

**NAME**

**cap\_enter**, **cap\_getmode** - Capability mode system calls

**LIBRARY**

Standard C Library (libc, -lc)

**SYNOPSIS**

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

*int*

```
cap_enter(void);
```

*int*

```
cap_getmode(u_int *modep);
```

**DESCRIPTION**

**cap\_enter()** places the current process into capability mode, a mode of execution in which processes may only issue system calls operating on file descriptors or reading limited global system state. Access to global name spaces, such as file system or IPC name spaces, is prevented. If the process is already in a capability mode sandbox, the system call is a no-op. Future process descendants created with **fork(2)** or **pdfork(2)** will be placed in capability mode from inception.

When combined with **cap\_rights\_limit(2)**, **cap\_ioctls\_limit(2)**, **cap\_fcntls\_limit(2)**, **cap\_enter()** may be used to create kernel-enforced sandboxes in which appropriately-crafted applications or application components may be run.

**cap\_getmode()** returns a flag indicating whether or not the process is in a capability mode sandbox.

**RUN-TIME SETTINGS**

If the **kern.trap\_enotcap** sysctl MIB is set to a non-zero value, then for any process executing in a capability mode sandbox, any syscall which results in either an **ENOTCAPABLE** or **ECAPMODE** error also generates the synchronous **SIGTRAP** signal to the thread on the syscall return. On signal delivery, the *si\_errno* member of the *siginfo* signal handler parameter is set to the syscall error value, and the *si\_code* member is set to **TRAP\_CAP**.

See also the **PROC\_TRAPCAP\_CTL** and **PROC\_TRAPCAP\_STATUS** operations of the **procctl(2)** function for similar per-process functionality.

**RETURN VALUES**

The **cap\_enter()** and **cap\_getmode()** functions return the value 0 if successful; otherwise the value -1 is

returned and the global variable *errno* is set to indicate the error.

When the process is in capability mode, **cap\_getmode()** sets the flag to a non-zero value. A zero value means the process is not in capability mode.

## ERRORS

The **cap\_enter()** and **cap\_getmode()** system calls will fail if:

[ENOSYS]           The running kernel was compiled without **options CAPABILITY\_MODE**.

The **cap\_getmode()** system call may also return the following error:

[EFAULT]           Pointer *modep* points outside the process's allocated address space.

## SEE ALSO

cap\_fcntls\_limit(2), cap\_ioctls\_limit(2), cap\_rights\_limit(2), fexecve(2), procctl(2), cap\_sandboxed(3), capsicum(4), sysctl(9)

## HISTORY

The **cap\_getmode()** system call first appeared in FreeBSD 8.3. 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.

## CAVEATS

Creating effective process sandboxes is a tricky process that involves identifying the least possible rights required by the process and then passing those rights into the process in a safe manner. Consumers of **cap\_enter()** should also be aware of other inherited rights, such as access to VM resources, memory contents, and other process properties that should be considered. It is advisable to use **fexecve(2)** to create a runtime environment inside the sandbox that has as few implicitly acquired rights as possible.