CS4410 - Fall 2008
Homework 3 Solution
Due October 7, 11:59PM

Q1. A single-lane bridge connects the two Vermont villages of North Tunbridge and South Tunbridge. Farmers in the two villages use this bridge to deliver their produce to the neighboring town. The bridge can become deadlocked if both a northbound and a southbound farmer get on the bridge at the same time (Vermont farmers are stubborn and are unable to back up).

a. Using exactly one semaphore, design an algorithm that prevents deadlock. Do not be concerned about starvation and inefficency.

b. Provide a solution using Monitor that is starvation-free.

Answer:

a.

```
semaphore ok_to_cross = 1;

void enter_bridge() {
      P(ok_to_cross);
}

void exit_bridge() {
      V(ok_ to_cross);
}
```

b.

```
/*the correct-direction enter function must be called before getting on
the bridge.  The corresponding exit function must be called when the
thread is ready to leave the bridge.*/

monitor bridge {
      int num_waiting_north = 0;
      int num_waiting_south = 0;
      int on_bridge = 0;
      condition ok_to_cross;
      int prev = 0;

      void enter_bridge_north() {
            num_waiting_north++;
            while (on_bridge ||(prev == 0 && num_waiting_south > 0))
                  ok_to_cross.wait();
            on_bridge=1;
            num_waiting_north--;
            prev = 0;
      }

      void exit_bridge_north() {
            on_bridge = 0;
            ok_to_cross.broadcast();
      }

      void enter_bridge_south() {
            num_waiting_south++;
            while (on_bridge ||(prev == 1 && num_waiting_north > 0))
                  ok_to_cross.wait();
            on_bridge=1;
            num_waiting_south--;
            prev = 1;
      }


      void exit_bridge_south() {
            on_bridge = 0;
            ok_to_cross.broadcast();
      }

}
```

Note that this problem can have many alternative solutions. One alternative solution is as follows (adapted from one student's homework submission):

```
Monitor Bridge {
      int nWaiting=0, sWaiting=0, sOnBridge=0, nOnBridge=0;
      condition north_turn, south_turn;
      enterNorth() {
            if (sWaiting>0 || sOnBridge>0)
            {
                  nWaiting++;
                  wait(north_turn);
                  while (sOnBridge>0)
                        wait(north_turn);
                  nWaiting--;
            }
            nOnBridge++;
            if (sWaiting==0)
                  signal(north_turn);
      }
      exitNorth() {
            nOnBridge--;
            if (nOnBridge ==0)
                  signal(south_turn);
      }
      enterSouth() {
            if (nWaiting>0 || nOnBridge>0)
            {
                  sWaiting++;
                  wait(south_turn);
                  while (nOnBridge>0)
                        wait(south_turn);
                  sWaiting--;
            }
            sOnBridge++;
            if (nWaiting==0)
                  signal(south_turn);
      }
      exitSouth() {
            sOnBridge- -;
            if(sOnBridge == 0)
                  signal(north_turn);
      }
}
```

Grading guide:
Q1(b) -2 Wait on a condition variable is not matched by signal/broadcast as a result the blocked process will never wake up.
Q1(b) -2 solution is not starvation-free.
Q1(b): -2 wait on Condition and semaphore is not same. First wait on a Condition blocked the process. So your solution blocked all farmers and no one can enter the bridge.
Q1(b): -1 Code is not placed inside a monitor
Q1. -2 Solution does not contain any pseudo-code/program. Only solution approach is provided.
Comment (no point deducted): Many of you used "if" instead of "while" before wait on a conditional variable. For mesa style monitor (actual implementation of monitor is of this type), you have to use "while" because Condition might not hold when waiter returns. See class lecture slides for details.

Q2. If the P() and V() semaphore operations are not executed atomically, then can mutual exclusion be violated? Why or why not?

Answer:
Yes, mutual exclusion can be violated. A P operation atomically decrements the value associated with a semaphore. If two P operations are executed on a semaphore when its value is 1, if the two operations are not performed atomically, then it is possible that both operations might proceed to decrement the semaphore value, thereby violating mutual exclusion.

Grading guide:
Almost all of you got this question right!

Q3. Consider the following snapshot of a system:

	Allocation	*Max*	*Available*
	A B C D	A B C D	A B C D
P_0	0 0 1 2	0 0 1 2	1 5 2 0
P_1	1 0 0 0	1 7 5 0	
P_2	1 3 5 4	2 3 5 6	
P_3	0 6 3 2	0 6 5 2	
P_4	0 0 1 4	0 6 5 6	

Answer the following questions using the banker's algorithm:

a. What is the content of the matrix Need?

b. Is the system in a safe state?

c. If a request from process P_1 arrives for (0,4,2,0), can the request be granted immediately?

Answer:

a. The values of *Need* for processes P_0 through P_4 respectively are (0, 0, 0, 0), (0, 7, 5, 0), (1,0, 0, 2), (0, 0, 2, 0), and (0, 6, 4, 2).

b. Yes. With *Available* being equal to (1,5, 2, 0), either process P_0 or P_3 could run. Once process P3 runs, it releases its resources, which allow all other existing processes to run.

c. Yes, it can. This results in the value of *Available* being (1, 1, 0, 0). One ordering of processes that can finish is P_0, P_2, P_3, P_1, and P_4.

Q4. Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Is this system deadlock-free? Why or why not?

Answer:
Yes, this system is deadlock-free.
Proof by contradiction. Suppose the system is deadlocked. This implies that each process is holding one resource and is waiting for one more. Since there are three processes and four resources, one process must be able to obtain two resources. This process requires no more resources and, therefore it will return its resources when done.

Q5. Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)?Which algorithm makes the most efficient use of memory?

Answer:

a. First-fit:
1. 212K is put in 500K partition
2. 417K is put in 600K partition
3. 112K is put in 288K partition (new partition 288K = 500K − 212K)
4. 426K must wait

b. Best-fit:
1. 212K is put in 300K partition
2. 417K is put in 500K partition
3. 112K is put in 200K partition
4. 426K is put in 600K partition

c. Worst-fit:
1. 212K is put in 600K partition
2. 417K is put in 500K partition
3. 112K is put in 388K partition
4. 426K must wait

In this example, best-fit turns out to be the best.