UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 537
Intro to Operating Systems

A. Arpaci-Dusseau
Spring 2000

Semaphores

Questions answered in these notes:

- Why do we need semaphores?
- How are semaphores used for mutual exclusion?
- How are semaphores used for scheduling constraints?
- Examples: Join and Producer/Consumer

Reading

- Chapter 6 (up through and including 6.5)

Motivation for Semaphores

Locks only provide mutual exclusion

- Ensure that one process at a time is in critical section

May want more: Place ordering on scheduling of processes

```
cpp | cc1 | cc2 | as
```

- Producer: Creates a resource
- Consumer: Uses a resource
- Don't want producers and consumers to operate in lock step
    - Place a fixed-size buffer between processes
    - Synchronize accesses: Producers wait if full; Consumers wait if empty

Two Purposes

- Mutex: Ensure processes don't access critical section at same time
- Scheduling Constraints: Ensure processes execute in specific order

Definition of Semaphores

Synchronization variable introduced by Dijkstra in 1960s

- Mutual exclusion and scheduling control

Variable contains non-negative integer values

- Cannot read or write value directly
- Initialize to value and two basic atomic operations

semaphore.P()

- Wait for *semaphore* to be greater than zero, then decrement by one
- "Test" in Dutch (proberen)

semaphore.V()

- Increment *semaphore* by one and wake one waiting process
- "Increment" in Dutch (verhogen)

Mutual Exclusion with Semaphores

Previous example

Lock lock = new Lock();
lock.acquire().
balance += amount;
lock.release();
	Semaphore s = new Semaphore(??);
	s.P().
	balance += amount;
	s.V();

What value should s be set to initially?

Binary semaphore is sufficient for mutex

- Instead of an integer value has a boolean value
- P(): waits until value is 1, then sets it to 0
- V(): sets value to 1, waking one waiting process

Scheduling Constraints with Semaphores

/General case: One thread waits for another to reach some point

Example: Implement thread.join()

- Parent thread waits for child thread to call exit();

Semaphore joinSem = new Semaphore(??);

```
exit() {
    JoinSem.V();
}
```

```
join() {
    JoinSem.P();
}
```


Producer/Consumer Threads

Constraints on shared buffer

- Consumers must wait for a producer to fill a buffer
- Producers must wait for consumer to empty buffer, if all filled

Requires two semaphores

- emptyBuffer -- Initialize to ??
- fullBuffer -- Initialize to ??

Producer

```
emptyBuffer.P();
FillBuffer();
fullBuffer.V();
```

Consumer

```
fullBuffer.P();
UseBuffer();
emptyBuffer.V();
```

Does this provide mutual exclusion?

Scheduling and Mutual Exclusion

What if only one process can access buffer at a time?

Requires three semaphores

- emptyBuffer: Initialize to ??
- fullBuffer: Initialize to ??
- mutex: Initialize to ??

Producer

```
mutex.P();
emptyBuffer.P();
FillBuffer();
fullBuffer.V();
mutex.V();
```

Consumer

```
mutex.P();
fullBuffer.P();
UseBuffer();
emptyBuffer.V();
mutex.V();
```

Will this work?

Scheduling and Mutual Exclusion #2

Three semaphores

- emptyBuffer: Initialize to number of buffers
- fullBuffer: Initialize to 0
- mutex: Initialize to 1

Producer

```
emptyBuffer.P();
mutex.P();
FillBuffer();
mutex.V();
fullBuffer.V();
```

Consumer

```
fullBuffer.P();
mutex.P();
UseBuffer();
mutex.V();
emptyBuffer.V();
```

Will this work?