Computer Science COMP-409, Fall 2010

Concurrent Programming

Midterm Solutions

1. A *modulo semaphore* is similar to a counting semaphore. Rather than a normal `down()` operation, however, it provides a `down(int m)` operation. This generalizes `down()`, causing a caller to block as long as the count is 0 mod m instead of just 0.

   Give an implementation in pseudo-code of modulo semaphores using monitors and condition variables. **5**

```
public class MS {
   private int count;
   public MS(int init) { count = init; }
   public synchronized void down(int m) {
      while (count % m == 0)
         try { wait(); } catch(InterruptedException ie) {}
      count--;
      notifyAll(); // decrementing may change waiting conditions
   }
   public synchronized void up() {
      count++;
      notifyAll(); // incrementing may change waiting conditions
   }
}
```

2. LL/SC and CAS are supposed to be at least as general as FA and TS. **5**

   Show how to implement FA and TS using LL/SC.

```
FA(x,v)
redo: lwarx r1,0,x
      addi r2,r1,v
      stwcx.r2,0,x
      bne redo
      return r1
```

```
TS(x,y)
redo: lwarx r1,0,x
      stwcx.y,0,x
      bne redo
      return r1
```

   Show how to implement FA and TS using CAS.

```
FA(x,v)
do {
      r1 = x;
      r2 = r1+v;
      r3 = CAS(x,r1,r2)
} while(r3!=r1);
return r1;
```

```
TS(x,y)
do {
      r1 = x
      r2 = CAS(x,r1,y)
} while(r2!=r1);
return r1;
```

3. A filing system is indexed by alphabetic strings. Suppose multiple threads (repeatedly) need to access a set of files, each thread requiring simultaneous and exclusive access of 3 files at a time. Each file is protected by a mutex. Give pseudo code for blocking functions:

   `acquire(String f1, String f2, String f3)`

   and:

   `release(String f1, String f2, String f3)`

   Your solution should never deadlock. **5**

```
acquire(String f1, String f2, String f3) {
   // sort the names first, lexicographically
   // and always lock in order:  no deadlock possible.
   String s1 = min(f1,min(f2,f3));
   String s3 = max(f1,max(f2,f3));
   String s2 = (s1==f1) ?  ((s2==f2) ?  f3 :  f2) :
         (s1==f2) ?  ((s2==f1) ?  f3 :  f1) :
         ((s2==f1) ?  f2 :  f1);
   s1.lock();
   s2.lock();
   s3.lock();
}
release(String f1, String f2, String f3) {
   f1.unlock(); // no special order required to unlock
   f2.unlock();
   f3.unlock();
}
```

4. A *3-way rendezvous* allows 3 threads to synchronously trade data. Thread A executes a = rendezvous(dataA), while thread B executes b = rendezvous(dataB), and C executes c = rendezvous(dataC). Each thread blocks until both of the others are at the rendezvous call. Once all are there, the data is exchanged, a==dataC and b==dataA and c==dataB, and the threads may proceed.

   Show how to build rendezvous. You may use any blocking synchronization. **5**

```
              sa=sb=sc=0
A                 B                 C
dataA=...         dataB=...         dataC=...
up(sb);           up(sc)            up(sa)
up(sb);           up(sc)            up(sa)
down(sa);         down(sa)          down(sb)
down(sc);         down(sb)          down(sc)
a=dataC           b=dataA           c=dataB
// another barrier after is required to allow it to be
// reused.
```

5. An algorithm is divided into n stages, which must be executed in sequence. Each stage takes equal time if executed sequentially, but has different parallelization limits: stage i ($i = 0, \ldots, n-1$) can be efficiently parallelized by only up to 2^i threads, after which performance of that stage is not improved by more threads. **6**

   What is the maximum speedup possible given an infinite number of threads for $n = 4$?

   With 4 sections, the total speedup can be calculated using Amdahl's law. Assume unit time for the sequential version. Then the speedup from the parallel version with as many threads as possible would be:

$$\frac{1}{(\frac{1}{4})/2^0 + (\frac{1}{4})/2^1 + (\frac{1}{4})/2^2 + (\frac{1}{4})/2^3}$$

   This reduces to:

$$\frac{1}{\frac{8}{32} + \frac{4}{32} + \frac{2}{32} + \frac{1}{32}}$$

   and then to:

$$\frac{32}{15}$$

   Develop a general formula; what is the maximum speedup for an arbitrary given $n > 1$?

   Each of the $1/n$ stages takes $1/n$ of the unit sequential time. Thus, total speedup is given by:

$$\frac{1}{\sum_{i=0}^{n-1} (\frac{1}{n}\frac{1}{2^i})}$$

   which reduces to:

$$\frac{1}{\frac{1}{n}\sum_{i=0}^{n-1} (\frac{1}{2^i})}$$

   and to:

$$\frac{n}{\sum_{i=0}^{n-1} (\frac{1}{2^i})}$$

6. In class you saw priority inversion explained using 3 threads and a single mutex in a single CPU context. Assume a strict priority-preemptive model. **4**

   Can priority inversion ever happen with 3 threads, a single lock, and a two-CPU system? What if there are 4 threads available (the 4th of any priority)?

   With 2 CPUs and 3 threads, no---a priority preemptive model would require the two highest priority threads be executing. If one blocked, then the third thread could certainly make progress. With 4 threads yes: the additional thread can be active on the 2nd CPU, reducing the problem to the original one of 3 threads on a single CPU.

   In general, under what combinations of n threads and m CPUs, can priority inversion occur?

   When $n > m + 1$ priority inversion can occur. $m - 1$ threads can be active and consuming $m - 1$ CPUs, leaving a single CPU with 3 threads.