



## ***Rocky Enterprise Linux 9.2 Manual Pages on command 'set\_robust\_list.2'***

**C:\>man set\_robust\_list.2**

GET\_ROBUST\_LIST(2)                  Linux System Calls                  GET\_ROBUST\_LIST(2)

### NAME

get\_robust\_list, set\_robust\_list - get/set list of robust futexes

### SYNOPSIS

```
#include <linux/futex.h>
```

```
#include <sys/types.h>
```

```
#include <syscall.h>
```

```
long get_robust_list(int pid, struct robust_list_head **head_ptr,  
                    size_t *len_ptr);
```

```
long set_robust_list(struct robust_list_head *head, size_t len);
```

Note: There are no glibc wrappers for these system calls; see NOTES.

### DESCRIPTION

These system calls deal with per-thread robust futex lists. These lists are managed in user space: the kernel knows only about the location of the head of the list. A thread can inform the kernel of the location of its robust futex list using `set_robust_list()`. The address of a thread's robust futex list can be obtained using `get_robust_list()`.

The purpose of the robust futex list is to ensure that if a thread accidentally fails to unlock a futex before terminating or calling `execve(2)`, another thread that is waiting on that futex is notified that the former owner of the futex has died. This notification consists of two pieces: the `FUTEX_OWNER_DIED` bit is set in the futex word, and the kernel performs a `futex(2) FUTEX_WAKE` operation on one of

the threads waiting on the futex.

The `get_robust_list()` system call returns the head of the robust futex list of the thread whose thread ID is specified in `pid`. If `pid` is 0, the head of the list for the calling thread is returned. The list head is stored in the location pointed to by `head_ptr`. The size of the object pointed to by `**head_ptr` is stored in `len_ptr`. Permission to employ `get_robust_list()` is governed by a `ptrace` access mode `PTRACE_MODE_READ_REALCREDS` check; see `ptrace(2)`.

The `set_robust_list()` system call requests the kernel to record the head of the list of robust futexes owned by the calling thread. The `head` argument is the list head to record. The `len` argument should be `sizeof(*head)`.

## RETURN VALUE

The `set_robust_list()` and `get_robust_list()` system calls return zero when the operation is successful, an error code otherwise.

## ERRORS

The `set_robust_list()` system call can fail with the following error:

`EINVAL` `len` does not equal `sizeof(struct robust_list_head)`.

The `get_robust_list()` system call can fail with the following errors:

`EFAULT` The head of the robust futex list can't be stored at the location `head`.

`EPERM` The calling process does not have permission to see the robust futex list of the thread with the thread ID `pid`, and does not have the `CAP_SYS_PTRACE` capability.

`ESRCH` No thread with the thread ID `pid` could be found.

## VERSIONS

These system calls were added in Linux 2.6.17.

## NOTES

These system calls are not needed by normal applications. No support for them is provided in `glibc`. In the unlikely event that you want to call them directly, use `syscall(2)`.

A thread can have only one robust futex list; therefore applications that wish to use this functionality should use the robust mutexes provided by `glibc`.

In the initial implementation, a thread waiting on a futex was notified that the owner had died only if the owner terminated. Starting with Linux 2.6.28, notification was extended to include the case where the owner performs an `execve(2)`.

The thread IDs mentioned in the main text are kernel thread IDs of the kind returned by `clone(2)` and `gettid(2)`.

#### SEE ALSO

`futex(2)`, `pthread_mutexattr_setrobust(3)`

`Documentation/robust-futexes.txt` and `Documentation/robust-futex-ABI.txt` in the Linux kernel source tree

#### COLOPHON

This page is part of release 5.05 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 <https://www.kernel.org/doc/man-pages/>.

Linux

2019-10-10

GET\_ROBUST\_LIST(2)