NAME

modify_ldt − get or set a per-process LDT entry

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

#include <asm/ldt.h> /* Definition of **struct user_desc** */
#include <sys/syscall.h> /* Definition of **SYS_*** constants */
#include <unistd.h>

int syscall(SYS_modify_ldt, int *func***, v oid** *ptr***[.***bytecount***],**
unsigned long *bytecount***);**

Note: glibc provides no wrapper for **modify_ldt**(), necessitating the use of **syscall**(2).

DESCRIPTION

modify_ldt() reads or writes the local descriptor table (LDT) for a process. The LDT is an array of segment descriptors that can be referenced by user code. Linux allows processes to configure a per-process (actually per-mm) LDT. For more information about the LDT, see the Intel Software Developer's Manual or the AMD Architecture Programming Manual.

When *func* is 0, **modify_ldt**() reads the LDT into the memory pointed to by *ptr*. The number of bytes read is the smaller of *bytecount* and the actual size of the LDT, although the kernel may act as though the LDT is padded with additional trailing zero bytes. On success, **modify_ldt**() will return the number of bytes read.

When *func* is 1 or 0x11, **modify_ldt**() modifies the LDT entry indicated by *ptr−>entry_number*. *ptr* points to a *user_desc* structure and *bytecount* must equal the size of this structure.

The *user_desc* structure is defined in *<asm/ldt.h>* as:

```
struct user_desc {
    unsigned int  entry_number;
    unsigned int  base_addr;
    unsigned int  limit;
    unsigned int  seg_32bit:1;
    unsigned int  contents:2;
    unsigned int  read_exec_only:1;
    unsigned int  limit_in_pages:1;
    unsigned int  seg_not_present:1;
    unsigned int  useable:1;
};
```

In Linux 2.4 and earlier, this structure was named *modify_ldt_ldt_s*.

The *contents* field is the segment type (data, expand-down data, non-conforming code, or conforming code). The other fields match their descriptions in the CPU manual, although**modify_ldt**() cannot set the hardware-defined "accessed" bit described in the CPU manual.

A *user_desc* is considered "empty" if *read_exec_only* and *seg_not_present* are set to 1 and all of the other fields are 0. An LDT entry can be cleared by setting it to an "empty" *user_desc* or, if *func* is 1, by setting both *base* and *limit* to 0.

A conforming code segment (i.e., one with *contents==3*) will be rejected if *func* is 1 or if *seg_not_present* is 0.

When *func* is 2, **modify_ldt**() will read zeros. This appears to be a leftover from Linux 2.4.

RETURN VALUE

On success, **modify_ldt**() returns either the actual number of bytes read (for reading) or 0 (for writing). On failure, **modify_ldt**() returns −1 and sets *errno* to indicate the error.

ERRORS

EFAULT

ptr points outside the address space.

EINVAL

ptr is 0, or *func* is 1 and *bytecount* is not equal to the size of the structure *user_desc*, or *func* is 1 or 0x11 and the new LDT entry has invalid values.

ENOSYS

func is neither 0, 1, 2, nor 0x11.

STANDARDS

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

NOTES

modify_ldt() should not be used for thread-local storage, as it slows down context switches and only supports a limited number of threads. Threading libraries should use **set_thread_area**(2) or **arch_prctl**(2) instead, except on extremely old kernels that do not support those system calls.

The normal use for **modify_ldt**() is to run legacy 16-bit or segmented 32-bit code. Not all kernels allow 16-bit segments to be installed, however.

Even on 64-bit kernels, **modify_ldt**() cannot be used to create a long mode (i.e., 64-bit) code segment. The undocumented field "lm" in *user_desc* is not useful, and, despite its name, does not result in a long mode segment.

BUGS

On 64-bit kernels before Linux 3.19, setting the "lm" bit in *user_desc* prevents the descriptor from being considered empty. Keep in mind that the "lm" bit does not exist in the 32-bit headers, but these buggy kernels will still notice the bit even when set in a 32-bit process.

SEE ALSO

arch_prctl(2), **set_thread_area**(2), **vm86**(2)