



Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'get_thread_area.2'

\$ man get_thread_area.2

set_thread_area(2) System Calls Manual set_thread_area(2)

NAME

get_thread_area, set_thread_area - manipulate thread-local storage information

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <sys/syscall.h>    /* Definition of SYS_* constants */
#include <unistd.h>

#if defined __i386__ || defined __x86_64__

#include <asm/ldt.h>    /* Definition of struct user_desc */

int syscall(SYS_get_thread_area, struct user_desc *u_info);
int syscall(SYS_set_thread_area, struct user_desc *u_info);

#elif defined __m68k__

int syscall(SYS_get_thread_area);
int syscall(SYS_set_thread_area, unsigned long tp);

#elif defined __mips__ || defined __csky__

int syscall(SYS_set_thread_area, unsigned long addr);

#endif
```

Note: glibc provides no wrappers for these system calls, necessitating the use of
syscall(2).

DESCRIPTION

These calls provide architecture-specific support for a thread-local storage implementation.

At the moment, set_thread_area() is available on m68k, MIPS, C-SKY, and x86 (both 32-bit and

64-bit variants); `get_thread_area()` is available on m68k and x86.

On m68k, MIPS and C-SKY, `set_thread_area()` allows storing an arbitrary pointer (provided in the `tp` argument on m68k and in the `addr` argument on MIPS and C-SKY) in the kernel data structure associated with the calling thread; this pointer can later be retrieved using `get_thread_area()` (see also NOTES for information regarding obtaining the thread pointer on MIPS).

On x86, Linux dedicates three global descriptor table (GDT) entries for thread-local storage. For more information about the GDT, see the Intel Software Developer's Manual or the AMD Architecture Programming Manual.

Both of these system calls take an argument that is a pointer to a structure of the following type:

```
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;
#ifdef __x86_64__
    unsigned int  lm:1;
#endif
};
```

`get_thread_area()` reads the GDT entry indicated by `u_info->entry_number` and fills in the rest of the fields in `u_info`.

`set_thread_area()` sets a TLS entry in the GDT.

The TLS array entry set by `set_thread_area()` corresponds to the value of `u_info->entry_number` passed in by the user. If this value is in bounds, `set_thread_area()` writes the descriptor pointed to by `u_info` into the thread's TLS array.

When `set_thread_area()` is passed an `entry_number` of -1, it searches for a free TLS entry.

If `set_thread_area()` finds a free TLS entry, the value of `u_info->entry_number` is set upon

return to show which entry was changed.

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. If an "empty" descriptor is passed to `set_thread_area()`, the corresponding TLS entry will be cleared. See BUGS for additional details.

Since Linux 3.19, `set_thread_area()` cannot be used to write non-present segments, 16-bit segments, or code segments, although clearing a segment is still acceptable.

RETURN VALUE

On x86, these system calls return 0 on success, and -1 on failure, with `errno` set to indicate the error.

On C-SKY, MIPS and m68k, `set_thread_area()` always returns 0. On m68k, `get_thread_area()` returns the thread area pointer value (previously set via `set_thread_area()`).

ERRORS

`EFAULT` `u_info` is an invalid pointer.

`EINVAL` `u_info->entry_number` is out of bounds.

`ENOSYS` `get_thread_area()` or `set_thread_area()` was invoked as a 64-bit system call.

`ESRCH` (`set_thread_area()`) A free TLS entry could not be located.

STANDARDS

Linux.

HISTORY

`set_thread_area()`

Linux 2.5.29.

`get_thread_area()`

Linux 2.5.32.

NOTES

These system calls are generally intended for use only by threading libraries.

`arch_prctl(2)` can interfere with `set_thread_area()` on x86. See `arch_prctl(2)` for details. This is not normally a problem, as `arch_prctl(2)` is normally used only by 64-bit programs.

On MIPS, the current value of the thread area pointer can be obtained using the instruction:

```
rdhwr dest, $29
```

This instruction traps and is handled by kernel.

BUGS

On 64-bit kernels before Linux 3.19, one of the padding bits in `user_desc`, if set, would

prevent the descriptor from being considered empty (see `modify_ldt(2)`). As a result, the only reliable way to clear a TLS entry is to use `memset(3)` to zero the entire `user_desc` structure, including padding bits, and then to set the `read_exec_only` and `seg_not_present` bits. On Linux 3.19, a `user_desc` consisting entirely of zeros except for `entry_number` will also be interpreted as a request to clear a TLS entry, but this behaved differently on older kernels.

Prior to Linux 3.19, the DS and ES segment registers must not reference TLS entries.

SEE ALSO

`arch_prctl(2)`, `modify_ldt(2)`, `ptrace(2)` (`PTRACE_GET_THREAD_AREA` and `PTRACE_SET_THREAD_AREA`)

Linux man-pages 6.17

2026-02-08

`set_thread_area(2)`