NAME

mprotect - set protection on a region of memory

SYNOPSIS

#include <sys/mman.h>

int mprotect(void ***addr*, **size_t *len*, **int** *prot***);**

DESCRIPTION

mprotect() changes protection for the calling process's memory page(s) containing any part of the address range in the interval [*addr*, *addr*+*len*-1]. *addr* must be aligned to a page boundary.

If the calling process tries to access memory in a manner that violates the protection, then the kernel generates a **SIGSEGV** signal for the process.

prot is either **PROT_NONE** or a bitwise-or of the other values in the following list:

PROT_NONE The memory cannot be accessed at all.

PROT_READ The memory can be read.

PROT_WRITE
The memory can be modified.

PROT_EXEC The memory can be executed.

RETURN VALUE

On success, **mprotect**() returns zero. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EACCES
The memory cannot be given the specified access. This can happen, for example, if you mmap(2) a file to which you have read-only access, then ask **mprotect**() to mark it **PROT_WRITE**.

EINVAL
addr is not a valid pointer, or not a multiple of the system page size.

ENOMEM
Internal kernel structures could not be allocated.

ENOMEM
Addresses in the range [*addr*, *addr*+*len*-1] are invalid for the address space of the process, or specify one or more pages that are not mapped. (Before kernel 2.4.19, the error **EFAULT** was incorrectly produced for these cases.)

CONFORMING TO

SVr4, POSIX.1-2001. POSIX says that the behavior of **mprotect**() is unspecified if it is applied to a region of memory that was not obtained via mmap(2).

NOTES

On Linux it is always permissible to call **mprotect**() on any address in a process's address space (except for the kernel vsyscall area). In particular it can be used to change existing code mappings to be writable.

Whether **PROT_EXEC** has any effect different from **PROT_READ** is architecture- and kernel version-dependent. On some hardware architectures (e.g., i386), **PROT_WRITE** implies **PROT_READ**.

POSIX.1-2001 says that an implementation may permit access other than that specified in *prot*, but at a minimum can allow write access only if **PROT_WRITE** has been set, and must not allow any access if **PROT_NONE** has been set.

EXAMPLE

The program below allocates four pages of memory, makes the third of these pages read-only, and then executes a loop that walks upward through the allocated region modifying bytes.

An example of what we might see when running the program is the following:

```
$ ./a.out
Start of region: 0x804c000
Got SIGSEGV at address: 0x804e000
```

Program source

```
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/mman.h>

#define handle_error(msg)
do { perror(msg); exit(EXIT_FAILURE); } while (0)

static char *buffer;

static void
handler(int sig, siginfo_t *si, void *unused)
{
printf(Got SIGSEGV at address: 0x%lxn,
(long) si->si_addr);
exit(EXIT_FAILURE);
}

int
main(int argc, char *argv[])
{
char *p;
int pagesize;
struct sigaction sa;

sa.sa_flags = SA_SIGINFO;
sigemptyset(&sa.sa_mask);
sa.sa_sigaction = handler;
if (sigaction(SIGSEGV, &sa, NULL) == -1)
handle_error(sigaction);

pagesize = sysconf(_SC_PAGE_SIZE);
if (pagesize == -1)
handle_error(sysconf);

/* Allocate a buffer aligned on a page boundary;
initial protection is PROT_READ | PROT_WRITE */

buffer = memalign(pagesize, 4 * pagesize);
if (buffer == NULL)
handle_error(memalign);

printf(Start of region: 0x%lxn, (long) buffer);

if (mprotect(buffer + pagesize * 2, pagesize,
PROT_READ) == -1)
handle_error(mprotect);

for (p = buffer ; ; )
*(p++) = a;

printf(Loop completedn); /* Should never happen */
exit(EXIT_SUCCESS);
}
```

SEE ALSO

mmap(2), sysconf(3)

COLOPHON

This page is part of release 3.74 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at http://www.kernel.org/doc/man-pages/.