POSIX Threads: a first step toward parallel programming

George Bosilca
bosilca@icl.utk.edu
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

Step 2
Shared/Exclusive Locks

Legend
- Active thread
- Sleeping thread

Writer 1
- \texttt{rw\_lock}
- \texttt{rw\_unlock}

Writer 2
- \texttt{rw\_lock}
- \texttt{rw\_unlock}

Reader 1
- \texttt{rd\_lock}
- \texttt{rd\_unlock}

Reader 2
- \texttt{rd\_lock}
- \texttt{rd\_unlock}

Step 3

Step 4
Shared/Exclusive Locks

Legend
- Active thread
- Sleeping thread

- Step 5
- Step 6
Shared/Exclusive Locks

Legend

<table>
<thead>
<tr>
<th>Active thread</th>
<th>Sleeping thread</th>
</tr>
</thead>
</table>

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 7
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
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

![Diagram showing active and waiting threads](image)
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
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
...
Semaphore (1)
...
Thread 2
...
get
release
...
Semaphore (2)
...
Thread 3
...
get
release
...
Semaphores

Thread 1

... 
get
release
...

Thread 2

... 
get
release
...

Thread 3

... 
get
release
...

Semaphore (1)

Semaphore (1)
Semaphores

Thread 1
... get
release ...

Thread 2
... get
release ...

Thread 3
... get
release ...

Semaphore (1)

Semaphore (2)
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
# Pthread API

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>pthread_</td>
<td>Thread management (create/destroy/cancel/join/exit)</td>
</tr>
<tr>
<td>pthread_attr_</td>
<td>Thread attributes</td>
</tr>
<tr>
<td>pthread_mutex_</td>
<td>Mutexes</td>
</tr>
<tr>
<td>pthread_mutexattr_</td>
<td>Mutexes attributes</td>
</tr>
<tr>
<td>pthread_cond_</td>
<td>Condition variables</td>
</tr>
<tr>
<td>pthread_condattr_</td>
<td>Condition attributes</td>
</tr>
<tr>
<td>pthread_key_</td>
<td>Thread-specific data key (TLS)</td>
</tr>
<tr>
<td>pthread_rwlock_</td>
<td>Read/write locks</td>
</tr>
<tr>
<td>pthread_barrier_</td>
<td>Synchronization barriers</td>
</tr>
</tbody>
</table>
Thread Management (create)

**Attributes:** Detached or joinable state, Scheduling inheritance, Scheduling policy, Scheduling parameters, Scheduling contention scope, Stack size, Stack address, Stack guard (overflow) size

**Questions:**
- Once created what will be the status of the thread and how it will be scheduled by the OS? (use `sched_setscheduler`)
- Where it will be run? (use `sched_setaffinity` or HWLOC)