NAME

cap_new, **cap_getrights** — System calls to manipulate capabilities

LIBRARY

Standard C Library (libc, −lc)

SYNOPSIS

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

int
cap_new(int fd, cap_rights_t rights);

int
cap_getrights(int fd, cap_rights_t *rightsp);
```

DESCRIPTION

Capabilities are special file descriptors derived from an existing file descriptor, such as one returned by fhopen(2), kqueue(2), mq_open(2), open(2), pipe(2), shm_open(2), socket(2), or socketpair(2), but with a restricted set of permitted operations determined by a rights mask set when the capability is created. These restricted rights cannot be changed after the capability is created, although further capabilities with yet more restricted rights may be created from an existing capability. In every other sense, a capability behaves in the same way as the file descriptor it was created from.

cap_new() creates a new capability for the existing file descriptor *fd*, and returns a file descriptor for it. Operations on the capability will be limited to those permitted by *rights*, which is static for the lifetime of the capability. If *fd* refers to an existing capability, then *rights* must be equal to or a subset of the rights on that capability. As with dup(2) and dup2(2), many properties are shared between the new capability and the existing file descriptor, including open file flags, blocking disposition, and file offset. Many applications will prefer to use the cap_limitfd(3) library call, part of libcapsicum(3), as it offers a more convenient interface.

cap_getrights() queries the rights associated with the capability referred to by file descriptor *fd*.

These system calls, when combined with cap_enter(2), may be used to construct process sandboxes with highly granular rights assignment.

RIGHTS

The following rights may be specified in a new capability rights mask:

CAP_ACCEPT	Permit accept(2).
CAP_ACL_CHECK	Permit checking of an ACL on a file descriptor; there is no cross-reference for this system call.
CAP_ACL_DELETE	Permit acl_delete_fd_np(3).
CAP_ACL_GET	Permit acl_get_fd(3) and acl_get_fd_np(3).
CAP_ACL_SET	Permit acl_set_fd(3) and acl_set_fd_np(3).
CAP_BIND	Permit bind(2). Note that sockets can also become bound implicitly as a result of connect(2) or send(2), and that socket options set with setsockopt(2) may also affect binding behavior.
CAP_CONNECT	Permit connect(2); also required for sendto(2) with a non-NULL destination address.

CAP_EVENT	Permit select(2), poll(2), and kevent(2) to be used in monitoring the file descriptor for events.
CAP_FEXECVE	Permit fexecve(2); CAP_READ will also be required.
CAP_EXTATTR_DELETE	Permit extattr_delete_fd(2).
CAP_EXTATTR_GET	Permit extattr_get_fd(2).
CAP_EXTATTR_LIST	Permit extattr_list_fd(2).
CAP_EXTATTR_SET	Permit extattr_set_fd(2).
CAP_FCHDIR	Permit fchdir(2).
CAP_FCHFLAGS	Permit fchflags(2).
CAP_FCHMOD	Permit fchmod(2).
CAP_FCHOWN	Permit fchown(2).
CAP_FCNTL	Permit fcntl(2); be aware that this call provides indirect access to other operations, such as flock(2).
CAP_FLOCK	Permit flock(2) and related calls.
CAP_FPATHCONF	Permit fpathconf(2).
CAP_FSCK	Permit UFS background-fsck operations on the descriptor.
CAP_FSTAT	Permit fstat(2).
CAP_FSTATFS	Permit fstatfs(2).
CAP_FSYNC	Permit aio_fsync(2) and fsync(2).
CAP_FTRUNCATE	Permit ftruncate(2).
CAP_FUTIMES	Permit futimes(2).
CAP_GETPEERNAME	Permit getpeername(2).
CAP_GETSOCKNAME	Permit getsockname(2).
CAP_GETSOCKOPT	Permit getsockopt(2).
CAP_IOCTL	Permit ioctl(2). Be aware that this system call has enormous scope, including potentially global scope for some objects.
CAP_KEVENT	Permit kevent(2); CAP_EVENT is also required on file descriptors that will be monitored using kevent(2).
CAP_LISTEN	Permit listen(2); not much use (generally) without CAP_BIND.
CAP_LOOKUP	Permit the file descriptor to be used as a starting directory for calls such as linkat(2), openat(2), and unlinkat(2). Note that these calls are not available in capability mode as they manipulate a global name space; see cap_enter(2) for details.
CAP_MAC_GET	Permit mac_get_fd(3).
CAP_MAC_SET	Permit mac_set_fd(3).
CAP_MMAP	Permit mmap(2); specific invocations may also require CAP_READ or CAP_WRITE.

CAP_PDGETPID	Permit pdgetpid(2).
CAP_PDKILL	Permit pdkill(2).
CAP_PDWAIT	Permit pdwait4(2).
CAP_PEELOFF	Permit sctp_peeloff(2).
CAP_READ	Allow aio_read(2), pread(2), read(2), recv(2), recvfrom(2), recvmsg(2), and related system calls. For files and other seekable objects, CAP_SEEK may also be required.
CAP_REVOKE	Permit frevoke(2) in certain ABI compatibility modes that support this system call.
CAP_SEEK	Permit operations that seek on the file descriptor, such as lseek(2), but also required for I/O system calls that modify the file offset, such as read(2) and write(2).
CAP_SEM_GETVALUE	Permit sem_getvalue(3).
CAP_SEM_POST	Permit sem_post(3).
CAP_SEM_WAIT	Permit sem_wait(3) and sem_trywait(3).
CAP_SETSOCKOPT	Permit setsockopt(2); this controls various aspects of socket behavior and may affect binding, connecting, and other behaviors with global scope.
CAP_SHUTDOWN	Permit explicit shutdown(2); closing the socket will also generally shut down any connections on it.
CAP_TTYHOOK	Allow configuration of TTY hooks, such as snp(4), on the file descriptor.
CAP_WRITE	Allow aio_write(2), pwrite(2), send(2), sendmsg(2), sendto(2), write(2), and related system calls. For files and other seekable objects, CAP_SEEK may also be required. For sendto(2) with a non-NULL connection address, CAP_CONNECT is also required.

CAVEAT

The **cap_new**() system call and the capabilities it creates may be used to assign fine-grained rights to sandboxed processes running in capability mode. However, the semantics of objects accessed via file descriptors are complex, so caution should be exercised in passing object capabilities into sandboxes.

RETURN VALUES

If successful, **cap_new**() returns a non-negative integer, termed a file descriptor. It returns -1 on failure, and sets *errno* to indicate the error.

The **cap_getrights**() function returns the value 0 if successful; otherwise the value −1 is returned and the global variable *errno* is set to indicate the error.

ERRORS

cap_new() may return the following errors:

[EBADF]	The *fd* argument is not a valid active descriptor.

[EINVAL] An invalid right has been requested in *rights*.

[EMFILE] The process has already reached its limit for open file descriptors.

[ENFILE] The system file table is full.

[EPERM] *rights* contains requested rights not present in the current rights mask associated with the capability referenced by *fd*, if any.

cap_getrights() may return the following errors:

[EBADF] The *fd* argument is not a valid active descriptor.

[EINVAL] The *fd* argument is not a capability.

SEE ALSO

accept(2), aio_fsync(2), aio_read(2), aio_write(2), bind(2), cap_enter(2), connect(2), dup(2), dup2(2), extattr_delete_fd(2), extattr_get_fd(2), extattr_list_fd(2), extattr_set_fd(2), fchflags(2), fchown(2), fcntl(2), fexecve(2), fhopen(2), flock(2), fpathconf(2), fstat(2), fstatfs(2), fsync(2), ftruncate(2), futimes(2), getpeername(2), getsockname(2), getsockopt(2), ioctl(2), kevent(2), kqueue(2), linkat(2), listen(2), mmap(2), mq_open(2), open(2), openat(2), pdgetpid(2), pdkill(2), pdwait4(2), pipe(2), poll(2), pread(2), pwrite(2), read(2), recv(2), recvfrom(2), recvmsg(2), sctp_peeloff(2), select(2), send(2), sendmsg(2), sendto(2), setsockopt(2), shm_open(2), shutdown(2), socket(2), socketpair(2), unlinkat(2), write(2), acl_delete_fd_np(3), acl_get_fd(3), acl_get_fd_np(3), acl_set_fd_np(3), cap_limitfd(3), libcapsicum(3), mac_get_fd(3), mac_set_fd(3), sem_getvalue(3), sem_post(3), sem_trywait(3), sem_wait(3), capsicum(4), snp(4)

HISTORY

Support for capabilities and capabilities mode was developed as part of the TrustedBSD Project.

AUTHORS

These functions and the capability facility were created by Robert N. M. Watson at the University of Cambridge Computer Laboratory with support from a grant from Google, Inc.

BUGS

This man page should list the set of permitted system calls more specifically for each capability right.

Capability rights sometimes have unclear indirect impacts, which should be documented, or at least hinted at.