



Software Lab

Advanced IPC

Roberto Farina
roberto.farina@cefriel.it

Summary



- Futex
 - ▶ General concepts
 - ▶ Futex in practice
- Barriers
 - ▶ General concepts
 - ▶ How they can be implemented

Futex



- **F**ast **U**terspace **muTEx**: a basic tool to realize locking and building higher-level locking
 - ▶ First appearance in the development kernel version 2.5.7
- A piece of memory (an aligned integer) that can be shared between processes
 - ▶ It can be incremented and decremented by atomic instructions
 - ▶ Processes can wait for the value to become positive

Why a new lock mechanism



- Classic mechanisms (Sys V semaphores, fcntl lock) represent **heavy weight** kernel approaches
- Kernel based mechanisms imply syscalls
 - ▶ Significant overhead when low contention rates
- Futex operations are done almost entirely in **userspace**
 - ▶ The kernel is only involved when a contended case needs to be arbitrated
 - ▶ Locking primitives implementing used futexes to be very efficient

Goals and requirements



- Goals
 - ▶ Avoid syscalls
 - ▶ Avoid unnecessary context switches
- Requirements
 - ▶ Fairness in the locking and release policies
 - ▶ Avoid the convoy problem when using FIFO policy

Implementation aspects



- A variable of type `int` at the user level
 - ▶ A size of 4 bytes on all platforms
- Kernel object mapped at **different virtual addresses** in different processes
 - ▶ A lookup has to be performed
- A queue for waiting threads
- A **single** multiplexed syscall
 - ▶ `long sys_futex(void* addr1, int val1, ...)`
 - ▶ Avoid problems with syscall number allocation
- Two basic operations: **FUTEX_UP** and **FUTEX_DOWN**

Recent implementation



- **FUTEX_WAIT**
 - ▶ Thread suspended in the kernel until notified
 - ▶ It is possible to specify a timeout
- **FUTEX_WAKE**
 - ▶ Wake up one or more threads waiting for the futex
 - ▶ It is possible to specify how many threads to wake up
- **FUTEX_FD**
 - ▶ The kernel generates a file descriptor to refer to the futex
 - ▶ It is possible to request asynchronous notification
 - A signal associated to the fd

Details



- Futexes are **non RT**
 - ▶ The kernel doesn't look through the queue of waiting threads to find the one with highest priority when a wake operation is requested
- **Contrast** with Sys V IPC mechanism
 - ▶ They merely export an handle to the user
 - ▶ All operations are performed in the kernel

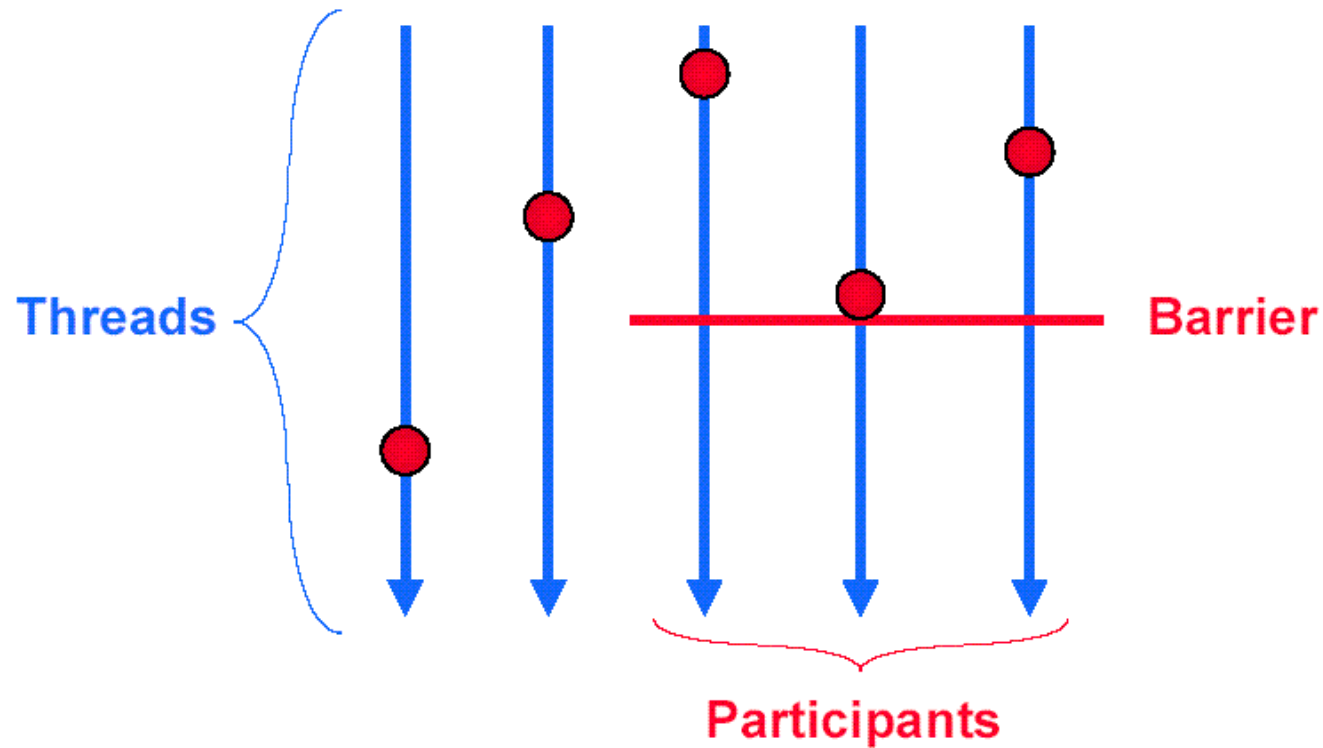
Barriers



- All processes have to reach a **specific point** before any one can proceed further
 - ▶ Rendezvous
- To ensure the computation has reached a **steady state**
- There's no data exchange or communication between processes



Synchronization Barriers



Barriers



- The number of threads synchronizing at a barrier is specified at initialization time
 - ▶ It cannot be changed at run time
- A thread calls `barrier_wait` (or `barrier_sync`) to synchronize with the barrier
 - ▶ The barrier keeps track about the number of threads that invoked this primitive
 - ▶ Returns -1 to the last thread that arrives, 0 otherwise

Bibliography



- “Fuss, Futexes and Furwocks: Fast Userlevel Locking in Linux” - Hubertus Franke, Rusty Russel, Matthew Kirkwood
- “Futexes are tricky” - Ulrich Drepper
- <ftp://ftp.kernel.org/pub/linux/kernel/people/rusty>