Semaphores

Questions answered in this lecture:
Why are semaphores necessary?
How are semaphores used for mutual exclusion?
How are semaphores used for scheduling constraints?
Examples: Join and Producer/Consumer

Motivation for Semaphores

Locks only provide mutual exclusion
• Ensure only one thread is in critical section at a time
May want more: Place ordering on scheduling of threads
• Example: Producer/Consumer
  – Producer: Creates a resource (data)
  – Consumer: Uses a resource (data)
• Example
  – ps | grep ”gcc” | wc
• Don’t want producers and consumers to operate in lock step
  – Place a fixed-size buffer between producers and consumers
  – Synchronize accesses to buffer
  – Producer waits if buffer full; consumer waits if buffer empty

Semaphores

Semaphores: Introduced by Dijkstra in 1960s
Semaphores have two purposes
• Mutex: Ensure threads don’t access critical section at same time
• Scheduling constraints: Ensure threads execute in specific order

Semaphore Operations

Allocate and Initialize
• Semaphore contains a non-negative integer value
• User cannot read or write value directly after initialization
  – Sem_t sem;
  – int sem_init(&sem, is_shared, init_value);
Wait or Test
• P() for ”test” in Dutch (proberen)
• Waits until value of sem is > 0, then decrements sem value
• int sem_wait(&sem);
Signal or Increment or Post
• V() for ”increment” in Dutch (verhogen)
• Increments value of semaphore
• int sem_post(&sem);
Semaphore Implementation

typedef struct {
    int value;
    queue tlist;
} semaphore;

sem_wait (semaphore *S) {  // Must be executed atomically
    S->value--;
    if (S->value < 0) {
        add this process to S->tlist;
        block();
    }
}

sem_signal (semaphore *S) { // Must be executed atomically
    S->value++;
    if (S->value <= 0) {
        remove thread t from S->tlist;
        wakeup(t);
    }
}

Semaphore Example

What happens if sem is initialized to 2?
- Scenario: Three processes call sem_wait(&sem)

Observations
- Sem value is negative --> Number of waiters on queue
- Sem value is positive --> Number of threads that can be in c.s. at some time

Mutual Exclusion with Semaphores

Previous example with locks:
Void deposit (int amount) {
    mutex_lock(&mylock);
    balance += amount;
    mutex_unlock(&mylock);
}

Example with semaphores:
Void deposit(int amount) {
    sem_wait(&sem);
    balance += amount;
    sem_signal(&sem);
}

To what value should sem be initialized???

Binary Semaphores

Binary semaphore is sufficient for mutex
- Binary semaphore has boolean value (not integer)
- bsem_wait(): Waits until value is 1, then sets to 0
- bsem_signal(): Sets value to 1, waking one waiting process

General semaphore is also called counting semaphore
### Scheduling Constraints with Semaphores

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

**Example:** Implement `thread_join()`
- Parent thread calls `thread_join()`, which must wait for child thread to call `exit()`.
- Shared sem between parent and child (created when child thread is created).

To what value is sem initialized???

<table>
<thead>
<tr>
<th>Parent thread</th>
<th>Child thread</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Thread_join()</code></td>
<td><code>exit()</code></td>
</tr>
<tr>
<td><code>sem_wait(&amp;sem)</code></td>
<td><code>sem_signal(&amp;sem)</code></td>
</tr>
</tbody>
</table>

### Producer/Consumer: Single Buffer

**Simplest case:**
- Single producer thread, single consumer thread
- Single shared buffer between producer and consumer

**Requirements**
- Consumer must wait for producer to fill buffer
- Producer must wait for consumer to empty buffer (if filled)

**Requires 2 semaphores**
- `emptyBuffer`: Initialize to ???
- `fullBuffer`: Initialize to ???

**Producer**
```
Producer
i = 0;
while (1) {
    sem_wait(&emptyBuffer);
    Fill(&buffer[i]);
    i = (i+1)%N;
    sem_signal(&fullBuffer);
}
```

**Consumer**
```
Consumer
j = 0;
while (1) {
    sem_wait(&fullBuffer);
    Use(&buffer[j]);
    j = (j+1)%N;
    sem_signal(&emptyBuffer);
}
```

### Producer/Consumer: Circular Buffer

**Next case:**
- Single producer thread, single consumer thread
- Shared buffer with N elements between producer and consumer

**Requirements**
- Consumer must wait for producer to fill buffer
- Producer must wait for consumer to empty buffer (if filled)

**Requires 2 semaphores**
- `emptyBuffer`: Initialize to ???
- `fullBuffer`: Initialize to ???

**Producer**
```
Producer
i = 0;
while (1) {
    sem_wait(&emptyBuffer);
    Fill(&buffer[i]);
    i = (i+1)%N;
    sem_signal(&fullBuffer);
}
```

**Consumer**
```
Consumer
j = 0;
while (1) {
    sem_wait(&fullBuffer);
    Use(&buffer[j]);
    j = (j+1)%N;
    sem_signal(&emptyBuffer);
}
```

### Producer/Consumer: Multiple Threads

**Final case:**
- Multiple producer threads, multiple consumer threads
- Shared buffer with N elements between producer and consumer

**Requirements**
- Consumer must wait for producer to fill buffer
- Producer must wait for consumer to empty buffer (if filled)
- Each consumer must grab unique filled element
- Each producer must grab unique empty element
- Why will previous code not work???

**Producer**
```
Producer
i = 0;
while (1) {
    sem_wait(&emptyBuffer);
    Fill(&buffer[i]);
    i = (i+1)%N;
    myi = findempty(&buffer);
    sem_signal(&fullBuffer);
}
```

**Consumer**
```
Consumer
j = 0;
while (1) {
    sem_wait(&fullBuffer);
    Use(&buffer[j]);
    j = (j+1)%N;
    myj = findfull(&buffer);
    sem_signal(&emptyBuffer);
}
```

Are `myi` and `myj` private or shared? Where is mutual exclusion needed???
Producer/Consumer: Multiple Threads

Consider three possible locations for mutual exclusion. Which work?? Which is best???

**Producer #1**

```c
sem_wait(&mutex);
sem_wait(&emptyBuffer);
myi = findempty(&buffer);
Fill(&buffer[myi]);
sem_signal(&fullBuffer);
sem_signal(&mutex);
```

**Consumer #1**

```c
sem_wait(&mutex);
sem_wait(&fullBuffer);
myj = findfull(&buffer);
Use(&buffer[myj]);
sem_signal(&emptyBuffer);
sem_signal(&mutex);
```

**Producer #2**

```c
sem_wait(&emptyBuffer);
sem_wait(&mutex);
myi = findempty(&buffer);
Fill(&buffer[myi]);
sem_signal(&fullBuffer);
sem_signal(&mutex);
```

**Consumer #2**

```c
sem_wait(&fullBuffer);
sem_wait(&mutex);
myj = findfull(&buffer);
Use(&buffer[myj]);
sem_signal(&emptyBuffer);
sem_signal(&mutex);
```

**Producer #3**

```c
sem_wait(&emptyBuffer);
sem_wait(&mutex);
myi = findempty(&buffer);
Fill(&buffer[myi]);
sem_signal(&fullBuffer);
```

**Consumer #3**

```c
sem_wait(&fullBuffer);
sem_wait(&mutex);
myj = findfull(&buffer);
Use(&buffer[myj]);
sem_signal(&emptyBuffer);
```