NAME

mremap - remap a virtual memory address

SYNOPSIS

#define _GNU_SOURCE /* See feature_test_macros(7) */
#include <sys/mman.h>

void *mremap(void * *old_address*, **size_t** *old_size*,
size_t *new_size*, **int** *flags*, **... /* void *** *new_address* ***/);**

DESCRIPTION

mremap() expands (or shrinks) an existing memory mapping, potentially moving it at the same time (controlled by the *flags* argument and the available virtual address space).

old_address is the old address of the virtual memory block that you want to expand (or shrink). Note that *old_address* has to be page aligned. *old_size* is the old size of the virtual memory block. *new_size* is the requested size of the virtual memory block after the resize. An optional fifth argument, *new_address*, may be provided; see the description of **MREMAP_FIXED** below.

In Linux the memory is divided into pages. A user process has (one or) several linear virtual memory segments. Each virtual memory segment has one or more mappings to real memory pages (in the page table). Each virtual memory segment has its own protection (access rights), which may cause a segmentation violation if the memory is accessed incorrectly (e.g., writing to a read-only segment). Accessing virtual memory outside of the segments will also cause a segmentation violation.

mremap() uses the Linux page table scheme. **mremap**() changes the mapping between virtual addresses and memory pages. This can be used to implement a very efficient realloc(3).

The *flags* bit-mask argument may be 0, or include the following flag:

MREMAP_MAYMOVE
By default, if there is not sufficient space to expand a mapping at its current location, then **mremap**() fails. If this flag is specified, then the kernel is permitted to relocate the mapping to a new virtual address, if necessary. If the mapping is relocated, then absolute pointers into the old mapping location become invalid (offsets relative to the starting address of the mapping should be employed).

MREMAP_FIXED (since Linux 2.3.31)
This flag serves a similar purpose to the **MAP_FIXED** flag of mmap(2). If this flag is specified, then **mremap**() accepts a fifth argument, *void *new_address*, which specifies a page-aligned address to which the mapping must be moved. Any previous mapping at the address range specified by *new_address* and *new_size* is unmapped. If **MREMAP_FIXED** is specified, then **MREMAP_MAYMOVE** must also be specified.

If the memory segment specified by *old_address* and *old_size* is locked (using mlock(2) or similar), then this lock is maintained when the segment is resized and/or relocated. As a consequence, the amount of memory locked by the process may change.

RETURN VALUE

On success **mremap**() returns a pointer to the new virtual memory area. On error, the value **MAP_FAILED** (that is, *(void *) -1*) is returned, and *errno* is set appropriately.

ERRORS

EAGAIN
The caller tried to expand a memory segment that is locked, but this was not possible without exceeding the **RLIMIT_MEMLOCK** resource limit.

EFAULT
Segmentation fault. Some address in the range *old_address* to *old_address*+*old_size* is an invalid virtual memory address for this process. You can also get **EFAULT** even if there exist mappings that cover the whole address space requested, but those mappings are of different types.

EINVAL
An invalid argument was given. Possible causes are: *old_address* was not page aligned; a value other than **MREMAP_MAYMOVE** or **MREMAP_FIXED** was specified in *flags*; *new_size* was zero; *new_size* or *new_address* was invalid; or the new address range specified by *new_address* and *new_size* overlapped the old address range specified by *old_address* and *old_size*; or **MREMAP_FIXED** was specified without also specifying **MREMAP_MAYMOVE**.

ENOMEM
The memory area cannot be expanded at the current virtual address, and the **MREMAP_MAYMOVE** flag is not set in *flags*. Or, there is not enough (virtual) memory available.

CONFORMING TO
This call is Linux-specific, and should not be used in programs intended to be portable.

NOTES
Prior to version 2.4, glibc did not expose the definition of **MREMAP_FIXED**, and the prototype for **mremap**() did not allow for the *new_address* argument.

SEE ALSO
brk(2), getpagesize(2), getrlimit(2), mlock(2), mmap(2), sbrk(2), malloc(3), realloc(3)

Your favorite text book on operating systems for more information on paged memory (e.g., *Modern Operating Systems*b y Andrew S. Tanenbaum, *Inside Linux* by Randolf Bentson, *The Design of the UNIX Operating System*b y Maurice J. Bach)

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/.