NAME

futex − fast user-space locking

SYNOPSIS

#include <linux/futex.h>

DESCRIPTION

The Linux kernel provides futexes ("Fast user-space mutexes") as a building block for fast user-space locking and semaphores. Futexes are very basic and lend themselves well for building higher-level locking abstractions such as mutexes, condition variables, read-write locks, barriers, and semaphores.

Most programmers will in fact not be using futexes directly but will instead rely on system libraries built on them, such as the Native POSIX Thread Library (NPTL) (see **pthreads**(7)).

A futex is identified by a piece of memory which can be shared between processes or threads. In these different processes, the futex need not have identical addresses. In its bare form, a futex has semaphore semantics; it is a counter that can be incremented and decremented atomically; processes can wait for the value to become positive.

Futex operation occurs entirely in user space for the noncontended case. The kernel is involved only to arbitrate the contended case. As any sane design will strive for noncontention, futexes are also optimized for this situation.

In its bare form, a futex is an aligned integer which is touched only by atomic assembler instructions. This integer is four bytes long on all platforms. Processes can share this integer using **mmap**(2), via shared memory segments, or because they share memory space, in which case the application is commonly called multithreaded.

Semantics

Any futex operation starts in user space, but it may be necessary to communicate with the kernel using the **futex**(2) system call.

To "up" a futex, execute the proper assembler instructions that will cause the host CPU to atomically increment the integer. Afterward, check if it has in fact changed from 0 to 1, in which case there were no waiters and the operation is done. This is the noncontended case which is fast and should be common.

In the contended case, the atomic increment changed the counter from −1 (or some other negative number). If this is detected, there are waiters. User space should now set the counter to 1 and instruct the kernel to wake up any waiters using the **FUTEX_WAKE** operation.

Waiting on a futex, to "down" it, is the reverse operation. Atomically decrement the counter and check if it changed to 0, in which case the operation is done and the futex was uncontended. In all other circumstances, the process should set the counter to −1 and request that the kernel wait for another process to up the futex. This is done using the**FUTEX_WAIT** operation.

The **futex**(2) system call can optionally be passed a timeout specifying how long the kernel should wait for the futex to be upped. In this case, semantics are more complex and the programmer is referred to **futex**(2) for more details. The same holds for asynchronous futex waiting.

VERSIONS

Initial futex support was merged in Linux 2.5.7 but with different semantics from those described above. Current semantics are available from Linux 2.5.40 onward.

NOTES

To reiterate, bare futexes are not intended as an easy-to-use abstraction for end users. Implementors are expected to be assembly literate and to have read the sources of the futex user-space library referenced below.

This man page illustrates the most common use of the **futex**(2) primitives; it is by no means the only one.

SEE ALSO

clone(2), **futex**(2), **get_robust_list**(2), **set_robust_list**(2), **set_tid_address**(2), **pthreads**(7)

Fuss, Futexes and Furwocks: Fast Userlevel Locking in Linux (proceedings of the Ottawa Linux Symposium 2002), futex example library, futex-*.tar.bz2 ⟨https://mirrors.kernel.org/pub/linux/kernel/people

/rusty/⟩.