NAME

mmap – map pages of memory

SYNOPSIS

#include <sys/types.h>
#include <sys/mman.h>

caddr_t mmap(caddr_t *addr*, **size_t** *len*, **int** *prot*, **int** *flags*, **int** *fildes*, **off_t** *off***);**

DESCRIPTION

The function **mmap( )** establishes a mapping between a process's address space and a virtual memory object. The format of the call is as follows:

pa = **mmap(***addr*, *len*, *prot*, *flags*, *fildes*, *off***);**

mmap establishes a mapping between the process's address space at an address *pa* for *len* bytes to the memory object represented by the file descriptor *fildes* at offset *off* for *len* bytes. The value of *pa* is an implementation-dependent function of the parameter *addr* and values of *flags*, further described below. A successful **mmap** call returns *pa* as its result. The address ranges covered by [*pa, pa* + *len*) and [*off, off* + *len*) must be legitimate for the possible (not necessarily current) address space of a process and the object in question, respectively.

mmap( ) allows [*pa, pa* + *len*) to extend beyond the end of the object, both at the time of the **mmap( )** and while the mapping persists, such as when the file was created just before the **mmap( )** and has no contents, or if the file is truncated. Any reference to addresses beyond the end of the object, however, will result in the delivery of a **SIGBUS** signal. In other words, **mmap( )** cannot be used to implicitly extend the length of files.

The mapping established by **mmap( )** replaces any previous mappings for the process's pages in the range [*pa, pa* + *len*).

Mappings established from *fildes* are not removed upon a **close**(2) of that descriptor. Use **munmap**(2) to remove a mapping.

The parameter *prot* determines whether read, write, execute, or some combination of accesses are permitted to the pages being mapped. The protection options are defined in **<sys/mman.h>** as:

PROT_READ	Page can be read.
PROT_WRITE	Page can be written.
PROT_EXEC	Page can be executed.
PROT_NONE	Page can not be accessed.

Not all implementations literally provide all possible combinations. **PROT_WRITE** is often implemented as **PROT_READ | PROT_WRITE** and **PROT_EXEC** as **PROT_READ | PROT_EXEC**. However, no implementation will permit a write to succeed where **PROT_WRITE** has not been set. The behavior of **PROT_WRITE** can be influenced by setting **MAP_PRIVATE** in the *flags* parameter, described below.

The parameter *flags* provides other information about the handling of the mapped pages. The options are defined in **<sys/mman.h>** as:

MAP_SHARED	Share changes.
MAP_PRIVATE	Changes are private.
MAP_FIXED	Interpret addr exactly.
MAP_NORESERVE	Don’t reserve swap space.

MAP_SHARED and **MAP_PRIVATE** describe the disposition of write references to the memory object. If **MAP_SHARED** is specified, write references will change the memory object. If **MAP_PRIVATE** is specified, the initial write reference will create a private copy of the memory object page and redirect the mapping to the copy. Either **MAP_SHARED** or **MAP_PRIVATE** must be specified, but not both. The mapping type is retained across a **fork**(2).

Note that the private copy is not created until the first write; until then, other users who have the object mapped **MAP_SHARED** can change the object.

MAP_FIXED informs the system that the value of *pa* must be *addr*, exactly. The use of **MAP_FIXED** is discouraged, as it may prevent an implementation from making the most effective use of system resources.

When **MAP_FIXED** is not set, the system uses *addr* in an implementation-defined manner to arrive at *pa*. The *pa* so chosen will be an area of the address space which the system deems suitable for a mapping of *len* bytes to the specified object. All implementations interpret an *addr* value of zero as granting the system complete freedom in selecting *pa*, subject to constraints described below. A non-zero value of *addr* is taken to be a suggestion of a process address near which the mapping should be placed. When the system selects a value for *pa*, it will never place a mapping at address **0**, nor will it replace any extant mapping, nor map into areas considered part of the potential data or stack “segments”.

MAP_NORESERVE specifies that no swap space be reserved for a mapping. Without this flag, the creation of a **MAP_PRIVATE** mapping reserves swap space equal to the size of the mapping; when the mapping is written into, the reserved space is employed to hold private copies of the data. A write into a **MAP_NORESERVE** mapping produces results which depend on the current availability of swap space in the system. If space is available, the write succeeds and a private copy of the written page is created; if space is not available, the write fails and a SIGBUS signal is delivered to the writing process. **MAP_NORESERVE** mappings are inherited across **fork**(2); at the time of the **fork**(2) swap space is reserved in the child for all private pages that currently exist in the parent; thereafter the child’s mapping behaves as described above.

The parameter *off* is constrained to be aligned and sized according to the value returned by **sysconf( )**. When **MAP_FIXED** is specified, the parameter *addr* must also meet these constraints. The system performs mapping operations over whole pages. Thus, while the parameter *len* need not meet a size or alignment constraint, the system will include, in any mapping operation, any partial page specified by the range [*pa*, *pa* + *len*).

The system will always zero-fill any partial page at the end of an object. Further, the system will never write out any modified portions of the last page of an object which are beyond its end. References to whole pages following the end of an object will result in the delivery of a **SIGBUS** signal. **SIGBUS** signals may also be delivered on various file system conditions, including quota exceeded errors.

If the process calls **mlockall**(3C) with the **MCL_FUTURE** flag, the pages mapped by all future calls to **mmap( )** will be locked in memory. In this case, if not enough memory could be locked, **mmap( )** fails and sets **errno** to EAGAIN.

RETURN VALUES

On success, **mmap( )** returns the address at which the mapping was placed (*pa*). On failure it returns **MAP_FAILED** and sets **errno** to indicate an error.

ERRORS

Under the following conditions, **mmap( )** fails and sets **errno** to:

EACCES *fildes* is not open for read, regardless of the protection specified, or *fildes* is not open for write and **PROT_WRITE** was specified for a **MAP_SHARED** type mapping.

EAGAIN The mapping could not be locked in memory.

There was insufficient room to reserve swap space for the mapping.

The file to be mapped is already locked using advisory or mandatory record locking. See **fcntl**(2).

EBADF *fildes* is not open.

EINVAL The arguments *addr* (if **MAP_FIXED** was specified) or *off* are not multiples of the page size as returned by **sysconf( )**.

The field in *flags* is invalid (neither **MAP_PRIVATE** or **MAP_SHARED**).

The argument *len* has a value less than or equal to **0**.

ENODEV *fildes* refers to an object for which **mmap( )** is meaningless, such as a terminal.

ENOMEM **MAP_FIXED** was specified and the range [*addr*, *addr* + *len*) exceeds that allowed for the address space of a process.

MAP_FIXED was "not" specified and there is insufficient room in the address space to effect the mapping.

The composite size of *len* plus the lengths of all previous mmappings exceeds **RLIMIT_VMEM** (see **getrlimit**(2)).

ENXIO The range [*off*, *off* + *len*) is illegal for mmapping to this device.

SEE ALSO

close(2), **fcntl**(2), **fork**(2), **getrlimit**(2), **mprotect**(2), **munmap**(2), **lockf**(3C), **mlockall**(3C), **plock**(3C), **sysconf**(3C)

NOTES

mmap( ) allows access to resources using address space manipulations instead of the **read/write** interface. Once a file is mapped, all a process has to do to access it is use the data at the address to which the object was mapped. Consider the following pseudo-code:

```
fildes = open( . . . )
lseek(fildes, offset)
read(fildes, buf, len)
/* use data in buf */
```

Here is a rewrite using **mmap( )**:

```
fildes = open( . . . )
address = mmap((caddr_t) 0, len, (PROT_READ | PROT_WRITE),
        MAP_PRIVATE, fildes, offset)
/* use data at address */
```