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

Thread 1  Thread 2  Thread 3
... lock lock lock...
unlock unlock unlock...
...
Mutual exclusion

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

![Diagram of mutual exclusion with examples of active and sleeping threads.]

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

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock</td>
<td>lock</td>
<td>lock</td>
</tr>
<tr>
<td>unlock</td>
<td>unlock</td>
<td>unlock</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Sleeping threads

<table>
<thead>
<tr>
<th>Thread 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock</td>
</tr>
<tr>
<td>unlock</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
**Mutual exclusion**

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

**Thread 1**

```
... lock unlock ...
```

**Thread 2**

```
... lock unlock ...
```

**Thread 3**

```
... lock unlock ...
```

**Sleeping threads**

```
mutex
```

**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.

Legend

<table>
<thead>
<tr>
<th>Active thread</th>
<th>Sleeping thread</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="rw_lock" alt="Active thread" /></td>
<td><img src="rw_unlock" alt="Sleeping thread" /></td>
</tr>
<tr>
<td><img src="rd_lock" alt="Active thread" /></td>
<td><img src="rd_unlock" alt="Sleeping thread" /></td>
</tr>
</tbody>
</table>
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 thread can wait for the same condition, they all get the signal

![Diagram of Condition Variable](image)

Active threads: Thread 1 (signal), Thread 3 (wait)
Sleeping threads: Thread 2 (wait), Thread 3 (wait)
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
...

Thread 2
...
get
release
...

Thread 3
...
get
release
...

Semaphore (2)

Semaphore (1)

Semaphore (0)

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