns:

zero on success, errno on failure

2.3.1.2 int io_destroy (UserAIO_ContextID *ctx*)

Note:

io_destroy, syscall 246 (32-bit) and 207 (64-bit).
Blocks until all inflight IOs are completed

Parameters:

← ***ctx*** an aync io context

Returns:

zero on success, errno on failure

2.3.1.3 int io_getevents (UserAIO_ContextID *ctx*, long *min_num_requests*, long *num_requests*, UserVA *events*, UserVA *timeout*)

Note:

io_getevents, syscall 247 (32-bit) and 208 (64-bit).

Parameters:

$\leftarrow$ ***ctx*** an aync io context

$\leftarrow$ ***min_num_requests*** min events to read

$\leftarrow$ ***num_requests*** max events to read

$\rightarrow$ ***events*** an address to store the events at

$\leftarrow$ ***timeout*** a timespec to specify the timeout

Returns:

number of events read on success, errno on failure

2.3.1.4 int io_setup (uint32 *nr_events*, UserVA *ctx*)

Note:

io_setup, syscall 245 (32-bit) and 206 (64-bit).
Only IOCB_CMD_PREAD and IOCB_CMD_PWRITE are supported

Parameters:

$\leftarrow$ ***nr_events*** number of events supported by this ctx

$\leftrightarrow$ ***ctx*** an aync io context pointer

Returns:

zero on success, errno on failure

2.3.1.5 int io_submit (UserAIO_ContextID *ctx*, long *num_requests*, UserVAConst *cbs*)

Note:

io_submit, syscall 248 (32-bit) and 209 (64-bit).

Parameters:

$\leftarrow$ ***ctx*** an aync io context

← ***num_requests*** the number of requests being issued

← ***cbs*** an array of IO command block pointers

Returns:

number of IOs submitted on success, errno on failure

2.4 descriptor related system calls.

Functions

- int64 access (UserVAConstuserPath, int32 mode)

  Check real user's permissions for a file.

- int64 chdir (UserVAConstuserPath)

  Change working directory.

- int64 chmod (UserVAConstuserPath, IdentityMode mode)

  Change permissions of a file.

- int64 chown (UserVAConstuserPath, LinuxUID uid, LinuxGID gid)

  Change ownership of a file.

- int64 chroot (const char ∗path)

  Change root directory.

- int64 close (LinuxFd fd)

  Close a file.

- int64 creat (UserVAConstuserPath, IdentityMode mode)

  Create a file.

- int64 dup (LinuxFd fd)

  Remove a directory.

- int64 dup2 (LinuxFd from, LinuxFd to)

  Duplicate a file descriptor.

- int64 dup3 (LinuxFd from, LinuxFd to, uint32 flags)

  Duplicate a file descriptor.

- int64 epoll_create (int size)

  Create an epoll descriptor.

- int64 epoll_create1 (int flags)

  Create an epoll descriptor.

- int64 epoll_ctl (int epfd, int op, int fd, UserVA event)

  Modify epoll state.

- int64 epoll_pwait (int epfd, UserVA events, int maxevents, int timeout, UserVA sigmask)

  Wait for epoll events.

- int64 epoll_wait (int epfd, UserVA events, int maxevents, int timeout)

  Wait for epoll events.

- int64 eventfd2 (uint64 initVal, int flags) *create a file descriptor for event notification*

- int64 execve (UserVAConstuserPath, UserVAConstuserArgp, UserVAConstuserEnvp)

  Execute program.

- int faccessat (int dirfd, const char ∗userPath, int mode)

  Check real user's permissions for a file.

- int64 fchdir (LinuxFd fd)

  Change working directory.

- int64 fchmod (LinuxFd fd, IdentityMode mode)

  Change permissions of a file.

- int fchmodat (int dirfd, const char ∗userPath, mode_t mode)

  Change permissions of a file.

- int64 fchown (LinuxFd fd, LinuxUID uid, LinuxGID gid)

  Change ownership of a file.

- int fchownat (int dirfd, const char ∗path, uid_t uid, gid_t gid, int flags)

  Change ownership of a file.

- int64 fcntl64 (LinuxFd fd, uint32 cmd, int64 arg)

  Manipulate file descriptor.

- int64 fdatasync (LinuxFd fd)

  Flush in-memory state into device.

- int64 flock (LinuxFd fd, uint32 op)

  lock/unlock an open file

- int64 fstat (int fd, UserVA statbuf)

  Get file status.

- int fstat64 (LinuxFd fd, UserVAstatbuf)

  Get file status.

- int fstatat (int dirfd, const char ∗userPath, char ∗statbuf, int flags)

  Get file status.

- int64 fstatfs (LinuxFd fd, UserVAuserBuf)

  Get filesystem statistics.

- int fstatfs64 (LinuxFd fd, uint32 userBufSize, UserVA userBuf)

  get file system statistics

- int64 fsync (LinuxFd fd)

  Flush memory into the device.

- int64 ftruncate (LinuxFd fd, int64 length)

  Truncate a file to a specified length.

- int ftruncate32 (LinuxFd fd, int32 length)

  Truncate a file to a specified length.

- int ftruncate64 (LinuxFd fd, uint32 llow, int32 lhigh)

  Truncate a file to a specified length.

- int futimesat (int dirFd, const char ∗userPath, const struct timeval ∗userTimes)

  Change file access and modification timestamps.

- int64 getcwd (UserVAbuf, uint64 bufsize)

  Get current working directory.

- int64 getdents (LinuxFd fd, UserVAuserBuf, uint32 nbyte)

  get directory entries

- int64 getdents64 (LinuxFd fd, UserVAuserBuf, uint32 nbyte)

  get directory entries

- int64 getxattr (UserVAConstuserPath, UserVAConstuserName, UserVAuserValue, int64 size)

  set an extended attribute value

- int64 ioctl (LinuxFd fd, uint32 cmd, UserVA userData)

  ioctl - control device

- int64 lchown (UserVAConstuserPath, LinuxUID uid, LinuxGID gid)

  Change ownership of a file.

- int64 link (UserVAConstoldPath, UserVAConstnewPath)

  Make a new link to a file.

- int linkat (int dirfd, const char ∗oldPath, const char ∗newPath)

  Make a new link to a file.

- int64 LinuxFileDesc_SymlinkAt (UserVAConstuserTo, LinuxFd userPathDirFD, UserVAConstuser-Path)

  Creates a symbolic link.

- int llseek (LinuxFd fd, int32 ohigh, uint32 olow, UserVAuserRes, uint32 whence)

  Set file offset.

- int64 lseek (LinuxFd fd, int64 offset, int32 whence)

  Set file offset.

- int64 lstat (UserVAConst userPath, UserVA statbuf)

  Get file status.

- int64 lstat64 (UserVAConstuserPath, UserVAstatbuf)

Get file status.

- int64 mkdir (UserVAConstuserPath, IdentityMode mode)

  create a directory

- int mkdirat (int dirfd, const char ∗userPath, mode_t mode)

  create a directory

- int64 mknod (UserVAConstuserPath, IdentityMode mode, uint64 devId)

  Creates a filesystem node (limited support).

- int mknodat (int dirFd, const char ∗userPath, mode_t mode, dev_t devId)

  rename a file

- int64 mount (UserVAConstuserSpecialfile, UserVAConstuserDir, UserVAConstfilesystemType, unsigned long mountflags, UserVAConstdata)

  Mount file system.

- int64 open (UserVAConstuserPath, uint32 flags, IdentityMode mode)

  Open a file.

- int openat (int dirfd, const char ∗userPath, int flags, mode_t mode)

  Open a file.

- int64 pipe (UserVA pipefds, uint32 flags)

  Create a pipe with extra flags.

- int64 pipe (UserVA pipefds)

  Create a pipe.

- int64 poll (UserVA userPollFds, uint64 nfds, int32 timeoutMillis)

  Wait for event on a file descriptor.

- int64 ppoll (UserVA userPollFds, uint64 nfds, struct timespec ∗timeout, const sigset_t ∗sigmask, size_t sigsetsize)

  Wait for event on a file descriptor.

- int64 pread64 (LinuxFd fd, UserVAuserBuf, uint64 nbyte, int64 offset)

  Read from a file descriptor starting at a give offset.

- int64 preadV (LinuxFd fd, UserVA userIovp, uint32 iovcnt, int64 offset)

  read data from multiple buffers starting at given offset

- int pselect (int n, fd_set ∗readfds, fd_set ∗writefds, fd_set ∗exceptfds, struct timeval ∗timeout, const sigset_t ∗sigmask)

  Synchronous I/O multiplexing.

- int64 pwrite64 (LinuxFd fd, UserVAConstuserBuf, uint64 nbyte, int64 offset)

  Write to a file descriptor starting at a give offset.

- int64 pwritev (LinuxFd fd, UserVA userIovp, uint32 iovcnt, int64 offset)

  write data from multiple buffers starting at given offset

- int64 read (LinuxFd fd, UserVA userBuf, uint32 nbyte)

  read from a file descriptor

- int64 readlink (UserVAConstuserPath, UserVAuserBuf, int64 inCount)

  read value of a symbolink link

- int readlinkat (int dirfd, const char ∗userPath, char ∗userBuf, size_t bufSize)

  read value of a symbolink link

- int64 readv (LinuxFd fd, UserVAuserIovp, uint32 iovcnt)

  read data from multiple buffers

- int64 rename (UserVAConstoldPath, UserVAConstnewPath)

  rename a file

- int renameat (int fromDirFd, const char ∗from, int toDirFd, const char ∗to)

  rename a file

- int64 rmdir (UserVAConstuserPath)

  Remove a directory.

- int select (int n, fd_set ∗readfds, fd_set ∗writefds, fd_set ∗exceptfds, struct timeval ∗timeout)

  synchronous I/O multiplexing

- int64 sendfile64 (LinuxFd out_fd, LinuxFd in_fd, UserVA offset, uint32 count)

  Write file data from the buffer cache directly to a socket.

- int64 setxattr (UserVAConstuserPath, UserVAConstuserName, UserVAuserValue, int64 size, int64 flags)

  set an extended attribute value

- int64 stat (UserVAConst userPath, UserVA statbuf)

  Get file status.

- int stat64 (UserVAConstuserPath, UserVAstatbuf)

  Get file status.

- int64 statfs (UserVAConstuserPath, UserVAuserBuf)

  Get filesystem statistics.

- int statfs64 (UserVAConstuserPath, uint32 userBufSize, UserVA userBuf)

  get file system statistics

- int64 symlink (UserVAConstuserTo, UserVAConstuserPath)

  Creates a symbolic link.

- int64 truncate (UserVAConstuserPath, int64 length)

Truncate a file to a specified length.

- int truncate32 (UserVAConstuserPath, int32 length)

  Truncate a file to a specified length.

- int truncate64 (UserVAConstuserPath, uint32 llow, int32 lhigh)

  Truncate a file to a specified length.

- int64 umask (uint32 newmask)

  set file mode creation mask

- int64 umount (UserVAConstuserDir, uint32 delayflag)

  Unmount file system.

- int64 unlink (UserVAConstuserPath)

  Delete a file.

- int unlinkat (int dirfd, const char ∗userPath, int flags)

  Delete a file.

- int utime (const char ∗userPath, const struct utimebuf ∗utimebuf)

  Change file last access and modification time.

- int utimensat (int dirFd, const char ∗userPath, const struct timeval ∗userTimes, int flags)

  Change file timestamps with nanosecond precision.

- int utimes (const char ∗userPath, const struct timeval ∗userTimes)

  Change file access and modification timestamps.

- int64 write (LinuxFd fd, UserVA64Const userBuf, uint64 nbyte)

  write to a file descriptor

- int64 writev (LinuxFd fd, UserVAuserIovp, uint32 iovcnt)

  write data from multiple buffers

2.4.1 Function Documentation

2.4.1.1 int64 access (UserVAConst *userPath*, int32 *mode*)

Note:

access, syscall 33 (32-bit) and 21 (64-bit)

Parameters:

← ***userPath*** pathname

← ***mode*** permission flag

Returns:

Checks whether the real (not effective!) uid of the cartel has access permission on the specified path. 0 on success, or appropriate Linux error code.

2.4.1.2 int64 chdir (UserVAConst *userPath*)

Note:

chdir, syscall 12 (32-bit) and 80 (64-bit)

Parameters:

← ***userPath*** pathname

Returns:

zero on success, errno on failure

2.4.1.3 int64 chmod (UserVAConst *userPath*, IdentityMode *mode*)

Note:

chmod, syscall 15 (32-bit) and 90 (64-bit)

Parameters:

← ***userPath*** pathname

← ***mode*** permission flag

Returns:

zero on success, errno on failure

2.4.1.4 int64 chown (UserVAConst *userPath*, LinuxUID *uid*, LinuxGID *gid*)

Note:

chown, syscall 212 (32-bit) and 92 (64-bit)

Parameters:

← ***userPath*** pathname

← ***uid*** user id

← ***gid*** group id

Returns:

zero on success, errno on failure

2.4.1.5 int64 chroot (const char ∗ *path*)

Note:

chroot, syscall 61 (32-bit) and 161 (64-bit)

Parameters:

← ***path***

Returns:

zero on success, -1 on failure, errno is set appropriately

2.4.1.6 int64 close (LinuxFd *fd*)

Note:

close, syscall 6 (32-bit) and 3 (64-bit)

Parameters:

← ***fd*** file descriptor

Returns:

zero on success, errno on failure

2.4.1.7 int64 creat (UserVAConst *userPath*, IdentityMode *mode*)

Note:

creat, syscall 8 (32-bit) and 85 (64-bit)
from man page; the function call "creat(path, mode)" shall be equivalent to "open(path, O_-WRONLY|O_CREAT|O_TRUNC, mode)"

Parameters:

← ***userPath*** pathname

← ***mode*** mode

Returns:

return file descriptor, -1 on error

2.4.1.8 int64 dup (LinuxFd *fd*)

Note:

dup, syscall 41 (32-bit) and 32 (64-bit)

Parameters:

← ***fd*** file descriptor

Returns:

new file descriptor on success, errno on failure

2.4.1.9 int64 dup2 (LinuxFd *from*, LinuxFd *to*)

Note:

dup2, syscall 63 (32-bit) and 33 (64-bit).

Parameters:

← ***from*** src fd

← ***to*** dst fd

Returns:

returns the new fd on success, errno on failure

2.4.1.10 int64 dup3 (LinuxFd *from*, LinuxFd *to*, uint32 *flags*)

Note:

dup3, O_CLOEXEC is not yet supported.

Parameters:

← ***from*** src fd

← ***to*** dst fd

← ***flags*** optional O_CLOEXEC

Returns:

returns the new fd on success, errno on failure

2.4.1.11 int64 epoll_create (int *size*)

Note:

epoll_create, syscall 254 (32-bit) and 213 (64-bit).
1. Limited support not all fd types support poll. 2. XXX add other limitations

Parameters:

← ***size*** unused

Returns:

fd on success, errno on failure

2.4.1.12 int64 epoll_create1 (int *flags*)

Note:

epoll_create1, EPOLL_CLOEXEC not supported today.
1. Limited support not all fd types support poll. 2. XXX add other limitations

Parameters:

← ***flags*** optional EPOLL_CLOEXEC

Returns:

fd on success, errno on failure

2.4.1.13 int64 epoll_ctl (int *epfd*, int *op*, int *fd*, UserVA *event*)

Note:

epoll_ctl, syscall 255 (32-bit) and 233 (64-bit)
XXX add other limitations

Parameters:

← ***epfd*** epoll fd

← ***op*** operation

← ***fd*** fd

← ***event*** struct epoll_event

Returns:

0 on success, errno on failure

2.4.1.14 int64 epoll_pwait (int *epfd*, UserVA *events*, int *maxevents*, int *timeout*, UserVA *sigmask*)

Note:

epoll_pwait, don't currently support sigmask.

Parameters:

← ***epfd*** epoll fd

→ ***events*** array of struct epoll_event

← ***maxevents*** length of events array

← ***timeout*** milliseconds

← ***sigmask*** signals to mask

Returns:

return the number of events on success, error code otherwise

2.4.1.15 int64 epoll_wait (int *epfd*, UserVA *events*, int *maxevents*, int *timeout*)

Note:

epoll_wait, syscall 256 (32-bit) and 232 (64-bit)
Limited support : Supported flags include EPOLLIN EPOLLOUT EPOLLERR EPOLLHUP EPOLL-RDNORM EPOLLWRNORM EPOLLET. EPOLLPRI implemented as EPOLLIN. XXX add other limitations

Parameters:

← ***epfd*** epoll fd

→ ***events*** array of struct epoll_event

← ***maxevents*** length of events array

← ***timeout*** milliseconds

Returns:

return the number of events on success, error code otherwise

2.4.1.16 int64 eventfd2 (uint64 *initVal*, int *flags*)

Note:

eventfd2, syscall 328 (32-bit) and 290 (64-bit)
The object contains an unsigned 64-bit integer (uint64_t) for both
32-bit and 64-bit implementation.
EFD_CLOEXEC flag is not supported

Parameters:

← ***initVal*** initial value

← ***flags*** flags

Returns:

return the file descriptor on success, error code otherwise

2.4.1.17 int64 execve (UserVAConst *userPath*, UserVAConst *userArgp*, UserVAConst *userEnvp*)

Note:

execve, syscall 11 (32-bit) and 59 (64-bit).
Not fully supported :
Exec'ing world needs to be single-threaded native userworld Most of the
'lack of resources' errors will see exec succeed and the exec'ed
process die right away instead of reporting exec failure.
Does not handle #! (why isn't it the job of glibc ?)
Error case:% Linux ELF specific (EISDIR/ELIBBAD) errors are reported as
ENOEXEC Check for exclusive write access is not done (no ETXTBSY)

Parameters:

← ***userPath*** filename

← ***userArgp*** argv

← ***userEnvp*** envp

Returns:

Does not return on success, errno on failure

2.4.1.18 int faccessat (int *dirfd*, const char ∗ *userPath*, int *mode*)

Note:

faccessat, syscall 307 (32-bit) and 269 (64-bit)
faccessat, AT_FDCWD is the only supported value for dirfd.
faccessat, linux syscall only takes 3 parameters. Posix flags are implemented within glibc wrapper.

Parameters:

← ***dirfd*** relative paths interpreted from dirfd

← ***userPath*** pathname

← ***mode*** permission mode

Returns:

Checks whether the real (not effective!) uid of the cartel has access permission on the specified path.
0 on success, or appropriate Linux error code.

2.4.1.19 int64 fchdir (LinuxFd *fd*)

Note:

fchdir, syscall 133 (32-bit) and 81 (64-bit)

Parameters:

← ***fd*** file descriptor

Returns:

zero on success, errno on failure

2.4.1.20 int64 fchmod (LinuxFd *fd*, IdentityMode *mode*)

Note:

fchmod, syscall 94 (32-bit) and 91 (64-bit)

Parameters:

$\leftarrow$ ***fd*** file descriptor

$\leftarrow$ ***mode*** permission flag

Returns:

zero on success, errno on failure

2.4.1.21 int fchmodat (int *dirfd*, const char $*$ *userPath*, mode_t *mode*)

Note:

fchmodat, syscall 306 (32-bit) and 268 (64-bit)
fchmodat, AT_FDCWD is the only supported value for dirfd.
fchmodat, linux syscall only takes 3 parameters. Posix flags are implemented within glibc wrapper.

Parameters:

$\leftarrow$ ***dirfd*** relative paths interpreted from dirfd

$\leftarrow$ ***userPath*** pathname

$\leftarrow$ ***mode*** permission mask

Returns:

zero on success, errno on failure

2.4.1.22 int64 fchown (LinuxFd *fd*, LinuxUID *uid*, LinuxGID *gid*)

Note:

fchown, syscall 207 (32-bit) and 93 (64-bit)

Parameters:

$\leftarrow$ ***fd*** file descriptor

$\leftarrow$ ***uid*** user id

$\leftarrow$ ***gid*** group id

Returns:

zero on success, errno on failure

2.4.1.23 int fchownat (int *dirfd*, const char ∗ *path*, uid_t *uid*, gid_t *gid*, int *flags*)

Note:

fchownat, syscall 298 (32-bit) and 260 (64-bit)
fchownat, AT_FDCWD is the only supported value for dirfd.
fchownat, AT_SYMLINK_NOFOLLOW is the only supported flag.

Parameters:

← ***dirfd*** relative paths interpreted from dirfd

← ***path*** file path

← ***uid*** user id

← ***gid*** group id

← ***flags*** optional AT_SYMLINK_NOFOLLOW

Returns:

zero on success, errno on failure

2.4.1.24 int64 fcntl64 (LinuxFd *fd*, uint32 *cmd*, int64 *arg*)

Note:

fcntl64, syscall 221 (32-bit) and 72 (64-bit)
Limited support: 1.supported command : GETFD SETFD GETFL SETFL(not fully supported) DUPFD

Parameters:

← ***fd*** file descriptor

← ***cmd*** command

← ***arg*** argument

Returns:

Depends on cmd, errno on failure

2.4.1.25 int64 fdatasync (LinuxFd *fd*)

Note:

fdatasync, syscall 148 (32-bit) and 75 (64-bit).

Parameters:

← ***fd*** file descriptor

Returns:

zero on success, errno on failure

2.4.1.26 int64 flock (LinuxFd *fd*, uint32 *op*)

Note:

flock, syscall 143 (32-bit) and 73 (64-bit).
Supported operations : LOCK_SH LOCK_EX LOCK_UN

Parameters:

← ***fd*** file descriptor

← ***op*** operation

Returns:

zero on success, errno on failure

2.4.1.27 int64 fstat (int *fd*, UserVA *statbuf*)

Note:

fstat, syscall 108 (32-bit) and 5 (64-bit).
NOT SUPPORTED in 32-bit (ENOSYS)

Parameters:

← ***fd*** file descriptor

← ***statbuf*** dst buffer

Returns:

zero on success, errno on failure

2.4.1.28 int fstat64 (LinuxFd *fd*, UserVA *statbuf*)

Note:

fstat64, syscall 197 (32-bit)
limited support not all some fields are not supported in some file types.

Parameters:

← ***fd*** file descriptor

← ***statbuf*** dst buffer

Returns:

zero on success, errno on failure

2.4.1.29 int fstatat (int *dirfd*, const char ∗ *userPath*, char ∗ *statbuf*, int *flags*)

Note:

fstatat, syscall 300 (32-bit) and 262 (64-bit)
fstatat, AT_FDCWD is the only supported value for dirfd.
fstatat, AT_SYMLINK_NOFOLLOW only supported flag.

Parameters:

← ***dirfd*** relative paths interpreted from dirfd

← ***userPath*** pathname

← ***statbuf*** dst buffer

← ***flags*** optional flags

Returns:

zero on success, errno on failure

2.4.1.30 int64 fstatfs (LinuxFd *fd*, UserVA *userBuf*)

Note:

fstatfs, syscall 100 (32-bit) and 138 (64-bit)
Copies statfs information to user's buffer.
Limited support: pipes, sockets, fifo return ENOSYS
Not all file tyes supports this call

Parameters:

← ***fd*** file descriptor

→ ***userBuf*** destination buffer

Returns:

zero on success, errno on failure

2.4.1.31 int fstatfs64 (LinuxFd *fd*, uint32 *userBufSize*, UserVA *userBuf*)

Note:

fstatfs64, syscall 269 (32-bit). Not supported in 64-bit
Limited support : socket, pipe, fifo don't support this call (ENOSYS)

Parameters:

← ***fd*** file descriptor
← ***userBufSize*** buffer size
→ ***userBuf*** destination buffer

Returns:

0 on success, errno on failure

2.4.1.32 int64 fsync (LinuxFd *fd*)

Note:

fsync, syscall 118 (32-bit) and 74 (64-bit).

Parameters:

← ***fd*** file descriptor

Returns:

zero on success, errno on failure

2.4.1.33 int64 ftruncate (LinuxFd *fd*, int64 *length*)

Note:

ftruncate, syscall 77 (64-bit)

Parameters:

← ***fd*** file descriptor
← ***length*** length

Returns:

zero on success, errno on failure

2.4.1.34 int ftruncate32 (LinuxFd *fd*, int32 *length*)

Note:

ftruncate, syscall 93 (32-bit)

Parameters:

← ***fd*** file descriptor
← ***length*** length

Returns:

zero on success, errno on failure

2.4.1.35 int ftruncate64 (LinuxFd *fd*, uint32 *llow*, int32 *lhigh*)

Note:

ftruncate, syscall 194 (32-bit)

Parameters:

← ***fd*** file descriptor
← ***llow*** low 32-bits of the new length
← ***lhigh*** upper 32-bits of the new length

Returns:

zero on success, errno on failure

2.4.1.36 int futimesat (int *dirFd*, const char ∗ *userPath*, const struct timeval ∗ *userTimes*)

Note:

futimesat, syscall 299 (32-bit) and 261 (64-bit)

AT_FDCWD is the only supported value for dirfd today

Parameters:

← ***dirFd*** relative paths interpreted from dirFd
← ***userPath*** pathname
← ***userTimes*** atime and mtime timeval

Returns:

zero on success, errno on failure

2.4.1.37 int64 getcwd (UserVA *buf*, uint64 *bufsize*)

Note:

getcwd, syscall 183 (32-bit) and 79 (64-bit)

Parameters:

← ***buf*** dst buffer
← ***bufsize*** buffer size

Returns:

number of bytes written in buf on success, errno on failure

2.4.1.38 int64 getdents (LinuxFd *fd*, UserVA *userBuf*, uint32 *nbyte*)

Note:

getdents, syscall 141 (32-bit) and 78 (64-bit)
Limited support: d_ino set to -1 if i-node overflows the field

Parameters:

← ***fd*** file descriptor

→ ***userBuf*** destination buffer

← ***nbyte*** size of the buffer

Returns:

number of bytes read on success, errno on failure

2.4.1.39 int64 getdents64 (LinuxFd *fd*, UserVA *userBuf*, uint32 *nbyte*)

Note:

getdents, syscall 220 (32-bit) and 214 (64-bit).

Parameters:

← ***fd*** file descriptor

→ ***userBuf*** destination buffer

← ***nbyte*** size of the buffer

Returns:

number of bytes read on success, errno on failure

2.4.1.40 int64 getxattr (UserVAConst *userPath*, UserVAConst *userName*, UserVA *userValue*, int64 *size*)

Note:

getxattr, syscall 191 (32-bit) and 229 (64-bit).
Limited support : Only supported for vmkaccess extended attribute

Parameters:

← ***userPath*** pathanme

← ***userName*** name

← ***userValue*** value

← ***size*** size

Returns:

0 on success, errno on failure

2.4.1.41 int64 ioctl (LinuxFd *fd*, uint32 *cmd*, UserVA *userData*)

Note:

ioctl, syscall 54 (32-bit) and 16 (64-bit).
Limited support : some file types don't support this call

Parameters:

← ***fd*** file descriptor

← ***cmd*** request

← ***userData*** data

Returns:

cmd-specific return value, -1 on failure

2.4.1.42 int64 lchown (UserVAConst *userPath*, LinuxUID *uid*, LinuxGID *gid*)

Note:

lchown, syscall 198 (32-bit) and 94 (64-bit)

Parameters:

← ***userPath*** pathname

← ***uid*** user id

← ***gid*** group id

Returns:

zero on success, errno on failure

2.4.1.43 int64 link (UserVAConst *oldPath*, UserVAConst *newPath*)

Note:

link, syscall 9 (32-bit) and 86 (64-bit)
Making a hard link to a symlink follows the symlink first.

Parameters:

← ***oldPath*** pathname

← ***newPath*** link name

Returns:

0 on success, errno on failure

2.4.1.44 int linkat (int *dirfd*, const char ∗ *oldPath*, const char ∗ *newPath*)

Note:

linkat, syscall 303 (32-bit) and 265 (64-bit)
linkat, AT_FDCWD is the only supported value for dirfd.

Parameters:

← ***dirfd*** relative paths interpreted from dirfd

← ***oldPath*** pathname

← ***newPath*** link name

Returns:

0 on success, errno on failure

2.4.1.45 int64 LinuxFileDesc_SymlinkAt (UserVAConst *userTo*, LinuxFd *userPathDirFD*, UserVAConst *userPath*)

Note:

symlinkat, AT_FDCWD is the only supported value for pathDirFD today.

Parameters:

← ***userTo*** target pathname

← ***userPathDirFD*** relative paths interpreted from userPathDirFD

← ***userPath*** link name

Returns:

zero on success, errno on failure

2.4.1.46 int llseek (LinuxFd *fd*, int32 *ohigh*, uint32 *olow*, UserVA *userRes*, uint32 *whence*)

Note:

llseek, syscall 140 (32-bit)

Parameters:

← ***fd*** file descriptor

← ***ohigh*** offset's upper 32 bit

← ***olow*** offset's lower 32 bit

→ ***userRes*** new offset

← ***whence*** whence

Returns:

return zero on success, errno on failure

2.4.1.47 int64 lseek (LinuxFd *fd*, int64 *offset*, int32 *whence*)

Note:

lseek, syscall 19 (32-bit) and 8 (64-bit)

Parameters:

← ***fd*** file descriptor

← ***offset*** offset

← ***whence*** whence

Returns:

returns the new offset on success, errno on failure

2.4.1.48 int64 lstat (UserVAConst *userPath*, UserVA *statbuf*)

Note:

lstat, syscall 107 (32-bit) and 6 (64-bit).
NOT SUPPORTED in 32-bit (ENOSYS)

Parameters:

← ***userPath*** pathname

← ***statbuf*** dst buffer

Returns:

zero on success, errno on failure

2.4.1.49 int64 lstat64 (UserVAConst *userPath*, UserVA *statbuf*)

Note:

lstat64, syscall 196 (32-bit)
limited support not all some fields are not supported in some file types.

Parameters:

← ***userPath*** pathname

← ***statbuf*** dst buffer

Returns:

zero on success, errno on failure

2.4.1.50 int64 mkdir (UserVAConst *userPath*, IdentityMode *mode*)

Note:

mkdir, syscall 39 (32-bit) and 83 (64-bit)

Parameters:

← ***userPath*** pathname

← ***mode*** permission mode

Returns:

zero on success, errno on failure

2.4.1.51 int mkdirat (int *dirfd*, const char ∗ *userPath*, mode_t *mode*)

Note:

mkdirat, syscall 296 (32-bit) and 258 (64-bit)
mkdirat, AT_FDCWD is the only supported value for dirfd.

Parameters:

← ***dirfd*** relative paths interpreted from dirfd

← ***userPath*** pathname

← ***mode*** permission mode

Returns:

zero on success, errno on failure

2.4.1.52 int64 mknod (UserVAConst *userPath*, IdentityMode *mode*, uint64 *devId*)

Note:

mknod, syscall 14 (32-bit) and 133 (64-bit)
supported flags : S_IFIFO S_IFCHR

Parameters:

← ***userPath*** pathname

← ***mode*** permission flag

← ***devId*** dev ID

Returns:

0 on success, errno on failure

2.4.1.53 int mknodat (int *dirFd*, const char ∗ *userPath*, mode_t *mode*, dev_t *devId*)

Note:

mknodat, syscall 297 (32-bit) and 259 (64-bit)
mknodat, AT_FDCWD is the only supported value for dirfd.

Parameters:

← ***dirFd*** from relative paths interpreted from dirFD

← ***userPath*** pathname

← ***mode*** permission flag

← ***devId*** dev ID

Returns:

zero on success, errno on failure

2.4.1.54 int64 mount (UserVAConst *userSpecialfile*, UserVAConst *userDir*, UserVAConst *filesystemType*, unsigned long *mountflags*, UserVAConst *data*)

Note:

mount, syscall 21 (32-bit) and syscall 165 (64-bit).
Not fully supported

Parameters:

← ***userSpecialfile*** source

← ***userDir*** target dir

← ***filesystemType*** fs type

← ***mountflags*** mount flags

← ***data*** data

Returns:

0 on success, errno on failure

2.4.1.55 int64 open (UserVAConst *userPath*, uint32 *flags*, IdentityMode *mode*)

Note:

open, syscall 5 (32-bit) and 2 (64-bit)
supported flags : O_RDONLY O_RDWR O_WONLY O_CREATE O_EXCLUSIVE O_NOTTY O_-TRUNCATE O_APPEND O_NONBLOCK O_SYNC O_DSYNC O_LARGEFILE O_DIRECT O_-DIRECTORY O_NOFOLLOW O_MULTIWRITER_LOCK O_PENULTIMATE O_IGNTRAILING O_STAT O_EXCLUSIVE_LOCK O_SWMR_LOCK

Parameters:

← ***userPath*** pathname

← ***flags*** open flags

← ***mode*** mode

Returns:

return file descriptor, -1 on error

2.4.1.56 int openat (int *dirfd*, const char ∗ *userPath*, int *flags*, mode_t *mode*)

Note:

openat, syscall 295 (32-bit) and 257 (64-bit)
openat, AT_FDCWD is the only supported value for dirfd.
supported flags : O_RDONLY O_RDWR O_WONLY O_CREATE O_EXCLUSIVE O_NOTTY O_-TRUNCATE O_APPEND O_NONBLOCK O_SYNC O_DSYNC O_LARGEFILE O_DIRECT O_-DIRECTORY O_NOFOLLOW O_MULTIWRITER_LOCK O_PENULTIMATE O_IGNTRAILING O_STAT O_EXCLUSIVE_LOCK

Parameters:

← ***dirfd*** relative paths interpreted from dirfd

← ***userPath*** pathname

← ***flags*** open flags

← ***mode*** mode

Returns:

return file descriptor, -1 on error

2.4.1.57 int64 pipe (UserVA *pipefds*, uint32 *flags*)

Note:

pipe, syscall 331 (32-bit) and 293 (64-bit)

Parameters:

→ ***pipefds*** array used to return the 2 fds

← ***flags*** currently supports O_NONBLOCK and O_CLOEXEC

Returns:

zero on success, errno on failure

2.4.1.58 int64 pipe (UserVA *pipefds*)

Note:

pipe, syscall 42 (32-bit) and 22 (64-bit)

Parameters:

→ ***pipefds*** array used to return the 2 fds

Returns:

zero on success, errno on failure

2.4.1.59 int64 poll (UserVA *userPollFds*, uint64 *nfds*, int32 *timeoutMillis*)

Note:

poll, syscall 168 (32-bit) and 7 (64-bit).
1. Limited support not all fd types support poll. 2. nfds == 0 case is not supported

Parameters:

← ***userPollFds*** list of fds

← ***nfds*** number of fds

← ***timeoutMillis*** timeout

Returns:

zero on success, errno on failure

2.4.1.60 int64 ppoll (UserVA *userPollFds*, uint64 *nfds*, struct timespec ∗ *timeout*, const sigset_t ∗ *sigmask*, size_t *sigsetsize*)

Note:

ppoll, syscall 309 (32-bit) and 271 (64-bit).
1. Limited support not all fd types support poll. 2. nfds == 0 case is not supported

Parameters:

← ***userPollFds*** list of fds

← ***nfds*** number of fds

← ***timeout*** struct timespec timeout

← ***sigmask*** signal set

← ***sigsetsize*** signal set size

Returns:

zero on success, errno on failure

2.4.1.61 int64 pread64 (LinuxFd *fd*, UserVA *userBuf*, uint64 *nbyte*, int64 *offset*)

Note:

pread, syscall 180 (32-bit) and 17 (64-bit).

Parameters:

← ***fd*** file descriptor

← ***userBuf*** dst buffer

← ***nbyte*** number of bytes to read

← ***offset*** offset in the file

Returns:

number of bytes read on success, errno on failure

2.4.1.62 int64 preadV (LinuxFd *fd*, UserVA *userIovp*, uint32 *iovcnt*, int64 *offset*)

Note:

preadv, syscall 333(32-bit) and 295(64-bit)
1. Supports vectors no longer than 10. (iovcnt <= 10)

Parameters:

← ***fd*** file descriptor

← ***userIovp*** io-vector

← ***iovcnt*** size of vector

← ***offset*** offset in the file

Returns:

number of bytes read on success, errno on failure

2.4.1.63 int pselect (int *n*, fd_set ∗ *readfds*, fd_set ∗ *writefds*, fd_set ∗ *exceptfds*, struct timeval ∗ *timeout*, const sigset_t ∗ *sigmask*)

Note:

pselect, syscall 308 (32-bit) and 270 (64-bit).
Not fully supported
1. Only readfds and writefds are supported. (exceptfds is ignored)
2. zero-fds case is not supported

Parameters:

← ***n*** The highest-numbered file descriptor plus 1

← ***readfds*** fd for reading

← ***writefds*** fd for writting

← ***exceptfds*** Ignored

↔ ***timeout*** select timeout

← ***sigmask*** signal set

Returns:

number of ready fds on success, errno on failure

2.4.1.64 int64 pwrite64 (LinuxFd *fd*, UserVAConst *userBuf*, uint64 *nbyte*, int64 *offset*)

Note:

pwrite, syscall 181 (32-bit) and 18 (64-bit).

Parameters:

← ***fd*** file descriptor

← ***userBuf*** src buffer

← ***nbyte*** number of bytes to write

← ***offset*** offset in the file

Returns:

number of bytes written on success, errno on failure

2.4.1.65 int64 pwritev (LinuxFd *fd*, UserVA *userIovp*, uint32 *iovcnt*, int64 *offset*)

Note:

pwritev, syscall 334 (32-bit) and 296 (64-bit).
1. Supports vectors no longer than 10. (iovcnt <= 10)

Parameters:

← ***fd*** file descriptor
← ***userIovp*** io-vector
← ***iovcnt*** size of vector
← ***offset*** offset in the file

Returns:

number of bytes written on success, errno on failure

2.4.1.66 int64 read (LinuxFd *fd*, UserVA *userBuf*, uint32 *nbyte*)

Note:

read, syscall 3 (32-bit) and 19 (64-bit)

Parameters:

← ***fd*** file descriptor
→ ***userBuf*** destination
← ***nbyte*** number of bytes to read

Returns:

number of bytes read is returned on sucess. errno on failure

2.4.1.67 int64 readlink (UserVAConst *userPath*, UserVA *userBuf*, int64 *inCount*)

Note:

readlink, syscall 85 (32-bit) and 89 (64-bit).

Parameters:

← ***userPath*** link path
→ ***userBuf*** buffer to copy link content
← ***inCount*** buffer size

Returns:

On success returns the number of bytes placed in userBuf, errno on failure

---

2.4.1.68 int readlinkat (int *dirfd*, const char ∗ *userPath*, char ∗ *userBuf*, size_t *bufSize*)

Note:

readlinkat, syscall 305 (32-bit) and 267 (64-bit)
readlinkat, AT_FDCWD is the only supported value for dirfd.

Parameters:

← ***dirfd*** relative paths interpreted from dirfd

← ***userPath*** link path

→ ***userBuf*** buffer to copy link content

← ***bufSize*** buffer size

Returns:

On success returns the number of bytes placed in userBuf, errno on failure

---

2.4.1.69 int64 readv (LinuxFd *fd*, UserVA *userIovp*, uint32 *iovcnt*)

Note:

readv, syscall 145 (32-bit) and 19 (64-bit).
1. Supports vectors no longer than 10. (iovcnt <= 10)

Parameters:

← ***fd*** file descriptor

← ***userIovp*** io-vector

← ***iovcnt*** size of the vector

Returns:

number of bytes read on success, errno on failure

2.4.1.70 int64 rename (UserVAConst *oldPath*, UserVAConst *newPath*)

Note:

rename, syscall 38 (32-bit) and 82 (64-bit)

Parameters:

← ***oldPath*** old pathname

← ***newPath*** new pathname

Returns:

zero on success, errno on failure

2.4.1.71 int renameat (int *fromDirFd*, const char ∗ *from*, int *toDirFd*, const char ∗ *to*)

Note:

renameat, syscall 302 (32-bit) and 264 (64-bit)
renameat, AT_FDCWD is the only supported value for dirfd.

Parameters:

← ***fromDirFd*** from relative paths interpreted from fromDirFD

← ***from*** old pathname

← ***toDirFd*** to relative paths interpreted from toDirFD

← ***to*** new pathname

Returns:

zero on success, errno on failure

2.4.1.72 int64 rmdir (UserVAConst *userPath*)

Note:

rmdir, syscall 40 (32-bit) and 84 (64-bit)

Parameters:

← ***userPath*** pathname

Returns:

zero on success, errno on failure

2.4.1.73 int select (int *n*, fd_set ∗ *readfds*, fd_set ∗ *writefds*, fd_set ∗ *exceptfds*, struct timeval ∗ *timeout*)

Note:

select, syscall 142 (32-bit) and 23 (64-bit).
Not fully supported
1. Only readfds and writefds are supported. (exceptfds is ignored)
2. zero-fds case is not supported

Parameters:

← ***n*** The highest-numbered file descriptor plus 1

← ***readfds*** fd for reading

← ***writefds*** fd for writting

← ***exceptfds*** Ignored

↔ ***timeout*** Select timeout

Returns:

number of ready fds on success, errno on failure

2.4.1.74 int64 sendfile64 (LinuxFd *out_fd*, LinuxFd *in_fd*, UserVA *offset*, uint32 *count*)

Note:

sendfile64, syscall 239 (32-bit) and 40 (64-bit)
in_fd must be a file and out_fd must be an AF_INET socket.

Parameters:

← ***out_fd*** Fd to write to

← ***in_fd*** Fd to read from

← ***offset*** VA of offset in in_fd, or NULL

← ***count*** Maximum amount of data to transfer

Returns:

return the number of bytes transferred on success, error code otherwise

---

2.4.1.75 int64 setxattr (UserVAConst *userPath*, UserVAConst *userName*, UserVA *userValue*, int64 *size*, int64 *flags*)

Note:

setxattr, syscall 226 (32-bit) and 118 (64-bit).
Limited support : Only supported for vmkaccess extended attribute

Parameters:

← ***userPath*** pathanme

← ***userName*** name

← ***userValue*** value

← ***size*** size

← ***flags*** flags

Returns:

0 on success, errno on failure

2.4.1.76 int64 stat (UserVAConst *userPath*, UserVA *statbuf*)

Note:

stat, syscall 106 (32-bit) and 4 (64-bit).

Parameters:

← ***userPath*** pathname
← ***statbuf*** dst buffer

Returns:

zero on success, errno on failure

Note:

stat, syscall 106 (32-bit).

Parameters:

← ***userPath*** pathname
← ***statbuf*** dst buffer

Returns:

zero on success, errno on failure

2.4.1.77 int stat64 (UserVAConst *userPath*, UserVA *statbuf*)

Note:

stat64, syscall 195 (32-bit)
limited support not all some fields are not supported in some file types.

Parameters:

← ***userPath*** pathname
← ***statbuf*** dst buffer

Returns:

zero on success, errno on failure

2.4.1.78 int64 statfs (UserVAConst *userPath*, UserVA *userBuf*)

Note:

statfs, syscall 99 (32-bit) and 137 (64-bit)
Copies statfs information to user's buffer.

Parameters:

← ***userPath*** pathname
→ ***userBuf*** destination buffer

Returns:

zero on success, errno on failure

2.4.1.79 int statfs64 (UserVAConst *userPath*, uint32 *userBufSize*, UserVA *userBuf*)

Note:

statfs64, syscall 268 (32-bit). Not supported in 64-bit

Parameters:

← ***userPath*** pathname

← ***userBufSize*** buffer size

→ ***userBuf*** destination buffer

Returns:

0 on success, errno on failure

2.4.1.80 int64 symlink (UserVAConst *userTo*, UserVAConst *userPath*)

Note:

dup2, syscall 83 (32-bit) and 88 (64-bit).

Parameters:

← ***userTo*** target pathname

← ***userPath*** link name

Returns:

zero on success, errno on failure

2.4.1.81 int64 truncate (UserVAConst *userPath*, int64 *length*)

Note:

truncate, syscall 76 (64-bit)

Parameters:

← ***userPath*** pathname

← ***length*** length

Returns:

zero on success, errno on failure

2.4.1.82 int truncate32 (UserVAConst *userPath*, int32 *length*)

Note:

truncate, syscall 92 (32-bit)

Parameters:

← ***userPath*** pathname

← ***length*** length

Returns:

zero on success, errno on failure

2.4.1.83 int truncate64 (UserVAConst *userPath*, uint32 *llow*, int32 *lhigh*)

Note:

truncate64, syscall 193 (32-bit)

Parameters:

← ***userPath*** pathname

← ***llow*** low 32 bits of the new length

← ***lhigh*** upper 32 bits of the new length

Returns:

zero on success, errno on failure

2.4.1.84 int64 umask (uint32 *newmask*)

Note:

sets the calling process's file mode creation mask umask, syscall 60 (32-bit) and 95 (64-bit).

Parameters:

← ***newmask*** new mask

Returns:

This system call always succeeds and it returns the old mask value

---

2.4.1.85 int64 umount (UserVAConst *userDir*, uint32 *delayflag*)

Note:

umount, syscall 52 (32-bit) and syscall 166 (64-bit)
Not fully supported

Parameters:

← ***userDir*** target directory

← ***delayflag*** flag

Returns:

0 on success, errno on failure

2.4.1.86 int64 unlink (UserVAConst *userPath*)

Note:

unlink, syscall 10 (32-bit) and 87 (64-bit)

Parameters:

← ***userPath*** pathname

Returns:

0 on success, errno on failure

2.4.1.87 int unlinkat (int *dirfd*, const char ∗ *userPath*, int *flags*)

Note:

unlinkat, syscall 301 (32-bit) and 263 (64-bit)
unlinkat, AT_FDCWD is the only supported value for dirfd.

Parameters:

← ***dirfd*** relative paths interpreted from dirfd

← ***userPath*** pathname

← ***flags*** optional AT_REMOVEDIR

Returns:

0 on success, errno on failure

2.4.1.88 int utime (const char ∗ *userPath*, const struct utimebuf ∗ *utimebuf*)

Note:

utime, syscall 30 (32-bit) and 132 (64-bit)
Limited support: vmfs and visorfs don't support 64 bit atime and mtime If the passed-in argument is overflowing the 32-bit this call returns an error.

Parameters:

← ***userPath*** pathname
← ***utimebuf*** utimebuf

Returns:

zero on success, errno on failure

2.4.1.89 int utimensat (int *dirFd*, const char ∗ *userPath*, const struct timeval ∗ *userTimes*, int *flags*)

Note:

utimensat, syscall 320 (32-bit) and 280 (64-bit) on x86, syscall 88 on arm64. AT_FDCWD is the only supported value for dirfd today, only AT_SYMLINK_NOFOLLOW supported for flags.
XXX vmfs, visorfs only support atime and mtime seconds in the file attributes. Nanoseconds will be dropped.

Parameters:

← ***dirFd*** relative paths interpreted from dirFd
← ***userPath*** pathname
← ***userTimes*** atime and mtime timespec
← ***flags*** optional AT_SYMLINK_NOFOLLOW

Returns:

zero on success, errno on failure

2.4.1.90 int utimes (const char ∗ *userPath*, const struct timeval ∗ *userTimes*)

Note:

utimes, syscall 271 (32-bit) and 235 (64-bit)

Parameters:

← ***userPath*** pathname
← ***userTimes*** atime and mtime timeval

Returns:

zero on success, errno on failure

2.4.1.91 int64 write (LinuxFd *fd*, UserVA64Const *userBuf*, uint64 *nbyte*)

Note:

read, syscall 4 (32-bit) and 1 (64-bit)

Parameters:

← ***fd*** file descriptor

← ***userBuf*** src

← ***nbyte*** number of bytes to write

Returns:

number of bytes written is returned on sucess. errno on failure

---

2.4.1.92 int64 writev (LinuxFd *fd*, UserVA *userIovp*, uint32 *iovcnt*)

Note:

readv, syscall 146 (32-bit) and 20 (64-bit).
1. Supports vectors no longer than 10. (iovcnt <= 10)

Parameters:

← ***fd*** file descriptor

← ***userIovp*** io-vector

← ***iovcnt*** size of the vector

Returns:

number of bytes written on success, errno on failure

2.5 Linux user/group identity

System calls related to user identity.

Functions

- LinuxGID getegid (void)

  get the effective group ID

- LinuxUID geteuid (void)

  get the effective user ID

- LinuxGID getgid (void)

  get the real group ID

- int64 getgroups (int32 ngids, UserVAuserGIDs)

  get supplementary group IDs

- int64 getresgid (UserVAuserRgid, UserVAuserEgid, UserVAuserSgid)

  get real, effective and saved group ID

- int64 getresuid (UserVAuserRuid, UserVAuserEuid, UserVAuserSuid)

  get real, effective and saved user ID

- LinuxUID getuid (void)

  get a real user ID

- int64 setgid (LinuxGID gid)

  set group ID

- int64 setgroups (uint32 ngids, UserVAConstuserGIDs)

  set list of supplementary group IDs

- int64 setregid (LinuxGID rgid, LinuxGID egid)

  set real and effective group IDs

- int64 setresgid (LinuxGID rgid, LinuxGID egid, LinuxGID sgid)

  set real, effective and saved group ID

- int64 setresuid (LinuxUID ruid, LinuxUID euid, LinuxUID suid)

  set real, effective and saved user ID

- int64 setreuid (LinuxUID ruid, LinuxUID euid)

  set real and effective user IDs

- int64 setuid (LinuxUID uid)

  set user ID

2.5.1 Function Documentation

2.5.1.1 LinuxGID getegid (void)

Note:

getegid, syscall 202(32-bit) and 108(64-bit)
This syscall is fully supported.

Returns:

Returns the effective primary gid of the thread.

2.5.1.2 LinuxUID geteuid (void)

Note:

geteuid, syscall 201(32-bit) and 107(64-bit)
This syscall is fully supported.

Returns:

Returns the effective uid of the thread.

2.5.1.3 LinuxGID getgid (void)

Note:

getgid, syscall 200(32-bit) and 104(64-bit)
This syscall is fully supported.

Returns:

Returns the real primary gid of the thread.

2.5.1.4 int64 getgroups (int32 *ngids*, UserVA *userGIDs*)

Note:

getgroups, syscall 205(32-bit) and 115(64-bit)
This syscall is fully supported.

Parameters:

$\leftarrow$ ***ngids*** Number of elements in the array ’userGIDs’
$\leftrightarrow$ ***userGIDs*** Grouplist array

Returns:

Returns the the number of supplementary group IDs on success, -1 on failure

---

2.5.1.5 int64 getresgid (UserVA *userRgid*, UserVA *userEgid*, UserVA *userSgid*)

Note:

getresgid, syscall 211(32-bit) and 120(64-bit)
This syscall is fully supported.

Parameters:

→ ***userRgid*** pointer to real group ID

→ ***userEgid*** pointer to effective group ID

→ ***userSgid*** pointer to saved (effective) group ID

Returns:

Returns 0 on success, -1 on error.

2.5.1.6 int64 getresuid (UserVA *userRuid*, UserVA *userEuid*, UserVA *userSuid*)

Note:

getresuid, syscall 209(32-bit) and 118(64-bit)
This syscall is fully supported.

Parameters:

→ ***userRuid*** pointer to real user ID

→ ***userEuid*** pointer to effective user ID

→ ***userSuid*** pointer to saved (effective) user ID

Returns:

Returns 0 on success, -1 on error.

2.5.1.7 LinuxUID getuid (void)

Note:

getuid, syscall 199(32-bit) and 102(64bit)
This syscall is fully supported.

Returns:

Returns the real uid of the thread.

2.5.1.8 int64 setgid (LinuxGID *gid*)

Note:

setgid, syscall 214(32-bit) and 106(64-bit)
This syscall is fully supported.

Parameters:

← ***gid*** New group id

Returns:

Returns 0 on success, -1 on failure

2.5.1.9 int64 setgroups (uint32 *ngids*, UserVAConst *userGIDs*)

Note:

setgroups, syscall 206(32-bit) and 116(64-bit)
This syscall is fully supported.

Parameters:

← ***ngids*** Number of elements in the array 'userGIDs'

← ***userGIDs*** Array of group IDs.

Returns:

Returns 0 on success, -1 on failure.

2.5.1.10 int64 setregid (LinuxGID *rgid*, LinuxGID *egid*)

Note:

setregid, syscall 204(32-bit) and 114(64-bit)
This syscall is fully supported.
Privileged processes with gid 0 can set the rgid and egid to any value. Unprivileged processes can set the rgid to either the previous rgid or the previous sgid. Unprivileged processes can set the egid to either the previous rgid, the previous egid or the previous sgid. If rgid is set or if egid is set to a value not equal to the previous rgid, then sgid is set to the value of the new egid.

Parameters:

← ***rgid*** Real group ID

← ***egid*** Effective group ID

Returns:

Returns 0 on success, -1 otherwise.

2.5.1.11 int64 setresgid (LinuxGID *rgid*, LinuxGID *egid*, LinuxGID *sgid*)

Note:

setresgid, syscall 210(32-bit) and 119(64-bit)
This syscall is fully supported.

Parameters:

← ***rgid*** real group ID

← ***egid*** effective group ID

← ***sgid*** saved (effective) group ID

Returns:

Returns 0 on success, -1 on error.

2.5.1.12 int64 setresuid (LinuxUID *ruid*, LinuxUID *euid*, LinuxUID *suid*)

Note:

setresuid, syscall 208(32-bit) and 117(64-bit)
This syscall is fully supported.

Parameters:

← ***ruid*** real user ID

← ***euid*** effective user ID

← ***suid*** saved (effective) user ID

Returns:

Returns 0 on success, -1 on error.

2.5.1.13 int64 setreuid (LinuxUID *ruid*, LinuxUID *euid*)

Note:

setreuid, syscall 203(32-bit) and 113(64-bit)
This syscall is fully supported.
Privileged processes with uid 0 can set the ruid and euid to any value. Unprivileged processes can set the ruid to either the previous ruid or the previous euid. Unprivileged processes can set the euid to either the previous ruid, the previous euid or the previous suid. If ruid is set or if euid is set to a value not equal to the previous ruid, then suid is set to the value of the new euid.

Parameters:

← ***ruid*** Real user ID

← ***euid*** Effective user ID

Returns:

Returns 0 on success, -1 otherwise.

2.5.1.14 int64 setuid (LinuxUID *uid*)

Note:

setuid, syscall 213(32-bit) and 105(64-bit)
This syscall is fully supported.

Parameters:

← ***uid***	New user id

Returns:

Returns 0 on success, -1 on failure

2.6 system calls

Supports addition of watches on files to monitor writes, Supports removal of watches.

2.7 SystemV IPC system calls.

System calls for interprocess communication.

Functions

- int32 ipc (uint32 whichCall, int32 arg1, int32 arg2, int32 arg3, void ∗ptr, int32 arg5)

  common entry for SystemV IPC operations

- int64 semctl (int32 semId, int32 semNum, int32 cmd, union semun semun)

  Perform semaphore control operations.

- int64 semget (int32 key, int32 nSems, int32 flags)

  Create or get semaphore set identifier for specified key.

- int64 semop (int32 semId, struct sembuf ∗semBuf, uint32 nOps)

  Perform semaphore P() and V() operations.

- int64 semtimedop (int32 semId, struct sembuf ∗semBuf, uint32 nOps, struct timespec ∗timeout)

  Perform semaphore P() and V() operations.

- UserVA shmat (int32 shmId, void ∗shmAddr, uint32 flags)

  Attach to shm segment, 64-bit version. This doesn't use the ipc() wrapper, so just returns the address directly in registers.

- int32 shmat (int32 shmId, void ∗shmAddr, uint32 flags, void ∗retAddr)

  Attach to shm segment, 32-bit version; called from ipc() wrapper. Returned address is copied back into userspace at retAddr.

- int64 shmctl (int32 shmId, uint32 cmd, void ∗ptr)

  Perform shared memory control operations.

- int64 shmdt (void ∗shmAddr)

  Attach to shm segment.

- int64 shmget (uint32 key, uint64 size, uint32 flags)

  Return identifier for SHM segment.

2.7.1 Function Documentation

2.7.1.1 int32 ipc (uint32 *whichCall*, int32 *arg1*, int32 *arg2*, int32 *arg3*, void ∗ *ptr*, int32 *arg5*)

Note:

ipc, syscall 117 (32 bit only)
Support: 33%, only semaphores supported
Error case: 33% , when inapplicable returns ENOSYS

Parameters:

← ***whichCall*** Which funciton to invoke
↔ ***arg1*** Function parameter 1
↔ ***arg2*** Function parameter 2
↔ ***arg3*** Function parameter 3
↔ ***ptr*** Function parameter 4
↔ ***arg5*** Function parameter 5

Returns:

Varies depending upon the whichCall parameter.

Postcondition:

May read or write data at address userData, depending on call

---

2.7.1.2 int64 semctl (int32 *semId*, int32 *semNum*, int32 *cmd*, union semun *semun*)

Note:

32 bit: ipc, syscall 117 with whichCall == SEMCTL
64 bit: syscall 66
Only 64 bit aligned ipc_perm supported.

Parameters:

← ***semId*** Semaphore set id
← ***semNum*** Semaphore number in
← ***cmd*** Semaphore control operation
↔ ***semun*** Semaphore union

Returns:

Non-negative on success, -1 otherwise

2.7.1.3 int64 semget (int32 *key*, int32 *nSems*, int32 *flags*)

Note:

32 bit: ipc, syscall 117 with whichCall == SEMGET
64 bit: syscall 64

Parameters:

← ***key*** Semaphore key
← ***nSems*** Number of semaphores in the set
← ***flags*** Supported flags: IPC_CREAT, IPC_EXCL

Returns:

Semaphore set identifier on success, -1 otherwise

2.7.1.4 int64 semop (int32 *semId*, struct sembuf ∗ *semBuf*, uint32 *nOps*)

Note:

32 bit: unsupported
64 bit: syscall 65

Parameters:

← ***semId*** Semaphore set id

← ***semBuf*** Semaphore operations buffer

← ***nOps*** Number of operations in buffer

Returns:

0 on success, -1 otherwise

2.7.1.5 int64 semtimedop (int32 *semId*, struct sembuf ∗ *semBuf*, uint32 *nOps*, struct timespec ∗ *timeout*)

Note:

32 bit: ipc, syscall 117 with whichCall == SEMTIMEDOP or SEMOP
64 bit: syscall 220

Parameters:

← ***semId*** Semaphore set id

← ***semBuf*** Semaphore operations buffer

← ***nOps*** Number of operations in buffer

← ***timeout*** Operation timeout, SEMTIMEDOP only

Returns:

0 on success, -1 otherwise

2.7.1.6 UserVA shmat (int32 *shmId*, void ∗ *shmAddr*, uint32 *flags*)

Parameters:

← ***shmId*** Shared memory ID

← ***shmAddr*** requested address

← ***flags*** flags

Returns:

Address of shm area, or (void∗)-1 on failure

2.7.1.7 int32 shmat (int32 *shmId*, void ∗ *shmAddr*, uint32 *flags*, void ∗ *retAddr*)

Parameters:

← ***shmId*** Shared memory ID

← ***shmAddr*** requested address

← ***flags*** flags

→ ***retAddr*** return address

Returns:

0 on success, -1 on failure (retAddr also -1)

2.7.1.8 int64 shmctl (int32 *shmId*, uint32 *cmd*, void ∗ *ptr*)

Note:

32 bit: ipc, syscall 117 with whichCall == SHMCTL
64 bit: syscall 31
Only 64 bit aligned ipc_perm supported.

Parameters:

← ***shmId*** Shared memory ID

← ***cmd*** Command

↔ ***ptr*** varies, see man page

Returns:

Non-negative on success, -1 otherwise

2.7.1.9 int64 shmdt (void ∗ *shmAddr*)

Parameters:

← ***shmAddr*** address to detach from

Returns:

0 on success, -1 on failure

2.7.1.10 int64 shmget (uint32 *key*, uint64 *size*, uint32 *flags*)

Parameters:

← ***key*** SHM segment key

← ***size*** size of shm segment

← ***flags*** permission flags

Returns:

Valid segment identifier, or -1 on failure

2.8 Memory management system calls

System calls for managing memory.

Functions

- int64 brk (UserVA dataEnd)

  Change data segment size.

- int64 mmap (UserVA start, uint64 len, int prot, int flags, int fd, uint64 offset)

  mmap - map regular files and anonymous memory.

- int64 mmap2 (UserVA addr, uint64 len, uint32 prot, uint32 flags, LinuxFd fd, uint64 pgoff)

  mmap2 map regular files and anonymous memory.

- int64 mprotect (UserVA addr, uint64 len, int prot)

  cotrol allowable accesses to a region of memory

- UserVA mremap (UserVA addr, uint64 oldLen, uint64 newLen, uint32 flags)

  re-map a virtual memory address

- int64 msync (UserVA addr, uint64 len, uint32 flags)

  synchronize a file with a memory map

- int64 munmap (UserVA addr, uint64 len)

  unmap files or devices from memory

2.8.1 Function Documentation

2.8.1.1 int64 brk (UserVA *dataEnd*)

Note:

brk, syscall 45
This syscall is fully supported.

Parameters:

← ***dataEnd*** Value of the end of data segment

Returns:

0 on success, -1 on error

Postcondition:

Illegal adjustments simply leave the brk unchanged. Changes range of pages that are valid in current cartel's heap.

2.8.1.2 int64 mmap (UserVA *start*, uint64 *len*, int *prot*, int *flags*, int *fd*, uint64 *offset*)

Note:

mmap, syscall 9 (64-bit)
This syscall is not fully supported. See limitations below
1. Only the following flags are supported.
LINUX_MMAP_PRIVATE
LINUX_MMAP_FIXED
LINUX_MMAP_ANONYMOUS
LINUX_MMAP_LOCKED
LINUX_MMAP_POPULATE.
2. mmap'ing anything other than regular files and anonymous memory is not supported.

Parameters:

← ***start*** Preferred start address of the mmap'ed region

← ***len*** Length in bytes of the mmaped region

← ***prot*** Desired memory protection

← ***flags*** Mapping options

← ***fd*** Valid file descriptor, unless MAP_ANONYMOUS is set

← ***offset*** Offset into the file

Returns:

pointer to the mapped area on success. MAP_FAILED otherwise.

2.8.1.3 int64 mmap2 (UserVA *addr*, uint64 *len*, uint32 *prot*, uint32 *flags*, LinuxFd *fd*, uint64 *pgoff*)

NOTE: We are talking about mmap2 system call. If you use 'mmap' in your 32bit application code glibc will internally use the mmap2 system call.

Note:

mmap2, syscall 192 (32-bit)
This syscall is not fully supported. See limitations below.
1. Only the following flags are supported.
LINUX_MMAP_PRIVATE
LINUX_MMAP_FIXED
LINUX_MMAP_ANONYMOUS
LINUX_MMAP_LOCKED.
2. mmap'ing anything other than regular files and anonymous memory is not supported.

Parameters:

← ***addr*** Preferred start address of the mmap'ed region

← ***len*** Length in bytes of the mmaped region

← ***prot*** Desired memory protection

← ***flags*** Mapping options

← ***fd*** Valid file descriptor, unless MAP_ANONYMOUS is set.

← ***pgoff*** Offset into the file in units of system page size.

Returns:

pointer to the mapped area on success. MAP_FAILED otherwise.

2.8.1.4 int64 mprotect (UserVA *addr*, uint64 *len*, int *prot*)

Note:

mprotect, syscall 125(32-bit) and 10(64-bit)
This syscall is not fully supported. See limitations below.
Does not support the PROT_NONE flag

Parameters:

← ***addr*** Start address of the range that needs to be protected

← ***len*** Length in bytes

← ***prot*** Flags

Returns:

0 on success, -1 otherwise

2.8.1.5 UserVA mremap (UserVA *addr*, uint64 *oldLen*, uint64 *newLen*, uint32 *flags*)

Note:

mremap, syscall 163(32-bit) and 25(64-bit)
This syscall is not fully supported. See limitations below.
Only the MREMAP_MAYMOVE flag is supported.
Does not support remmapping of multiple overlapping mmap regions.

Parameters:

← ***addr*** Address of the region to remap

← ***oldLen*** Length of the address space that needs to be moved

← ***newLen*** Length of the new region

← ***flags*** flag bitmask

Returns:

Pointer to the new virtual memory area on success, MAP_FAILED on failure

---

2.8.1.6 int64 msync (UserVA *addr*, uint64 *len*, uint32 *flags*)

Note:

msync, syscall 134(32-bit) and 26(64-bit)
This syscall is not fully supported. See limitations below.
Does not support MS_INVALIDATE.

Parameters:

← ***addr*** Start address of the range

← ***len*** Length in bytes

← ***flags*** Flags

Returns:

0 on success, -1 otherwise.

2.8.1.7 int64 munmap (UserVA *addr*, uint64 *len*)

Note:

munmap, syscall 91
This syscall is fully supported.

Parameters:

← ***addr*** Pointer to the region being unmapped

← ***len*** Number of bytes to unmap

Returns:

0 on success, -1 otherwise

2.9 Signal system calls

Signal related system calls.

Functions

- int kill (pid_t pid, int sig)
  Sends any signal to the cartel or group.
- int pause (void)
  Will cause the cartel to slepp until a signal is received.
- int rt_sigaction (int sig, const struct sigaction ∗act, struct sigaction ∗oact, size_t sigsetsize)
  Installs a signal handler for the signum.
- int rt_sigpending (sigset_t ∗set, size_t sigsetsize)
  Get a pending signal raised while blocked.
- int rt_sigprocmask (int32 how, sigset_t ∗set, sigset_t ∗oset, size_t sigsetsize)
  Get and set the blocked signals.
- int rt_sigqueueinfo (LinuxPid pid, int sig, siginfo_t ∗siginfo)
  queue a signal to a process
- int rt_sigreturn (void ∗contextAddr, void ∗baseAddr)
  Returns from the signal handler.
- long rt_sigsuspend (sigset_t ∗mask, size_t sigsetsize)
  Suspends the calling world.
- int rt_sigtimedwait (const sigset_t ∗set, siginfo_t ∗info, const struct struct timespec ∗timeout, size_t sigsetsize)
  Wait for the pending signals specified.
- int sigaltstack (stack_t ∗ss, stack_t ∗oss)
  Get or set alternate signal stack.
- long tgkill (int tgid, int pid, int sig)
  Sends a signal to a specific thread.
- int tkill (int tid, int sig)
  Send a signal to a single thread.

2.9.1 Function Documentation

2.9.1.1 int kill (pid_t *pid*, int *sig*)

Note:

kill, syscall 37 (32-bit) and 62 (64-bit)

This syscall is fully supported.

Parameters:

$\leftarrow$ ***pid***

$\leftarrow$ ***sig***

Returns:

0 on success, appropriate error otherwise

2.9.1.2 int pause (void)

Note:

pause, syscall 29 (32-bit) and 34 (64-bit).
This syscall is fully supported.

Returns:

EINTR

2.9.1.3 int rt_sigaction (int *sig*, const struct sigaction $*$ *act*, struct sigaction $*$ *oact*, size_t *sigsetsize*)

Note:

rt_sigaction, syscall 174 (32-bit) and 13 (64-bit).
This syscall is not fully supported. See limitations below.
SA_NOCLDSTOP, SA_NOCLDWAIT, SA_RESETHAND (SA_ONESHOT)
and SA_RESTART are not supported.

Parameters:

$\leftarrow$ ***sig***

$\leftarrow$ ***act***

$\rightarrow$ ***oact***

$\leftarrow$ ***sigsetsize***

Returns:

0 on success, or appropriate error otherwise

2.9.1.4 int rt_sigpending (sigset_t $*$ *set*, size_t *sigsetsize*)

Note:

rt_sigpending, syscall 176 (32-bit) and 127 (64-bit).
This syscall is fully supported.

Parameters:

$\rightarrow$ ***set***

$\leftarrow$ ***sigsetsize***

Returns:

0 on success, or appropriate error otherwise

2.9.1.5 int rt_sigprocmask (int32 *how*, sigset_t $*$ *set*, sigset_t $*$ *oset*, size_t *sigsetsize*)

Note:

rt_sigprocmask, syscall 175 (32-bit) and 14 (64-bit).
This syscall is fully supported.

Parameters:

$\leftarrow$ ***how***

$\leftarrow$ ***set***

$\rightarrow$ ***oset***

$\leftarrow$ ***sigsetsize***

Returns:

0 on success, or appropriate error otherwise

2.9.1.6 int rt_sigqueueinfo (LinuxPid *pid*, int *sig*, siginfo_t $*$ *siginfo*)

Note:

rt_sigqueueinfo, syscall 178 (32 bit) and 129 (64 bit).
Currently only supports sending SIGALRM

Parameters:

$\leftarrow$ ***pid***

$\leftarrow$ ***sig***

$\leftarrow$ ***siginfo***

Returns:

0 on success, or appropriate error otherwise

2.9.1.7 int rt_sigreturn (void ∗ *contextAddr*, void ∗ *baseAddr*)

Note:

rt_sigreturn, syscall 173 (32-bit) and 15 (64-bit).
This syscall is supported, but shouldn't be called directly.

Parameters:

← ***contextAddr***

← ***baseAddr***

Returns:

0 on success, or appropriate error otherwise

2.9.1.8 long rt_sigsuspend (sigset_t ∗ *mask*, size_t *sigsetsize*)

Note:

rt_sigsuspend, syscall 179 (32-bit) and 130 (64-bit).
This syscall is fully supported.

Parameters:

← ***mask***

← ***sigsetsize***

Returns:

EINTR

2.9.1.9 int rt_sigtimedwait (const sigset_t ∗ *set*, siginfo_t ∗ *info*, const struct struct timespec ∗ *timeout*, size_t *sigsetsize*)

Note:

rt_sigtimedwait, syscall 177 (32 bit) and 128 (64 bit).
This syscall is fully supported.

Parameters:

← ***set***

→ ***info***

← ***timeout***

← ***sigsetsize***

Returns:

Signal that was pending, error otherwise.

2.9.1.10 int sigaltstack (stack_t ∗ *ss*, stack_t ∗ *oss*)

Note:

sigaltstack, syscall 186 (32-bit) and 131 (64-bit).
This syscall is fully supported.

Parameters:

→ ***ss***

← ***oss***

Returns:

0 on success, or appropriate error otherwise

2.9.1.11 long tgkill (int *tgid*, int *pid*, int *sig*)

Note:

tgkill, syscall 270 (32-bit) and 234 (64-bit)
This syscall is fully supported.

Parameters:

← ***tgid***

← ***pid***

← ***sig***

Returns:

0 on success, appropriate error otherwise

2.9.1.12 int tkill (int *tid*, int *sig*)

Note:

tkill, syscall 238 (32-bit) and 200 (64-bit).
This syscall is fully supported.

Parameters:

← ***tid***

← ***sig***

Returns:

0 on success, appropriate error otherwise

2.10 Network system calls

System calls for managing network sockets and communication.

Functions

- int64 accept (LinuxFd socketfd, UserVA name, UserVA nameLen, int flags)

  Accept a connection on the specified socket.

- int64 accept (LinuxFd socketfd, UserVA name, UserVA nameLen)

  Accept a connection on the specified socket.

- int64 bind (int socketfd, LinuxSocketName ∗name, uint32 nameLen)

  Bind the socket to the specified local address.

- int64 connect (int socketfd, LinuxSocketName ∗name, uint32 nameLen)

  Connect socket to the specified address.

- int64 getpeername (int socketfd, UserVA name, UserVA nameLen)

  Get the name of the connected peer.

- int64 getsockname (int socketfd, UserVA name, UserVA nameLen)

  Get the name of the specified socket.

- int64 getsockopt (int socketfd, int level, int optName, UserVA optVal, UserVA optLen)

  Get options on a socket.

- int64 listen (LinuxFd socketfd, int backlog)

  Listen for connections on a socket.

- int64 recvfrom (LinuxFd socketfd, UserVA buf, size_t len, unsigned int flags, LinuxSocketName ∗name, uint32 ∗nameLen)

  Receive data on a socket.

- int64 recvmsg (LinuxFd socketfd, UserVA msg, int flags)

  Receive a message from a socket.

- int64 sendmmsg (LinuxFd socketfd, UserVA msgvec, unsigned int vlen, int flags)

  Send several messages to a socket.

- int64 sendmsg (LinuxFd socketfd, UserVA msg, int flags)

  Send a message to a socket.

- int64 sendto (int socketfd, UserVA buf, uint32 len, uint32 flags, LinuxSocketName ∗name, uint32 nameLen)

  Send data on a socket.

- int64 setsockopt (int socketfd, int level, int optName, UserVA optVal, int optLen)

  Set socket options.

- int64 shutdown (int socketfd, int how)

  Shut down all or part of a full duplex socket connection.

- int64 socket (int family, int type, int protocol)

  Create a new socket for the given family, type and protocol.

- int socketcall (uint32 whichCall, UserVA userArgs)

  common entry for socket operations

- int64 socketpair (int family, int type, int protocol, UserVA socketfds)

  Create a pair of connected sockets.

2.10.1 Function Documentation

2.10.1.1 int64 accept (LinuxFd *socketfd*, UserVA *name*, UserVA *nameLen*, int *flags*)

Note:

32 bit: socketcall, syscall 102 with whichCall == SYS_ACCEPT4
64 bit: accept, syscall 288

Parameters:

← ***socketfd*** Socket file descriptor
← ***name*** Socket address (varies by address family)
← ***nameLen*** Socket address length
← ***flags*** Configuration for the new file descriptor

Returns:

non-negative descriptor for the accepted socket on success, -1 on failure

---

2.10.1.2 int64 accept (LinuxFd *socketfd*, UserVA *name*, UserVA *nameLen*)

Note:

32 bit: socketcall, syscall 102 with whichCall == SYS_ACCEPT
64 bit: accept, syscall 43

Parameters:

← ***socketfd*** Socket file descriptor
← ***name*** Socket address (varies by address family)
← ***nameLen*** Socket address length

Returns:

non-negative descriptor for the accepted socket on success, -1 on failure

---

2.10.1.3 int64 bind (int *socketfd*, LinuxSocketName ∗ *name*, uint32 *nameLen*)

Note:

32 bit: socketcall, syscall 102 with whichCall == SYS_BIND
64 bit: bind, syscall 49

Parameters:

← ***socketfd*** Socket file descriptor

← ***name*** Socket address (varies by address family)

← ***nameLen*** Socket address length

Returns:

0 on success, -1 on failure

Postcondition:

Binds a socket in the kernel.

2.10.1.4 int64 connect (int *socketfd*, LinuxSocketName ∗ *name*, uint32 *nameLen*)

Note:

32 bit: socketcall, syscall 102 with whichCall == SYS_CONNECT
64 bit: connect, syscall 42

Parameters:

← ***socketfd*** Socket file descriptor

← ***name*** Socket address (varies by address family)

← ***nameLen*** Socket address length

Returns:

0 on success, -1 on failure

Postcondition:

Connects the socket to the address.

2.10.1.5 int64 getpeername (int *socketfd*, UserVA *name*, UserVA *nameLen*)

Note:

32 bit: socketcall, syscall 102 with whichCall == SYS_GETPEERNAME
64 bit: getpeername, syscall 52

Parameters:

← ***socketfd*** Socket file descriptor

← ***name*** Socket address (varies by address family)

← ***nameLen*** Socket address length

Returns:

0 on success, -1 on failure

2.10.1.6 int64 getsockname (int *socketfd*, UserVA *name*, UserVA *nameLen*)

Note:

32 bit: socketcall, syscall 102 with whichCall == SYS_GETSOCKNAME
64 bit: getsockname, syscall 51

Parameters:

← ***socketfd*** Socket file descriptor

← ***name*** Socket address (varies by address family)

← ***nameLen*** Socket address length

Returns:

0 on success, -1 on failure

2.10.1.7 int64 getsockopt (int *socketfd*, int *level*, int *optName*, UserVA *optVal*, UserVA *optLen*)

Note:

32 bit: socketcall, syscall 102 with whichCall == SYS_GETSOCKOPT
64 bit: getsockopt, syscall 55
Unix domain socket does not support this call.
There is a special case with SO_PEERCRED where the call returns 0
and initializes optVal to zero.

Parameters:

← ***socketfd*** Socket file descriptor

← ***level*** Socket level

← ***optName*** Option name

→ ***optVal*** Option value

→ ***optLen*** Option length

Returns:

0 on success, -1 on failure

2.10.1.8 int64 listen (LinuxFd *socketfd*, int *backlog*)

Note:

32 bit: socketcall, syscall 102 with whichCall == SYS_LISTEN
64 bit: listen, syscall 50

Parameters:

← ***socketfd*** Socket file descriptor

← ***backlog*** Maximum length of pending connection queue

Returns:

0 on success, -1 on failure

2.10.1.9 int64 recvfrom (LinuxFd *socketfd*, UserVA *buf*, size_t *len*, unsigned int *flags*, LinuxSocketName ∗ *name*, uint32 ∗ *nameLen*)

Note:

32 bit: socketcall, syscall 102 with whichCall == SYS_RECVFROM
64 bit: recvfrom, syscall 45

Parameters:

← ***socketfd*** Socket file descriptor

→ ***buf*** Message buffer

← ***len*** Message buffer length

← ***flags*** Message flags

→ ***name*** Socket source address (varies by address family)

→ ***nameLen*** Socket source address length

Returns:

Number of bytes received on success, -1 on failure

2.10.1.10 int64 recvmsg (LinuxFd *socketfd*, UserVA *msg*, int *flags*)

Note:

32 bit: socketcall, syscall 102 with whichCall == SYS_RECVMSG
64 bit: recvmsg, syscall 47

Parameters:

← ***socketfd*** Socket file descriptor

→ ***msg*** Message struct

← ***flags*** Message flags

Returns:

Number of bytes received on success, -1 on failure

2.10.1.11 int64 sendmmsg (LinuxFd *socketfd*, UserVA *msgvec*, unsigned int *vlen*, int *flags*)

Note:

64 bit: sendmmsg, syscall 307

Parameters:

← ***socketfd*** Socket file descriptor

← ***msgvec*** Message struct vector

← ***vlen*** Number of messages

← ***flags*** Message flags

Returns:

Number of messages sent on success, -1 on failure

2.10.1.12 int64 sendmsg (LinuxFd *socketfd*, UserVA *msg*, int *flags*)

Note:

32 bit: socketcall, syscall 102 with whichCall == SYS_SENDMSG
64 bit: sendmsg, syscall 46

Parameters:

← ***socketfd*** Socket file descriptor

← ***msg*** Message struct

← ***flags*** Message flags

Returns:

Number of bytes sent on success, -1 on failure

2.10.1.13 int64 sendto (int *socketfd*, UserVA *buf*, uint32 *len*, uint32 *flags*, LinuxSocketName ∗ *name*, uint32 *nameLen*)

Note:

32 bit: socketcall, syscall 102 with whichCall == SYS_SENDTO
64 bit: sendto, syscall 44

Parameters:

← ***socketfd*** Socket file descriptor

← ***buf*** Message buffer

← ***len*** Message buffer length

← ***flags*** Message flags

← ***name*** Socket target address (varies by address family)

← ***nameLen*** Socket target address length

Returns:

Number of bytes sent on success, -1 on failure

2.10.1.14 int64 setsockopt (int *socketfd*, int *level*, int *optName*, UserVA *optVal*, int *optLen*)

Note:

32 bit: socketcall, syscall 102 with whichCall == SYS_SETSOCKOPT
64 bit: setsockopt, syscall 54

Parameters:

← ***socketfd*** Socket file descriptor

← ***level*** Socket level

← ***optName*** Option name

← ***optVal*** Option value

← ***optLen*** Option length

Returns:

0 on success, -1 on failure

2.10.1.15 int64 shutdown (int *socketfd*, int *how*)

Note:

32 bit: socketcall, syscall 102 with whichCall == SYS_SHUTDOWN
64 bit: shutdown, syscall 48

Parameters:

← ***socketfd*** Socket file descriptor

← ***how***

Returns:

0 on success, -1 on failure

Postcondition:

Shuts down all or part of a socket connection

2.10.1.16 int64 socket (int *family*, int *type*, int *protocol*)

Note:

32 bit: socketcall, syscall 102 with whichCall == SYS_SOCKET
64 bit: socket, syscall 41

Parameters:

← ***family*** Communication domain (i.e. PF_INET)

← ***type*** Communication semantics (i.e. SOCK_STREAM)

← ***protocol*** Socket protocol (i.e. IP)

Returns:

A positive file description on success, negative otherwise

Postcondition:

Creates a socket in the kernel. This is resource which should be limited and monitored.

2.10.1.17 int socketcall (uint32 *whichCall*, UserVA *userArgs*)

Note:

socketcall, syscall 102 (32 bit only)
Support: 100%, but not all socket semantics have been tested
Error case: 100%

Parameters:

← ***whichCall*** Which function to invoke

↔ ***userArgs*** Function paramters

Returns:

Varies depending upon the whichCall parameter.

Postcondition:

Varies depending upon the whichCall parameter.

2.10.1.18 int64 socketpair (int *family*, int *type*, int *protocol*, UserVA *socketfds*)

Note:

32 bit: socketcall, syscall 102 with whichCall == SYS_SOCKETPAIR
64 bit: socketpair, syscall 53

Parameters:

← ***family*** Socket file descriptor

← ***type*** Socket type

← ***protocol*** Socket address (varies by address family)

→ ***socketfds*** New sockets

Returns:

0 on success, -1 on failure

2.11 Configure system params

Support for the linux 'sysctl' interface. Linux userland apps have mostly deprecated the use of this call in favor of /proc-based lookups. Currently, sysctl gets used by applications to differentiate between running as a UserWorld in the vmkernel vs running with a linux kernel in the Service Console.

Functions

- int sysctl (struct __sysctl_args userArgs)

  Get and set kernel parameters.

2.11.1 Function Documentation

2.11.1.1 int sysctl (struct __sysctl_args *userArgs*)

Note:

sysctl, syscall 149 (32-bit) and 156 (64-bit)
Support: Only get from some of CTL_KERN, CTL_NET and CTL_FS supported.

Parameters:

↔ ***userArgs*** Buffer for kernel parameters

Returns:

0 on success, errno on failure

2.12 Get information about the system configuration

Support for the linux 'sysinfo' and 'uname' interfaces.

Functions

- int64 LinuxSysinfo_Uname (UserVA userUtsName)
  Get information about the kernel.
- int64 sysinfo (UserVA sysInfo)
  Get information about system statistics.

2.12.1 Function Documentation

2.12.1.1 int64 LinuxSysinfo_Uname (UserVA *userUtsName*)

Note:

: 32 bit: uname, syscall 122
: 64 bit: uname, syscall 63
: On ESXi, returns system information about the vmkernel.

Parameters:

→ ***userUtsName*** Structure containing kernel information

Returns:

0 on success, -1 otherwise

2.12.1.2 int64 sysinfo (UserVA *sysInfo*)

Note:

32 bit: sysinfo, syscall syscall 116
64 bit: sysinfo, syscall syscall 99
Only fills only uptime field of the structure

Parameters:

→ ***sysInfo*** Buffer for system information

Returns:

0 on success, -1 otherwise

2.13 system calls

System calls related to threading.

Functions

- int64 clone (int32 linuxFlags, UserVA stack, UserVA parentTidptr, UserVA tlsDesc, UserVA childTidptr)

  Creates a new process.

- int64 futex (UserVA l_uaddr, int op, int val, UserVA l_timeout, UserVA l_uaddr2, int val3)

  Fast Userspace Locking.

- int64 get_robust_list (LinuxPid pid, UserVA head, UserVA len_ptr)

  Retrieves the robust list head pointer.

- int64 getpgid (LinuxPid pid)

  get process group

- int64 getpgrp (void)

  get process group

- int64 getpid (void)

  Returns the process ID of the calling process.

- int64 getppid (void)

  get parent's process ID (limited support)

- int64 getsid (LinuxPid pid)

  Returns the session ID of the process pid.

- int64 gettid (void)

  gettid.

- int64 nanosleep (UserVA requestTimespec, UserVA remainTimespec)

  nano sleep

- int64 sched_get_priority_max (int policy)

  get static priority max for the specified sched policy

- int64 sched_get_priority_min (int policy)

  get static priority min for the specified sched policy

- int64 sched_getaffinity (LinuxPid tid, LinuxSizeT masksize, UserVA mask)

  Get a process's CPU affinity mask.

- int64 sched_getparam (LinuxPid pid, UserVA sched_param)

  get sched parameters

- int64 sched_getscheduler (LinuxPid pid)

  get scheduling policy parameters

- int64 sched_setaffinity (LinuxPid tid, LinuxSizeT masksize, UserVA mask)

  Set a process's CPU affinity mask.

- int64 sched_setparam (LinuxPid pid, UserVA sched_param)

  set sched parameters

- int64 sched_setscheduler (LinuxPid pid, int32 policy, UserVA sched_param)

  set scheduling policy/parameters

- int64 sched_yield (void)

  Yield the processor.

- int64 set_robust_list (UserVA head, size_t len)

  Stores the robust list head pointer.

- int64 set_tid_address (UserVA l_tidptr)

  Set pointer to thread ID.

- int64 setpgid (LinuxPid pid, LinuxPid pgid)

  set process group

- int64 setsid (void)

  Creates a session and sets the session ID.

- int64 wait4 (LinuxPid linuxPid, UserVA userOutStatus, int32 options, UserVA userRusage)

  Wait for process to change state (not fully supported).

- int waitpid (LinuxPid pid, UserVA userOutStatus, int32 options)

  Wait for process to change state.

2.13.1 Function Documentation

2.13.1.1 int64 clone (int32 *linuxFlags*, UserVA *stack*, UserVA *parentTidptr*, UserVA *tlsDesc*, UserVA *childTidptr*)

Note:

x86_32 : syscall 120. aarch64 : syscall 220. clone(linuxFlags, stack, parentTidptr, tlsDesc, childTidptr)
x86_64 : syscall 56. clone(linuxFlags, stack, parentTidptr, childTidptr, tlsDesc)
Supported flags : CLONE_FS CLONE_FILES CLONE_SIGHAND CLONE_VM CLONE_PTRACE CLONE_THREAD CLONE_SYSVSEM CLONE_SETTLS CLONE_PARENT_SETTID CLONE_-CHILD_SETTID CLONE_CHILD_CLEARTID CLONE_DETACHED

Parameters:

← ***linuxFlags*** clone flags

← ***stack*** child's user stack

← ***parentTidptr*** parent's TLS

← ***tlsDesc*** TLS descriptor

→ ***childTidptr*** child's TLS

Returns:

On success returns the new pid, -1 on failure

2.13.1.2 int64 futex (UserVA *l_uaddr*, int *op*, int *val*, UserVA *l_timeout*, UserVA *l_uaddr2*, int *val3*)

Note:

syscall 240 (32-bit) and 202 (64-bit)
Limited support, supported op : FUTEX_WAIT FUTEX_WAKE

Parameters:

← ***l_uaddr*** ptr to counter

← ***op*** operation

← ***val*** counter value

← ***l_timeout*** timeout

← ***l_uaddr2*** Ignored

← ***val3*** Ignored

Returns:

0 on success, error code otherwise

2.13.1.3 int64 get_robust_list (LinuxPid *pid*, UserVA *head*, UserVA *len_ptr*)

Note:

syscall 312 (32-bit) and 274 (64-bit)

Parameters:

← ***pid*** pid whose robust list head is being requested

→ ***head*** user ptr to store the robust list head

→ ***len_ptr*** size of the robust list head struct

Returns:

0 on success, and errno on failure

2.13.1.4 int64 getpgid (LinuxPid *pid*)

Note:

syscall 132 (32-bit) and 121 (64-bit)

Parameters:

← ***pid*** process ID

Returns:

Returns the process group identifier (pgid) of pid. Error code on failure.

2.13.1.5 int64 getpgrp (void)

Note:

syscall 65 (32-bit) and 111 (64-bit)

Returns:

Returns the process group identifier (pgid) of the current process.

2.13.1.6 int64 getpid (void)

Note:

syscall 20 (32-bit) and 39 (64-bit)

Returns:

On success returns the pid, -1 on failure

2.13.1.7 int64 getppid (void)

Note:

getppid, syscall 64 (32-bit) and 110 (64-bit)
If the the parent process is multi-threaded this call returns the main process instead of the specifc thread that forked the child.

Returns:

parent’s process ID or -1 on failure

2.13.1.8 int64 getsid (LinuxPid *pid*)

Note:

syscall 147 (32-bit) and 124 (64-bit)

Parameters:

← ***pid*** process ID

Returns:

Returns the session ID of the process pid.

2.13.1.9 int64 gettid (void)

Note:

syscall 224 (32-bit) and 186 (64-bit)

Returns:

Returns the caller's thread ID.

2.13.1.10 int64 nanosleep (UserVA *requestTimespec*, UserVA *remainTimespec*)

Note:

nanosleep, syscall 162 (32-bit) and 35 (64-bit)

Parameters:

← ***requestTimespec*** requested time
→ ***remainTimespec*** remaining time

Returns:

0 on sucess -1 on failure

2.13.1.11 int64 sched_get_priority_max (int *policy*)

Note:

syscall 159 (32-bit) and 146 (64-bit)

Parameters:

← ***policy*** sched policy

Returns:

max priority

2.13.1.12 int64 sched_get_priority_min (int *policy*)

Note:

syscall 160 (32-bit) and 147 (64-bit)

Parameters:

← ***policy*** sched policy

Returns:

min priority

2.13.1.13 int64 sched_getaffinity (LinuxPid *tid*, LinuxSizeT *masksize*, UserVA *mask*)

Note:

syscall 242 (32-bit) and 204 (64-bit)

Parameters:

← ***tid***

← ***masksize***

→ ***mask***

Returns:

number of bytes in mask on success, error code otherwise

2.13.1.14 int64 sched_getparam (LinuxPid *pid*, UserVA *sched_param*)

Note:

syscall 155 (32-bit) and 143 (64-bit)

Parameters:

← ***pid*** process ID

→ ***sched_param*** sched parameters

Returns:

0 on success, error otherwise

2.13.1.15 int64 sched_getscheduler (LinuxPid *pid*)

Note:

syscall 157 (32-bit) and 145 (64-bit)
limited support. Always return SCHED_OTHER

Parameters:

← ***pid*** process ID

Returns:

SCHED_OTHER

2.13.1.16 int64 sched_setaffinity (LinuxPid *tid*, LinuxSizeT *masksize*, UserVA *mask*)

Note:

syscall 241 (32-bit) and 203 (64-bit)

Parameters:

← ***tid***

← ***masksize***

← ***mask***

Returns:

0 on success, error code otherwise

2.13.1.17 int64 sched_setparam (LinuxPid *pid*, UserVA *sched_param*)

Note:

syscall 154 (32-bit) and 142 (64-bit)

Parameters:

← ***pid*** process ID

← ***sched_param*** sched parameters

Returns:

0 on success, error otherwise

2.13.1.18 int64 sched_setscheduler (LinuxPid *pid*, int32 *policy*, UserVA *sched_param*)

Note:

syscall 156 (32-bit) and 144 (64-bit)

Parameters:

← ***pid*** process ID
← ***policy*** sched policy
← ***sched_param*** sched parameters

Returns:

0 on success, error otherwise

2.13.1.19 int64 sched_yield (void)

Note:

syscall 158 (32-bit) and 24 (64-bit)

Returns:

returns zero.

2.13.1.20 int64 set_robust_list (UserVA *head*, size_t *len*)

Note:

syscall 311 (32-bit) and 273 (64-bit)

Parameters:

← ***head*** user ptr to the robust list head
← ***len*** size of the robust list head struct

Returns:

0 on success, and errno on failure

2.13.1.21 int64 set_tid_address (UserVA *l_tidptr*)

Note:

syscall 258 (32-bit) and 218 (64-bit)

Parameters:

← ***l_tidptr*** tid ptr

Returns:

Returns pid of the current process.

2.13.1.22 int64 setpgid (LinuxPid *pid*, LinuxPid *pgid*)

Note:

setpgid, syscall 57 (32-bit) and 109 (64-bit)
If cartel leader is dead, cartel won't be found (PR133355).

Parameters:

← ***pid*** processID
← ***pgid*** groupID

Returns:

0 on sucess -1 on failure

2.13.1.23 int64 setsid (void)

Note:

syscall 66 (32-bit) and 112 (64-bit)

Returns:

New process session id.

2.13.1.24 int64 wait4 (LinuxPid *linuxPid*, UserVA *userOutStatus*, int32 *options*, UserVA *userRusage*)

Note:

syscall 114 (32-bit) and 61 (64-bit)
Interrupt semantics – EINTR on delivery of unblocked signal – not supported.
Supported flags : WNOHANG WUNTRACED WCONTINUED WALL WCLONE

Parameters:

← ***linuxPid*** Target pid
→ ***userOutStatus*** If not NULL, save return status
← ***options*** Option flag
→ ***userRusage*** rusage

Returns:

On success returns the process ID of the terminated child, -1 on failure

---

2.13.1.25 int waitpid (LinuxPid *pid*, UserVA *userOutStatus*, int32 *options*)

Note:

32 bit : syscall 7
64 bit : NOT SUPPORTED
Supported flags : WNOHANG WUNTRACED WCONTINUED WALL WCLONE

Parameters:

$\leftarrow$ ***pid*** Target pid

$\rightarrow$ ***userOutStatus*** If not NULL, save return status

$\leftarrow$ ***options*** Option flag

Returns:

On success returns the process ID of the terminated child, -1 on failure

2.14 Timer/clock system calls

System calls for managing timers and clocks.

Functions

- int adjtimex (struct timex ∗userTimex)

    Read and set NTP clock parameters.

- int alarm (unsigned int secs)

    Sets an alarm for the delivery of SIGALRM.

- int clock_getres (clockid_t clockId, struct timespec ∗userTS)

    Get the resolution of the clock specified by clockId into the area specified by userTS.

- int clock_gettime (clockid_t clockId, struct timespec ∗userTS)

    Gets the clock specified by clockId into the area specified by userTS.

- int clock_settime (clockid_t clockId, const struct timespec ∗userTS)

    Sets the clock specified by clockId using the time specified by userTS.

- int getitimer (int which, struct itimerval ∗userItv)

    Get the value of an interval timer.

- int gettimeofday (struct timeval ∗tvp, struct timezone ∗tzp)

    Get the time of day in seconds and microseconds.

- int setitimer (int which, const struct itimerval ∗userItv, struct itimerval ∗userOitv)

    Set the value of an interval timer.

- int settimeofday (const struct timeval ∗tvp, const struct timezone ∗tzp)

    Changes system clock time of day in seconds and microseconds.

- time_t time (time_t ∗tm)

    Returns the time since the unix epoch in seconds.

- int times (struct tms ∗timesArgBuf)

    Get current user and system times, in ticks.

2.14.1 Function Documentation

2.14.1.1 int adjtimex (struct timex ∗ *userTimex*)

Note:

32 bit: adjtimex, syscall 124
64 bit: adjtimex, syscall 159

Parameters:

$\leftrightarrow$ ***userTimex*** NTP clock parameter data

Returns:

TIME_STATE_xxx, or negative value on error

Postcondition:

May set the kernel NTP clock or adjust clock parameters.

2.14.1.2 int alarm (unsigned int *secs*)

Note:

32 bit: alarm, syscall 27
64 bit: alarm, syscall 37

Parameters:

$\leftarrow$ ***secs*** Time in seconds when alarm should be delivered

Returns:

Number of seconds remaining before previous timer would have fired, or 0 if there was no previous timer. Not allowed to fail.

Postcondition:

May set the kernel NTP clock

2.14.1.3 int clock_getres (clockid_t *clockId*, struct timespec $*$ *userTS*)

Note:

32 bit: clock_getres, syscall 266
64 bit: clock_getres, syscall 229
Support: 60%, only CLOCK_REALTIME, CLOCK_MONOTONIC, CLOCK_MONOTONIC_RAW
Error case: 100%

Parameters:

$\leftarrow$ ***clockId*** Specified clock

$\rightarrow$ ***userTS*** Pointer to timespec

Returns:

0 on success, non-zero otherwise

2.14.1.4 int clock_gettime (clockid_t *clockId*, struct timespec ∗ *userTS*)

Note:

32 bit: clock_gettime, syscall 265
64 bit: clock_gettime, syscall 228
supports CLOCKTYPE_REALTIME, CLOCKTYPE_MONOTONIC, CLOCK_MONOTONIC_-RAW

Parameters:

← ***clockId*** Specified clock
→ ***userTS*** Pointer to timespec

Returns:

0 on success, non-zero otherwise

2.14.1.5 int clock_settime (clockid_t *clockId*, const struct timespec ∗ *userTS*)

Note:

32 bit: clock_settime, syscall 264
64 bit: clock_settime, syscall 227

Parameters:

← ***clockId*** Specified clock
← ***userTS*** Pointer to timespec

Returns:

0 on success, non-zero otherwise

Postcondition:

May set the system clock.

2.14.1.6 int getitimer (int *which*, struct itimerval ∗ *userItv*)

Note:

32 bit: getitimer, syscall 105
64 bit: getitimer, syscall 36

Parameters:

← ***which*** Specified timer
→ ***userItv*** Pointer to itimer value

Returns:

0 on success, non-zero otherwise

2.14.1.7 int gettimeofday (struct timeval ∗ *tvp*, struct timezone ∗ *tzp*)

Note:

32 bit: gettimeofday, syscall 78
64 bit: gettimeofday, syscall 96
timezone argument unsupported

Parameters:

→ ***tvp*** Pointer to the returned timeval data

→ ***tzp*** Pointer to the returned timezone data

Returns:

0 on success, non-zero otherwise

2.14.1.8 int setitimer (int *which*, const struct itimerval ∗ *userItv*, struct itimerval ∗ *userOitv*)

Note:

32 bit: setitimer, syscall 104
64 bit: setitimer, syscall 38
ITIMER_VIRTUAL runs during both user and system time

Parameters:

← ***which*** Specified timer

← ***userItv*** Pointer to itimer value

→ ***userOitv*** Pointer to old itimer value

Returns:

0 on success, non-zero otherwise

Postcondition:

Configures an interval timer. This is resource which should be limited and monitored.

2.14.1.9 int settimeofday (const struct timeval ∗ *tvp*, const struct timezone ∗ *tzp*)

Note:

32 bit: settimeofday, syscall 79
64 bit: settimeofday, syscall 164
timezone argument unsupported

Parameters:

← ***tvp*** Pointer to the returned timeval data

← ***tzp*** Pointer to the returned timezone data

Returns:

0 on success, non-zero otherwise

Postcondition:

May set the system clock. Should be restricted to super-user.

2.14.1.10 time_t time (time_t ∗ *tm*)

Note:

32 bit: time, syscall 13
64 bit: time, syscall 201

Parameters:

→ ***tm*** time value in seconds

Returns:

time of day in seconds since the epoch

2.14.1.11 int times (struct tms ∗ *timesArgBuf*)

Note:

32 bit: times, syscall 43
64 bit: times, syscall 100
kernel sets CLK_TICK to 10ms

Parameters:

→ ***timesArgBuf*** Pointer to struct tms buffer

Returns:

: Clock ticks since boot

Index