POSIX Threads: a first step toward parallel programming

George Bosilca
bosilca@icl.utk.edu
Process vs. Thread

• A process is a collection of virtual memory space, code, data, and system resources.
• A thread (lightweight process) is code that is to be serially executed within a process.
• A process can have several threads.

Threads executing the same block of code maintain separate stacks. Each thread in a process shares that process's global variables and resources.

Possible to create more efficient applications?
Process vs. Thread

• Multithreaded applications must avoid two threading problems: deadlocks and races.
• A deadlock occurs when each thread is waiting for the other to do something.
• A race condition occurs when one thread finishes before another on which it depends, causing the former to use a bogus value because the latter has not yet supplied a valid one.
The key is synchronization

- Synchronization = gaining access to a shared resource.
- Synchronization REQUIRE cooperation.
POSIX Thread

• What’s POSIX?
  – Widely used UNIX specification
  – Most of the UNIX flavor operating systems

POSIX is the Portable Operating System Interface, the open operating interface standard accepted world-wide. It is produced by IEEE and recognized by ISO and ANSI.
Mutual exclusion

• Simple lock primitive with 2 states: lock and unlock
• Only one thread can lock the mutex.
• Several politics: FIFO, random, recursive
Mutual exclusion

- Simple lock primitive with 2 states: lock and unlock
- Only one thread can lock the mutex.
- Several politics: FIFO, random, recursive
Mutual exclusion

- Simple lock primitive with 2 states: lock and unlock
- Only one thread can lock the mutex.
- Several politics: FIFO, random, recursive
Mutual exclusion

- Simple lock primitive with 2 states: lock and unlock
- Only one thread can lock the mutex.
- Several politics: FIFO, random, recursive
Mutual exclusion

- Simple lock primitive with 2 states: lock and unlock
- Only one thread can lock the mutex.
- Several politics: FIFO, random, recursive

Active threads

Thread 1
... lock
... unlock
...

Thread 2
... lock
... unlock
...

Thread 3
... lock
... unlock
...

Sleeping threads

mutex
Mutual exclusion

- Simple lock primitive with 2 states: lock and unlock
- Only one thread can lock the mutex.
- Several politics: FIFO, random, recursive
Mutual exclusion

• Spin vs. sleep?
• What’s the desired lock grain?
  – Fine grain – spin mutex
  – Coarse grain – sleep mutex
• Spin mutex: use CPU cycles and increase the memory bandwidth, but when the mutex is unlock the thread continue his execution immediately.
Shared/Exclusive Locks

• **ReadWrite Mutual exclusion**
• Extension used by the reader/writer model
• 4 states: write_lock, write_unlock, read_lock and read_unlock.
• multiple threads may hold a shared lock simultaneously, but only one thread may hold an exclusive lock.
• if one thread holds an exclusive lock, no threads may hold a shared lock.
Shared/Exclusive Locks

Legend

- Active thread
- Sleeping thread

Step 1

- Writer 1: rw_lock
  - rw_unlock
  - ...
- Writer 2: rw_lock
  - rw_unlock
  - ...
- Reader 1: rd_lock
  - rd_unlock
  - ...
- Reader 2: rd_lock
  - rd_unlock
  - ...

Step 2

- Writer 1: rw_lock
  - rw_unlock
  - ...
- Writer 2: rw_lock
  - rw_unlock
  - ...
- Reader 1: rd_lock
  - rd_unlock
  - ...
- Reader 2: rd_lock
  - rd_unlock
  - ...
Shared/Exclusive Locks

Legend

Active thread

Sleeping thread

Writer 1

... rw_lock...

rw_unlock...

Writer 2

... rw_lock...

rw_unlock...

Reader 1

... rd_lock...

rd_unlock...

Reader 2

... rd_lock...

rd_unlock...

Step 5

Writer 1

... rw_lock...

rw_unlock...

Writer 2

... rw_lock...

rw_unlock...

Reader 1

... rd_lock...

rd_unlock...

Reader 2

... rd_lock...

rd_unlock...

Step 6
Condition Variable

- Block a thread while waiting for a condition
- Condition_wait / condition_signal
- Several threads can wait for the same condition, they all get the signal
Condition Variable

- Block a thread while waiting for a condition
- `Condition_wait / condition_signal`
- Several thread can wait for the same condition, they all get the signal
Condition Variable

- Block a thread while waiting for a condition
- Condition_wait / condition_signal
- Several threads can wait for the same condition, they all get the signal
Condition Variable

- Block a thread while waiting for a condition
- `Condition_wait / condition_signal`
- Several threads can wait for the same condition, they all get the signal
Condition Variable

- Block a thread while waiting for a condition
- Condition_wait / condition_signal
- Several thread can wait for the same condition, they all get the signal

Active threads
Thread 1
Thread 2
Thread 3

Sleeping threads

... signal

... wait
... wait
... condition
Semaphores

- simple counting mutexes
- The semaphore can be hold by as many threads as the initial value of the semaphore.
- When a thread get the semaphore it decrease the internal value by 1.
- When a thread release the semaphore it increase the internal value by 1.
Semaphores

Thread 1
...
get
release
...

Thread 2
...
get
release
...

Thread 3
...
get
release
...

Semaphore (2)

Thread 1
...
get
release
...

Thread 2
...
get
release
...

Thread 3
...
get
release
...

Semaphore (1)
Semaphores

Thread 1
...
get
release
...

Thread 2
...
get
release
...

Thread 3
...
get
release
...

Semaphore (0)

Thread 1
...
get
release
...

Thread 2
...
get
release
...

Thread 3
...
get
release
...

Semaphore (0)
Semaphores

Thread 1
…
get
release
…

Thread 2
…
get
release
…

Thread 3
…
get
release
…

Semaphore (1)

Thread 1
…
get
release
…

Thread 2
…
get
release
…

Thread 3
…
get
release
…

Semaphore (1)
Atomic instruction

• Is any operation that a CPU can perform such that all results will be made visible to each CPU at the same time and whose operation is safe from interference by other CPUs
  – TestAndSet
  – CompareAndSwap
  – DoubleCompareAndSwap
  – Atomic increment
  – Atomic decrement