CS 5600

LECTURE 3

SYNCHRONIZATION

- PROCESS SYNCHRONIZATION
- VIRTUAL MEMORY

- Pictures to help understand the homework

![Diagram of memory and file system]

**Mutexes**

**Creating Mutexes**

- Disable interrupts
- Spinlock

```c
m.lock() spin_lock m.lock
    if owner == null
        owner = me
    else
        add me to queue
    tmp = owner

spinlock m.lock
    if tmp != me
        sleep
```

**Mutex Object**

As following fields:

- Lock
- Spinlock
- Owner
- Queue
  - Pointer to thread
  - Queue of threads
UNLOCK() SPIN-LOCK m.lock
if Q is empty
    owner = null
else
    owner = pop(Queue)
    wake owner
spin.unlock(m.lock)

Bounded Buffer

<table>
<thead>
<tr>
<th>WRITE(s)</th>
<th>put()</th>
<th>BUFFER</th>
<th>get()</th>
<th>READER(s)</th>
</tr>
</thead>
</table>

Circular Buffer

int head, tail
item buf[N]

put():
    while (head + 1) mod N = tail
    do nothing
    buffer[head] = item
    head = (head + 1) mod N

get is the reverse of this!

Each variable carries information in one direction
**Semaphore**

**Value** - integer

**Operations**
- `p()` \ lock \ down
- `v()` \ unlock \ up

**Semaphore** `s = new SEM(3)`

**Mutex** `m`

**Semaphore** `SPACES(N), VALUES(0)`

**Get:**
- `wait(values)`
- `lock(m)`
  - `val = buf[Tail]`
  - `Tail = Tail + 1 mod N`
- `unlock(m)`
- `signal(spaces)`

**Put:**
- `wait(spaces)`
- `lock(m)`
  - `buf[head] = val`
  - `head = head + 1 mod N`
- `unlock(m)`
- `signal(values)`

`VALUES = 0`, `SPACE = 3`

`N = 3`

`get() wait(values)`

`put()`

`wait(spaces)`

`add`

`signal(values)`

`signal(spaces)`
Now, what happens if we had this:

\[ \text{Thr1} \quad \text{2} \quad \text{Lock(A)} \quad \text{Lock(B)} \quad \text{Lock(B)} \quad \text{Lock(A)} \quad \vdots \quad \text{Unlock(B)} \quad \text{Unlock(A)} \]

**Lock Ranking**

Ordering all locks

\[ A \rightarrow B \rightarrow C \rightarrow Y \rightarrow T \]

↑ if you get a lock here, you cannot get a lock on a low level.

**Reader/Writer Problem**

Sharing reads, exclusive writes

\[ \text{Read-Lock / Unlock} \quad \text{Write-Lock / Unlock} \]

**Implementation**

mutex m, int count, sem. wlock(1)

\[ \text{RL: Lock(m)} \quad \text{RL: Lock(m)} \quad \text{WL: Wait(wlock)} \]
\[ \text{if COUNT = 0} \quad \text{if COUNT = 0} \]
\[ \text{WAIT (WLOCK)} \quad \text{SIGNAL (WLOCK)} \]
\[ \text{COUNT ++} \quad \text{COUNT --} \]
\[ \text{Unlock(m)} \quad \text{Unlock(m)} \]

deadlock!!

1) Don’t read shared variables outside of a mutex. Use a local copy instead.

2) Rank locks to avoid deadlocks.

3) Don’t sleep holding a mutex.

Page 4