NAME

mmap - map pages of memory

SYNOPSIS

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

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

DESCRIPTION

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

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

The mmap() function establishes a mapping between the address space of the process 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 a function of the addr argument and values of flags, further described below. A successful mmap() call returns pa as its result. The address range starting at pa and continuing for len bytes will be legitimate for the possible (not necessarily current) address space of the process. The range of bytes starting at off and continuing for len bytes will be legitimate for the possible (not necessarily current) offsets in the file or shared memory object represented by fildes.

The mmap() function 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 is created prior to the mmap() call and has no contents, or when the file is truncated. Any reference to addresses beyond the end of the object, however, will result in the delivery of a SIGBUS or SIGSEGV signal. The mmap() function cannot be used to implicitly extend the length of files.

The mapping established by mmap() replaces any previous mappings for those whole pages containing any part of the address space of the process starting at pa and continuing for len bytes.

If the size of the mapped file changes after the call to mmap() as a result of some other operation on the mapped file, the effect of references to portions of the mapped region that correspond to added or removed portions of the file is unspecified.

The mmap() function is supported for regular files and shared memory objects. Support for any other type of file is unspecified.

The prot argument determines whether read, write, execute, or some combination of accesses are permitted to the data being mapped. The prot argument should be either PROT_NONE or the bitwise inclusive OR of one or more of the other flags in the following table, defined in the header <sys/mman.h>.

PROT_READ
Data can be read.

PROT_WRITE
Data can be written.

PROT_EXEC
Data can be executed.

PROT_NONE
Data cannot be accessed.

If an implementation of mmap() for a specific platform cannot support the combination of access types specified by prot, the call to mmap() fails. An implementation may permit accesses other than those specified by prot; however, the

implementation will not permit a write to succeed where PROT_WRITE has not been set or permit any access where PROT_NONE alone has been set. Each platform-specific implementation of mmap() supports the following values of prot: PROT_NONE, PROT_READ, PROT_WRITE, and the inclusive OR of PROT_READ and PROT_WRITE. On some platforms, the PROT_WRITE protection option is implemented as PROT_READ|PROT_WRITE and PROT_EXEC as PROT_READ|PROT_EXEC. The file descriptor fildes is opened with read permission, regardless of the protection options specified. If PROT_WRITE is specified, the application must have opened the file descriptor fildes with write permission unless MAP_PRIVATE is specified in the flags argument as described below.

The flags argument provides other information about the handling of the mapped data. The value of flags is the bitwise inclusive OR of these options, defined in <sys/mman.h>:

MAP_SHARED
Changes are shared.

MAP_PRIVATE
Changes are private.

MAP_FIXED
Interpret addr exactly.

MAP_NORESERVE
Do not reserve swap space.

MAP_ANON
Map anonymous memory.

MAP_ALIGN
Interpret addr as required aligment.

The MAP_SHARED and MAP_PRIVATE options describe the disposition of write references to the underlying 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. 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. Either MAP_SHARED or MAP_PRIVATE must be specified, but not both. The mapping type is retained across fork(2).

When MAP_FIXED is set in the flags argument, the system is informed that the value of pa must be addr, exactly. If MAP_FIXED is set, mmap() may return (void *)-1 and set errno to EINVAL. If a MAP_FIXED request is successful, the mapping established by mmap() replaces any previous mappings for the process’s pages in the range [pa, pa + len). The use of MAP_FIXED is discouraged, since it may prevent a system from making the most effective use of its resources.

When MAP_FIXED is set and the requested address is the same as previous mapping, the previous address is unmapped and the new mapping is created on top of the old one.

When MAP_FIXED is not set, the system uses addr to arrive at pa. The pa so chosen will be an area of the address space that the system deems suitable for a mapping of len bytes to the file. The mmap() function interprets an addr value of 0 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".

When MAP_ALIGN is set, the system is informed that the alignment of pa must be the same as addr. The alignment value in addr must be 0 or some power of two multiple of page size as returned by sysconf(3C). If addr is 0, the system will choose a suitable alignment.

The MAP_NORESERVE option specifies that no swap space be reserved for a mapping. Without this flag, the creation of a writable 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 or SIGSEGV signal is delivered to the writing process. MAP_NORESERVE mappings are inherited across fork(); at the time of the fork(), 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.

When MAP_ANON is set in flags, and fildes is set to -1, mmap() provides a direct path to return anonymous pages to the caller. This operation is equivalent to passing mmap() an open file descriptor on /dev/zero with MAP_ANON elided from the flags argument.

The off argument is constrained to be aligned and sized according to the value returned by sysconf(3C) when passed _SC_PAGESIZE or _SC_PAGE_SIZE. When MAP_FIXED is specified, the addr argument must also meet these constraints. The system performs mapping operations over whole pages. Thus, while the len argument 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 or SIGSEGV signal. SIGBUS signals may also be delivered on various file system conditions, including quota exceeded errors.

The mmap() function adds an extra reference to the file associated with the file descriptor fildes which is not removed by a subsequent close(2) on that file descriptor. This reference is removed when there are no more mappings to the file by a call to the munmap(2) function.

The st_atime field of the mapped file may be marked for update at any time between the mmap() call and the corresponding munmap(2) call. The initial read or write reference to a mapped region will cause the file's st_atime field to be marked for update if it has not already been marked for update.

The st_ctime and st_mtime fields of a file that is mapped with MAP_SHARED and PROT_WRITE, will be marked for update at some point in the interval between a write reference to the mapped region and the next call to msync(3C) with MS_ASYNC or MS_SYNC for that portion of the file by any process. If there is no such call, these fields may be marked for update at any time after a write reference if the underlying file is modified as a result.

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

Upon successful completion, the mmap() function returns the address at which the mapping was placed (pa); otherwise, it returns a value of MAP_FAILED and sets errno to indicate the error. The symbol MAP_FAILED is defined in the header <sys/mman.h>. No successful return from mmap() will return the value MAP_FAILED.

If mmap() fails for reasons other than EBADF, EINVAL or ENOTSUP, some of the mappings in the address range starting at addr and continuing for len bytes may have been unmapped.

ERRORS

The mmap() function will fail if:

EACCES The fildes file descriptor 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.

EBADF The fildes file descriptor is not open (and MAP_ANON was not specified).

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

The argument addr (if MAP_ALIGN was specified) is not 0 or some power of two multiple of page size as returned by sysconf(3C).

MAP_FIXED and MAP_ALIGN are both specified.

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

The argument len has a value equal to 0.

MAP_ANON was specified, but the file descriptor was not -1.

EMFILE The number of mapped regions would exceed an implementation-dependent limit (per process or per system).

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

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

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

The mapping could not be locked in memory, if required by mlockall(3C), because it would require more space than the system is able to supply.

The composite size of len plus the lengths obtained from all previous calls to mmap() exceeds RLIMIT_VMEM (see getrlimit(2)).

ENOTSUP The system does not support the combination of accesses requested in the prot argument.

ENXIO Addresses in the range [off, off + len) are invalid for the object specified by fildes.

The MAP_FIXED option was specified in flags and the combination of addr, len and off is invalid for the object specified by fildes.

EOVERFLOW
The file is a regular file and the value of off plus len exceeds the offset maximum establish in the open file description associated with fildes.

The mmap() function may fail if:

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

USAGE

Use of mmap() may reduce the amount of memory available to other memory allocation functions.

MAP_ALIGN is useful to assure a properly aligned value of pa for subsequent use with memcntl(2) and the MC_HAT_ADVISE command.
This is best used for large, long-lived, and heavily referenced regions. MAP_FIXED and MAP_ALIGN are always mutually-exclusive.

Use of MAP_FIXED may result in unspecified behavior in further use of brk(2), sbrk(2), malloc(3C), and shmat(2). The use of MAP_FIXED is discouraged, as it may prevent an implementation from making the most effective use of resources.

The application must ensure correct synchronization when using mmap() in conjunction with any other file access method, such as read(2) and write(2), standard input/output, and shmat(2).

The mmap() function has a transitional interface for 64-bit file offsets. See lf64(5).

The mmap() function 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, whence)
read(fildes, buf, len)
/* use data in buf */
```

The following 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 */
```

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Standard
MT-Level	Async-Signal-Safe

SEE ALSO

close(2), exec(2), fcntl(2), fork(2), getrlimit(2), memcntl(2), mprotect(2), munmap(2), shmat(2), lockf(3C), mlockall(3C), msync(3C), plock(3C), sysconf(3C), attributes(5), lf64(5), standards(5), null(7D), zero(7D)

SunOS 5.9 Last change: 10 Apr 2002