NAME

mlock, **munlock** - lock (unlock) physical pages in memory

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <sys/mman.h>

int
mlock(*const void *addr*, *size_t len*);

int
munlock(*const void *addr*, *size_t len*);

DESCRIPTION

The **mlock**() system call locks into memory the physical pages associated with the virtual address range starting at *addr* for *len* bytes. The **munlock**() system call unlocks pages previously locked by one or more **mlock**() calls. For both, the *addr* argument should be aligned to a multiple of the page size. If the *len* argument is not a multiple of the page size, it will be rounded up to be so. The entire range must be allocated.

After an **mlock**() system call, the indicated pages will cause neither a non-resident page nor address-translation fault until they are unlocked. They may still cause protection-violation faults or TLB-miss faults on architectures with software-managed TLBs. The physical pages remain in memory until all locked mappings for the pages are removed. Multiple processes may have the same physical pages locked via their own virtual address mappings. A single process may likewise have pages multiply-locked via different virtual mappings of the same physical pages. Unlocking is performed explicitly by **munlock**() or implicitly by a call to **munmap**() which deallocates the unmapped address range. Locked mappings are not inherited by the child process after a fork(2).

Since physical memory is a potentially scarce resource, processes are limited in how much they can lock down. The amount of memory that a single process can **mlock**() is limited by both the per-process RLIMIT_MEMLOCK resource limit and the system-wide "wired pages" limit *vm.max_user_wired*. *vm.max_user_wired* applies to the system as a whole, so the amount available to a single process at any given time is the difference between *vm.max_user_wired* and *vm.stats.vm.v_user_wire_count*.

If *security.bsd.unprivileged_mlock* is set to 0 these calls are only available to the super-user.

RETURN VALUES

Upon successful completion, the value 0 is returned; otherwise the value -1 is returned and the global variable *errno* is set to indicate the error.

If the call succeeds, all pages in the range become locked (unlocked); otherwise the locked status of all pages in the range remains unchanged.

ERRORS

The **mlock**() system call will fail if:

[EPERM] *security.bsd.unprivileged_mlock* is set to 0 and the caller is not the super-user.

[EINVAL] The address range given wraps around zero.

[ENOMEM] Some portion of the indicated address range is not allocated. There was an error faulting/mapping a page. Locking the indicated range would exceed the per-process or system-wide limits for locked memory.

The **munlock**() system call will fail if:

[EPERM] *security.bsd.unprivileged_mlock* is set to 0 and the caller is not the super-user.

[EINVAL] The address range given wraps around zero.

[ENOMEM] Some or all of the address range specified by the addr and len arguments does not correspond to valid mapped pages in the address space of the process.

[ENOMEM] Locking the pages mapped by the specified range would exceed a limit on the amount of memory that the process may lock.

SEE ALSO

fork(2), mincore(2), minherit(2), mlockall(2), mmap(2), munlockall(2), munmap(2), setrlimit(2), getpagesize(3)

HISTORY

The **mlock**() and **munlock**() system calls first appeared in 4.4BSD.

BUGS

Allocating too much wired memory can lead to a memory-allocation deadlock which requires a reboot to recover from.

The per-process and system-wide resource limits of locked memory apply to the amount of virtual

memory locked, not the amount of locked physical pages. Hence two distinct locked mappings of the same physical page counts as 2 pages aginst the system limit, and also against the per-process limit if both mappings belong to the same physical map.

The per-process resource limit is not currently supported.