An Overview of Locking in the FreeBSD Kernel

Brought to you by

Dr. Marshall Kirk McKusick

BSDCan Conference
May 11, 2012
University of Ottawa
Ottawa, Canada

Copyright 2012 Marshall Kirk McKusick. All Rights Reserved.
Outline

• Historic synchronization
• Lock hierarchy
• Turnstiles and sleep queues
• Details of each lock type
• Witness system
Historic Synchronization

1) Check for Resource

2) If NOT Available
   • set WANT flag
   • sleep on it

3) If IS Available
   • set LOCK flag
   • use it (while possibly sleeping)
   • clear LOCK flag
   • if WANT flag set wakeup all processes sleeping on it
Lock Hierarchy

• Hardware – memory interlock test-and-set
• Spin mutex – spin lock
• Locks that block briefly, but may not sleep
  • Blocking mutex – spin for a while, then block on a turnstile
  • Pool mutex – general-use blocking mutex
• Reader-Writer locks – mutexes with shared-exclusive semantics
• Read-mostly locks – fast access for reading
• Locks using sleep-queue interface
  • Shared-Exclusive locks – fast and simple sleep locks
• Condition variables – wrapper on traditional sleep/wakeup
• Lock manager – long-term full-function sleep lock
• Witness – partially-ordered sleep locks
Turnstiles

- Used by blocking mutexes, reader-writer, and read-mostly locks

- Designed for short periods, typically a few tens of instructions

- Used to protect read and write access to data structures and lists

- May not own a turnstile lock when requesting a sleep-queue lock

- Tracks current lock holder

- Priority propagation from waiter to holder
Turnstile Implementation

• Hash header to quickly find a lock’s turnstile. The turnstile points to the thread holding the lock and to any threads waiting for the lock.

• A turnstile is needed each time a thread blocks. Since a thread can only block on one lock at a time, it provides its own turnstile.

• Unneeded turnstiles are saved and returned when a thread awakens.

• If the holder of a lock has a lower priority than the thread about to be blocked, recursively propagate the higher priority to the holder (but only until it releases the lock).
Turnstile Data Structures

Thread 1
- owned

Thread 2
- owned

Thread 3
- owned

Thread 4
- owned

Thread 5
- owned

Thread 6
- owned

Turnstile hash header

owner

lock

waiting

Lock 18

Lock 15

Lock 6

Lock 4

owner

lock

waiting

owner

lock

waiting

owner

lock

waiting

owner

lock

waiting

owner

lock

waiting
Sleep Queues

• Used by shared-exclusive locks, condition variables, and lock-manager locks

• Designed for long periods, typically waiting for I/O events or user input

• No priority propagation

• May not own a turnstile lock when requesting a sleep-queue lock

• Tracks current exclusive lock holder

• May be recursive
Critical Sections

- Uses critical_enter() and critical_exit()

- While in a critical section:
  - The thread cannot be preempted by another thread
  - The thread cannot be migrated to another CPU

- Critical sections are much like the old single threaded kernel

- Useful for per-CPU data structures like a run-queue or CPU-specific memory allocation structures

- Cannot protect systemwide data structures
Hardware Requirements for Locking

• Minimum requirement is test-and-set instruction

• On modern hardware, FreeBSD uses compare-and-swap
  
    • Owner field for a free lock contains MTX_UNOWNED

    • Owner field for a held lock contains pointer to owning thread

    • Allocation attempt compares lock owner with MTX_UNOWNED and if it matches stores pointer to acquiring thread and returns previous owner value

    • If previous owner value was MTX_UNOWNED, acquisition succeeded

    • Store MTX_UNOWNED in owner field for lock to release it
Spin Mutex

- Exclusive access only

- Loops waiting for the mutex to become available

- Runs inside a critical section while held to avoid deadlock

- More expensive to obtain than a blocking mutex

- In FreeBSD, used only for low-level scheduling and context-switching
Blocking Mutex

- Exclusive access only

- Uses adaptive spinning which only spins if the owner of the lock is currently running
  - Current owner typically done with it quickly
  - If owner on run queue, blocking lets waiter give its CPU to owner

- All waiters are awakened when lock is released
  - Cheaper to release an uncontested lock since just a store rather than find and traverse the turnstile
  - Often end up scheduling sequentially
  - When scheduled concurrently, adaptive spinning usually ensures that they will not block
Pool Mutex

• Used for small short-lived data structures
  • Just need a pointer to a mutex rather than large mutex itself
  • Mutex is preallocated so avoid high creation and destruction times

• Example is poll system call that needs a structure to track a poll request from the time the system call is entered until the arrival of data for one of the polled descriptors.
Reader-Writer Locks

• In addition to exclusive access of a mutex also provide shared semantics

• Uses a turnstile so cannot be held when thread goes to sleep

• Provides priority propagation for exclusive access

• Does not provide priority propagation for shared access

• May specify permission to recurse
Read-Mostly Locks

• Same properties as reader-writer locks except they add priority propagation for shared access by tracking shared owners using a caller-supplied tracker data structure

• Designed for fast access for readers (shared access) assuming there will be few writers (exclusive access)
  • Read without a lock then check if write happened
  • If write happened fall back to using lock to get coherent access

• The routing table is a good example of a read-mostly data structure

• Best way to implement read-mostly locks is patented by IBM
  • IBM allows GPL’ed code to use their patented implementation at no cost
  • FreeBSD is not GPL, so we have to use a slower technique
Shared-Exclusive Locks

- Fastest and simplest of the locks that can sleep
- Provide shared and exclusive access
- May specify permission to recurse
- May request interruption by a signal
- Limited upgrade and downgrade capabilities
- Like all sleep locks, does not implement priority propagation
**Condition Variables**

- Wrapper on traditional sleep and wakeup
- Allows waiting with optional time out and/or interruption by a signal
- Allows waking up one or all waiters
- Must hold a mutex before awakening or waiting (mutex is released while waiting).
Lock Manager Locks

• Most full-featured of the locks that can sleep

• Provide shared and exclusive access

• May specify permission to recurse

• May request a time out and/or interruption by a signal

• Allows downgrade, upgrade, and exclusive upgrade

• The ability to pass ownership of the lock from a thread to the kernel

• The ability to drain all accessing threads in preparation for being deallocated

• Like all sleep locks, does not implement priority propagation
Witness

Partial ordering requires:

1) A thread may acquire only one lock in a class

2) A thread may acquire only a lock in a higher-numbered class than the highest-numbered class for which it already holds a lock

• Programmers can define lock classes
• Witness code observes actual lock ordering and complains when either rule is violated
Questions

Marshall Kirk McKusick
<mckusick@mckusick.com>
http://www.mckusick.com