Based on slides by Harsha V. Madhyastha

EECS 482 Introduction to Operating Systems

Spring/Summer 2020

Lecture 6: Semaphores

Nicole Hamilton
https://web.eecs.umich.edu/~nham/
nham@umich.edu

Project 2 is out

Implement a thread library

- Create threads
- Switch between threads
- Manage interactions (locks and CVs)
- Optionally schedule threads on multiple CPUs

Lectures next week will begin to tell you how to do the project.

- Go over spec and handout code before then.

Recap

Multi-threaded code with monitors:

- Locks for mutual exclusion
- Condition variables for ordering constraints

Every thread is in one of the following states:

- Running outside any critical section
- Running inside a critical section
- Waiting on a mutex to enter a critical section
- Waiting on a cv inside a critical section

Semaphores

Generalized lock/unlock.

A non-negative integer initialized to user-specified value and two operations:

down( ) waits for the value to become positive and atomically decrements it.

up( ) increments the value atomically.

```
void down( )
   {
   // Wait for semaphore value
   // to become positive, then
   // decrement value by 1.

   while ( true )
      if ( value > 0 )
         {
         value--;
         break;
         }
   }

void up( )
   {
   value++;
   }
```

Atomic

Atomic

Two types of semaphores

Mutex semaphore (or binary semaphore)

- Represents single resource (critical section)
- Up() atomically **sets** value to 1

Counting semaphore (or general semaphore)

- Represents a resource with many units, or a resource that allows concurrent access (e.g., reading)
- Multiple threads can up/down the semaphore

Uses of Semaphores

Mutual exclusion

```
semaphore sem( 1 );
sem.down( );
critical section;
sem.up( );
```

Ordering constraints

Example: thread B wants to wait for thread A to finish.

```
semaphore sem( 0 );

// Thread A
do task
sem.up( );

// Thread B
sem.down( );
continue execution;
```

Coke machine with semaphores

As before, think about shared data, mutual exclusion, and before-after relations

Assign semaphore for each:

mutex: for exclusive access to coke machine

fullSlots: before removing a coke, cokes > 0

Counts filled slots in machine

emptySlots: before adding a coke, cokes < MAX

Counts free spaces in the machine

Coke machine with semaphores

```
// Initialization
Semaphore mutex = 1;
Semaphore emptySlots = N;
Semaphore fullSlots = 0;
```

```
producer( )
   {
   // wait for empty slot
   emptySlots.down( );

   mutex.down( );
   Add coke to machine;
   mutex.up( );

   // note a full slot
   fullSlots.up( );
   }
```

```
consumer( )
   {
   // wait for full slot
   fullSlots.down( );

   mutex.down( );
   Take coke from machine;
   mutex.up( );

   // note an empty slot
   emptySlots.up( );
   }
```

Coke machine with monitors

```
Producer( )
   {
   // wait for empty slot
   cokeLock.lock( );
   while ( numCokes == MAX )
      waitingProducers.wait(
         &cokeLock );

   Add coke to machine;
   numCokes++;

   // note a full slot
   waitingConsumers.signal( );
   cokeLock.unlock( );
   }
```

```
Consumer( )
   {
   // wait for full slot
   cokeLock.lock( );
   while ( numCokes == 0 )
      waitingConsumers.wait(
         &cokeLock );

   Take coke from machine;
   numCokes--

   // note an empty slot
   waitingProducers.signal( );
   cokeLock.unlock( );
   }
```

Questions

1. What if there's 1 full slot, and multiple consumers call down() at the same time?
2. Why do we need different semaphores for fullSlots and emptySlots?
3. Does the order of down() calls matter?
4. Does the order of up() calls matter?
5. What if a context switch happens between emptySlots.down() and mutex.down()?
6. What if fullSlots.up() before mutex.down()?

```
producer( )
    {
    // wait for empty slot
    emptySlots.down();

    mutex.down( );
    Add coke to machine;
    mutex.up( );

    // note a full slot
    fullSlots.up();
    }

consumer( )
    {
    // wait for full slot
    fullSlots.down( );

    mutex.down( );
    Take coke from machine;
    mutex.up( );

    // note an empty slot
    emptySlots.up( );
    }
```

Readers/Writers with Semaphores

Use three variables

- integer readcount – number of threads reading
- Semaphore mutex – control access to readcount
- Semaphore w_or_r – write-mode or read-mode

Readers/Writers with Semaphores

```
// number of readers
int readcount = 0;
// mutual exclusion to readcount
Semaphore mutex = 1;
// write-mode or read-mode
Semaphore w_or_r = 1;

writer
   {
   w_or_r.down( );
   Write;
   w_or_r.up( );
   }
```

```
reader
   {
   mutex.down( );
   readcount++;
   if ( readcount == 1 )
      w_or_r.down( );
   mutex.up( );
   Read;
   mutex.down( );
   readcount--;
   if ( readcount == 0 )
      w_or_r.up( );
   mutex.up( );
   }
```

Questions

1. Why don't writers use mutex?
2. If a writer is writing, where will readers be waiting?
3. Once a writer exits, which reader gets to go first?
4. Is it guaranteed that all readers will fall through?
5. What if mutex.up() is above "if (readcount == 1)"?
6. If read in progress when writer arrives, when can writer get access?

```
writer
   {
   w_or_r.down( );
   Write;
   w_or_r.up( );
   }

reader
   {
   mutex.down( );
   readcount++;
   if ( readcount == 1 )
      w_or_r.down( );
   mutex.up( );
   Read;
   mutex.down( );
   readcount--;
   if ( readcount == 0 )
      w_or_r.up( );
   mutex.up( );
   }
```

Monitors vs. Semaphores

Semaphores: 1 mechanism for both mutual exclusion and ordering

- Elegant
- Can be difficult to use

Monitor lock = binary semaphore (initialized to 1)

- lock() = down()
- unlock() = up()

Condition variable versus semaphore

Condition variable	**Semaphore**
while(cond) {wait();}	down()
Can safely handle spurious wakeups	No spurious wakeups
Conditional code in user program; more	Conditional code in semaphore defi
User p protec	Sem variable (inte ations on that variable (down, up)
No memory of past signals	Remembers past up calls

```
T1: wait()
T2: signal()
T3: signal()
T4: wait()
```

```
T1: down()
T2: up()
T3: up()
T4: down()
```

Implementing custom waiting condition with semaphores

Semaphores work best if the shared integer and waiting condition (value==0) map naturally to problem domain

How to implement custom waiting condition with semaphores?

Producer-consumer with monitors

```
Consumer( )
   {
   cokeLock.lock( );

   while ( numCokes == 0 )
      waitingConsumers.wait(
         &cokeLock );

   take coke out of machine;
   numCokes--;

   waitingProducers.signal( );

   cokeLock.unlock( );
   }
```

```
Producer( )
   {
   cokeLock.lock( );

   while ( numCokes == MAX )
      waitingProducers.wait(
         &cokeLock );

   add coke to machine;
   numCokes++;

   waitingConsumers.signal( );

   cokeLock.unlock( );
   }
```

Producer-consumer with semaphores, monitor style

```
Consumer( )
   {
   mutex.down()
   while ( numCokes == 0 )
      {

      go to sleep;



      }
   take coke out of machine;
   numCokes--;

   wake up any waiting producer;



   mutex.up( );
   }
```

```
Producer( )
   {
   mutex.down()
   while ( numCokes == MAX )
      {

      go to sleep;



      }
   add coke to machine
   numCokes++

   wake up any waiting consumer;



   mutex.up( );
   }
```

Producer-consumer with semaphores, monitor style

```
Consumer( )
   {
   mutex.down()
   while ( numCokes == 0 )
      {
      semaphore s = 0;
      waitingConsumers.push( &s );
      mutex.up( );
      s.down( );
      mutex.down( );
      }
   take coke out of machine;
   numCokes--;
   if ( !waitingProducers.empty( ) )
      {
      waitingProducers.front()->up( );
      waitingProducers.pop( );
      }
   mutex.up( );
   }
```

```
Producer( )
   {
   mutex.down()
   while ( numCokes == MAX )
      {
      semaphore s = 0;
      waitingProducers.push( &s );
      mutex.up( );
      s.down( );
      mutex.down( );
      }
   add coke to machine
   numCokes++
   if ( !waitingConsumers.empty( ) )
      {
      waitingConsumers.front()->up();
      waitingConsumers.pop( );
      }
   mutex.up( );
   }
```

Exercise to try …

Convert monitor-style reader/writer lock implementation to use semaphores