Homework 8 Master Solutions

April 21, 2013

1 Problem 1

The following code was proposed (by David Seidman CS-350 class of S'04) as the rendezvous code for N processes using N semaphores:

```
signal(semaphore[i]);
for(j = 0; j < N; j++) {
  wait(semaphore[j]);
  signal(semaphore[j]);
}
[[Rendezvous Code Here]]]
```

1. Prove that the above code will work. What should be the initialization of semaphore[i]?

   - The code will work. To see that, it should be sufficient to point out that each process is blocking while waiting for all other processes to arrive. Once all processes arrive, then all processes will be able to leave (since all processes would have signaled to each other).

2. When presented with the above code, another student suggested that a better solution is the one below.

   ```
   signal(semaphore[i]);
   for(j = 0; j < N; j++) {
     wait(semaphore[j]);
     signal(semaphore[j]);
   }
   [[Rendezvous Code Here]]]
   wait(semaphore[i]);
   ```

   Could a process ever block as a result of executing the last instruction in the above code? If yes, give an example, if not, then explain why not.

   - Yes it could block. Consider the case of two processes P[0] and P[1], if P[0] follows normally and blocks on waiting for P[1] signal, and P[1] is interrupted as soon as it signals S[1], and before entering the for loop, P[0] could potentially exit the loop.

3. Why do you think the last instruction in the above code is included?

   - I think it's included to reset the semaphores to their original values, but it was not placed correctly.

2 Problem 2

Strict Turns. You are to write a procedure TakeTurns(int) that will be called by any one of N different processes, where N is a known constant. The argument to TakeTurns is an integer between 0 and N-1 that uniquely identifies the calling process. The result of calling TakeTurns by a process Pi is one of two things: (1) if the last process to have "taken its turn" is Pi-1, then TakeTurn will simply print "It is the turn of process i", (2) otherwise, Pi will block until Pi-1 is able to take its turn. The intended behavior is that the following set of strings be printed endlessly: "It is the turn of process 0", "It is the turn of process 1" , "It is the turn of process 2" , ... "It is the turn of process N-1" , "It is the turn of process 0". Implement TakeTurns in Java using the least number of semaphores, and show that it works. [here is a primer on using semaphores in Java]

Example solution with N semaphores

```
static int N = 4;
static volatile Semaphore[] lockThreads = new Semaphore[N];

private int id;
public Strict(int i) { id = i; }

private void TakeTurns(int id)
{
try
{
lockThreads[id].acquire();

System.out.println("It is turn of thread " + id);


lockThreads[(id+1)%N].release();
}
catch(InterruptedException e){}
}

public void run()
{
for(int i=0; i<10; i++)
{
TakeTurns(id);
```

```
}
}


public static void main(String[] args)
{
for (int i = 0; i < N; i++)
{
if(i==0)
lockThreads[i] = new Semaphore(1);
else
lockThreads[i] = new Semaphore(0);
Example e = new Example(i);
e.start();
}
}
```

3 Problem 3

Fair Access. It is desirable that the following fairness property be enforced with respect to access to some resource: "It is never the case that a process is able to access the resource for the Kth time if any other process has accessed the resource for less than K-1 times." In other words, the number of accesses by difference processes cannot differ by more than one. Using semaphores implement a function FairAccess(int) to be used by processes to access the resource according to the fairness criterion described above. The argument to FairAccess is an integer between 0 and N-1 that uniquely identifies the calling process. Calling FairAccess(i) by a process Pi will do the following: if granting process Pi access to the resource will result in a violation of the fairness property, then Pi blocks, otherwise access is granted. Implement FairAccess in Java using semaphores and show that it works. [here is a primer on using semaphores in Java]

4 Problem 4

Laundromat Brawl. The local laundromat has just entered the computer age. As each customer enters, he or she puts coins into slots at one of two stations and types in the number of washing machines he/she will need. The stations are connected to a central computer that automatically assigns available machines and outputs tokens that identify the machines to be used. The customer puts laundry into the machines and inserts each token into the machine indicated on the token. When a machine finishes its cycle, it informs the computer that it is available again. The computer maintains a boolean array available[M] to represent if corresponding machines are available (M is a constant indicating how many machines there are in the laundromat), and a semaphore nfree that

indicates how many machines are available. The pseudo code to allocate and release machines is as follows. The available array is initialized to all true, and nfree is initialized to M.

```
int allocate() /* Returns index of available machine */
{
  wait(nfree); /* Wait until a machine is available */
  for (int i=0; i < M; i++)
    if (available[i]) {
      available[i] = FALSE;
      return i;
    }
}
void return(int machine) /* Make machine available again */
{
  available[machine] = TRUE;
  signal(nfree);
}
```

1. It seems that if two people make requests at the two stations at the same time, they will occasionally be assigned the same machine. This has resulted in several brawls in the laundromat, and you have been called in by the owner to fix the problem. Assume that one thread handles each customer station. Explain how the same washing machine can be assigned to two different customers.

   - If there are 2 or more machines free, 2 or more threads can get to execute the for loop. Thus 2 or more machines may be reading and writing the variable available[i], and infer that machine i is available.

2. Modify the pseudo code above to eliminate the problem. You need not implement this in Java, but feel free to do so if you wish.

   - Protect the if statement (or the entire for statement) with a mutex semaphore:

     ```
     binary_sem mutex=1;
     wait(mutex)
     for(..)
     signal(mutex)
     ```

5 Problem 5

Critical Section with Priorities: A set of processes share a critical section of code, which is to be accessed in a mutually exclusive fashion. Each process is assigned a priority according to its ID – the lower the ID, the higher the priority, so P1 is higher priority than P2. If two or more processes need to

access the critical section, then the one with "higher priority" enters the critical section next. The following is a sketch of the solution for this problem using semaphores:

1. We keep a vector V[] of booleans. V[i] is "True" if Pi needs to use the critical section.
2. We use a vector of binary semaphores B[] to block processes from entering their critical section: B[i] will be the semaphore blocking process Pi.
3. A special scheduler process SCHED is used whenever a blocked process needs to be awakened to use the critical section.
4. SCHED is blocked by waiting on a special semaphore S.
5. When a process Pi needs to enter the critical section, it sets V[i] to "True", signals the semaphore S and then waits on the semaphore B[i].
6. Whenever SCHED is unblocked, it selects the process Pi (if any) with the smallest index i for which V[i] is "True". Process Pi is then awakened by signaling B[i]. SCHED goes back to sleep by blocking on semaphore S.
7. When a process Pi leaves the critical section, it signals S.

Answer the questions below:

1. Implement the operation of the above sketched system in Java. Clearly explain the various data structures you use and initialization thereof.

   - Code:

```
import java.util.Random;
import java.util.concurrent.Semaphore;

//AUTHOR: VATCHE ISHAKIAN
//Class: CS350
//Critical Section with Priorities:

public class question1 extends Thread{
private static int N = 0;
private static int V[] = new int[5];
private static Semaphore B[] = new Semaphore[5];
private static Semaphore mutex1 =  new Semaphore(1, false); //mutex1 semaphore
private static Semaphore S =  new Semaphore(0, false); //semaphore to invoke schedu
private static Semaphore output =  new Semaphore(1, false); //output semaphore


    private int id;
    private String ProcessType;
    private static Random rand = new Random();
```

```
 public question1(int i, String i_ProcessType) {
     id = i;
     ProcessType = i_ProcessType;
     B[id] = new Semaphore(0, false);
 }

 private void busy(int delay) {
     try {
      sleep(rand.nextInt(delay));
     } catch (InterruptedException e) {}
 }

 public void run()
 {
  if (ProcessType =="PROCESS")
  {
  while(true)
  {
  try {
      mutex1.acquire();
  } catch (InterruptedException e){} ;
  N++;
  try {
  output.acquire();
    } catch (InterruptedException e){} ;
    System.out.println("Process " + id + " is requesting CS");
    output.release();
    V[id] = 1;
    if( N == 1)
    {
    S.release();
    }
    mutex1.release();
try {
B[id].acquire();
    } catch (InterruptedException e){} ;

    try {
  output.acquire();
    } catch (InterruptedException e){} ;
    System.out.println("Process " + id + " is in the critical section");
    busy(200);
    output.release();
    //remainder section
    try {
```

```
            mutex1.acquire();
          } catch (InterruptedException e){} ;
          V[id] = 0;
          N--;
          try {
            output.acquire();
        } catch (InterruptedException e){} ;
        System.out.println("Process " + id + " is exiting the CS.");
        output.release();
        if(N > 0)
        {
        S.release();
        }
        mutex1.release();
        busy(200);
        }
        }
        else //The process is a SCHED
        {
        while(true)
        {
        try {
                 S.acquire();
             } catch (InterruptedException e){} ;
try {
            mutex1.acquire();
        } catch (InterruptedException e){} ;
        for(int j=0; j<5 ; j=j+1)
        {
        if(V[j] == 1){
        try {
                 output.acquire();
             } catch (InterruptedException e){} ;
             System.out.println("Process " + j + " is signaled.");
        output.release();
        B[j].release();
        break;
        }
        }
        mutex1.release();
        }
        }
       }

       public static void main(String[] args) {
        question1[] p1 = new question1[5];
```

```
        question1 SCHED;

           SCHED= new question1(0, "SCHED");
           SCHED.start();

           for (int i = 0; i < 5; i++) {
               p1[i] = new question1(i, "PROCESS");
               p1[i].start();
           }
       }
   }
```

2. Show your Java code in action. Assume that there are five processes P0, P1, P2, ... P5 with P0 being the highest priority, P1 being the next, etc. Each process requests the critical section for a total of five times (say in a loop from 1 to 5). Process Pi should print "Pi is requesting CS" before proceeding with the step described in (e) above. It should print "Pi is in the CS" whenever it is in the critical section. And, it should print "Pi is exiting the CS" when it gets to the step described in (g). Your code needs fixing if you get a printout indicating that two processes Pi and Pj (where i¡j) are requesting the CS and Pj gets to the CS before Pi (even if Pj requested it before Pi).

   - Solutions will vary, but should be obviously right or wrong.

3. Explain how you could use the above template to design a protocol for a set of processes to access the CS according to some arbitrary metric – e.g., using EDF based on a deadline imposed on the process, or by accounting for the number of times a process was allowed to enter the CS, and giving priority to the process that has used the CS the least so far in its lifetime, etc.

   - Priority is currently based on process ID. Simply remap the priority level using the dispatching function associated with the scheduler you want to implement.