

Generic Mutex Subsystem of Linux Kernel

Keyan Chen(kc32), Zhiyuan Tang(zt6)

8 pages

Abstract

Sample abstract.

1. Breif History and Background

In the linux kernel, mutexes refer to a particular locking primitive that enforces serialization on shared memory systems, and not only to the generic term referring to mutual exclusion found in academia or similar theoretical textbooks.

Before 2006, when developers wanted to gain mutual exclusion among their shared memory systems, they would use binary semaphores, which are sleeping locks. However, Mutexes were introduced in 2006 as an alternative to binary semaphores and this new data structure provided a number of advantages, including simpler interfaces, and at that time smaller code.

1.1 Why Mutex?

Why do we need a new mutex subsystem? And what's wrong with semaphores?

1. 'struct mutex' is smaller.

- (a) On x86, 'struct semaphore' is 20 bytes, 'struct mutex' is 16 bytes. A smaller structure size means less RAM footprint, and better CPU-cache utilization

2. Mutex can result in tighter code

- (a) On x86 we can get the following .text sizes when switching all mutex-alike semaphores in the kernel to the mutex subsystem

text	data	bss	dec	hex	filename
3280380	868188	396860	4545428	455b94	vmlinux-semaphore
3255329	865296	396732	4517357	44eded	vmlinux-mutex

- (b) That's 25021 bytes of code saves, or a 0.76% win off the hottest codes paths of the kernel

- (c) Smaller code means better in-cache footprint, which is one of the major optimization goals in the linux kernel when people were proposing the addition of mutex subsystem in 2006.

3. The mutex subsystem is faster and has superior scalability for contented workloads.

- (a) On a 8-way x86 system, running a mutex based kernel and testing create+unlink+close(of separate, per-task files) in /tmp with 16 parallel tasks, the average number of ops/sec is

Semaphores:

```
$ ./test-mutex V 16 10
8 CPUs, running 16 tasks.
checking VFS performance.
avg loops/sec:      34713
CPU utilization:    63%
```

Mutexes:

```
$ ./test-mutex V 16 10
8 CPUs, running 16 tasks.
checking VFS performance.
avg loops/sec:      84153
CPU utilization:    22%
```

- (b) In this workload, mutex based kernel was **2.4 times** faster than the semaphore based kernel, and it also had **2.8 times** less CPU utilization.

1.2 Design and Implementation details

Mutex is represented by struct mutex, defined in include/linux/mutex.h and implemented in kernel/locking/mutex.c. The mutex uses a three state atomic counter to represent the different possible transitions that can occur during the lifetime of a mutex:

- 1: unlocked
- 0: locked, no waiters
- negative: locked, with potential waiters

In its most basic form it also includes a wait-queue and a spinlock that serializes access to it. When acquiring a mutex, there are three possible paths that can be taken, depending on the state of lock:

1. Fastpath: tries to atomically acquire the lock by decrementing the counter. If it was already taken by another task it goes to the next possible path. This logic is architecture specific.

2. Mid-path: aka optimistic spinning. It tries to spin for acquisition while the lock owner is running and there are no other tasks ready to run that have higher priority. The rationale is that if the lock owner is running, it is likely to release the lock soon. The mutex spinners are queued up using MCS lock so that only one spinner can compete for the mutex. The MCS lock is a simple spinlock with the desirable properties of being fair and with each cpu trying to acquire the lock spinning on a local variable. It avoids expensive cache-line bouncing that common test-and-set spinlock implementations incur. And MCS-like lock is specially tailored for optimistic spinning for sleeping lock implementation.
3. Slowpath: last resort, if the lock is still unable to be acquired, the task is added to the wait-queue and sleeps until woken up by the unlock path. Under normal circumstances it blocks as TASK_UNINTERRUPTIBLE.

2. Interesting Findings

2.1 Unique Optimistic Spinning in Linux Mutex Subsystem