HW 1: Processes, Threads, and Concurrency

Solutions

February 11, 2009

1 Scheduling

a) Shortest Job First (SJF)
SJF is unfair - large jobs may be starved.

b) Processor Sharing (PS)
PS is fair - all jobs are running continuously with a share of the processor.

c) Priority Scheduling (PRI)
PRI is unfair - large jobs may be starved.

2 Who needs locks? - Let's use memory

a) Taking Turns

i) Yes, a thread can only enter on it's turn. For two threads to be in critical section simultaniously, turn would have to somehow change from one of their ids to the other while one is in there. As it changes only when one exists the critical section, this is impossible.

ii) and iii) These are not really true because they depend on both threads

maintaining interest in running the critical section. If one thread loses interest in the critical section, however, the other threads can end up waiting forever. As long as both threads remain interested, though, they will both continue to be able to gain access.

b) Being Polite

i) Yes, it is impossible for both threads to enter the critical section at the same time. Once a thread passes `want[my_id] = true`, the other thread

can no longer exit the while loop. As a thread must reach this statement before the while loop, once one thread reaches the while loop, it's impossible for the other thread to exit.

ii) No, it is possible to acheive deadlock. If both threads run `want[my_id] = true` before either thread exits the while loop, then both threads will get stuck forever in the while loop.

iii) No, a deadlock exists as described above.

3 Internet orders and semaphores

```
Semaphore roses;
Semaphore chocolates;
Semaphore orders;

void placeOrder(unsigned int chocolateBoxes, unsigned int numRoses)
{
  for (int i = 0; i < chocolateBoxes; i++)
    chocolates.V();

  for (int i = 0; i < numRoses; i++)
    roses.V();

  orders.P();
}

void packAndDeliverOrders()
{
  while(beforeValentinesDay) {
    orders.V();

    packOrder();
    doDelivery();
  }
}

void boughtChocolate(unsigned int chocolateBoxes) {
  for (int i = 0; i < chocolateBoxes; i++)
    chocolates.P();
```

```
}

void boughtRoses(unsigned int numRoses) {
  for (int i = 0; i < numRoses; i++)
    roses.P();
}
```

4 Computer Lab Monitor

using Mesa semantics:

```
int scanners = 5;
int videoCapture = 3;
int cardReaders = 8;

int photoWaiters = 0;
ConditionVariable photoAvailable;

int filmWaiters = 0;
ConditionVariable filmAvailable;

int musicWaiters = 0;
ConditionVariable musicAvailable;

void LogMusicOn() {
  musicWaiters++;

  while(cardReaders == 0)
    musicAvailable.wait();

  cardReaders--;

  musicWaiters--;
}

void LogMusicOff() {
  cardReaders++;

  if (musicWaiters)
```

```
    musicAvailable.signal();
  else if (photoWaiters && scanners)
    photoAvailable.signal();
  else if (filmWaiters && videoCapture)
    filmAvailable.signal();
}

void LogPhotoOn() {
  photoWaiters++;

  while(scanners == 0 || cardReaders == 0)
    photoAvailable.wait();

  scanners--;
  cardReaders--;

  photoWaiters--;
}

void LogPhotoOff() {
  scanners++;
  cardReaders++;

  if (photoWaiters)
    photoAvailable.signal();
  else if (musicWaiters)
    musicAvailable.signal();
  else if (filmWaiters && videoCapture)
    filmAvailable.signal();
}

void LogFilmOn() {
  filmWaiters++;

  while(videoCapture == 0 || cardReaders == 0)
    filmAvailable.wait();

  videoCapture--;
  cardReaders--;
```

```
  filmWaiters--;
}

void LogFilmOff() {
  videoCapture++;
  cardReaders++;

  if (filmWaiters)
    filmAvailable.signal();
  else if (musicWaiters)
    musicAvailable.signal();
  else if (photoWaiters && scanners)
    photoAvailable.signal();
}
```