CMPSCI 377: Operating Systems

Solution to homework 3: Monitors and Deadlock

1. (20 pts) **Semaphores & Monitors.**

   (a) (16 pts) Solve the candy shop problem with semaphores and with monitors. In the candy shop, a customer enters and takes a number (don't worry about running out of numbers). Some number of sales people wait on them in order of their arrival. If a sales person is free when the customer takes a number, the sales person waits on the customer right away (assume a FIFO queue on the wait queue). If no sales person is free when the customer takes a number, the customer waits. Write `Enter`, and `Finish` routines for the customers.

   Solution:

   Monitor:

```
class CandyShop {
   public:
     void Enter(); void Finish();
   private:
     Lock lock;   // control access to shared variables
     int ticket, served, availSalesPeople;
     CVar service; // Sales people
}
CandyShop::CandyShop(int n) {
     lock.value = 1;
     ticket = 0;
     served = 0;
     availSalesPeople = n;      // Number of Sales people
}
CandyShop::Enter() {
   lock.Wait();
   ticket++;
   if (availSalesPeople == 0) {
      service.Wait(lock);
   }
   availSalesPeople--;
   served++;
   lock.Signal();
}
CandyShop:: Finish() {
   lock.Wait();
   availSalesPeople++;
   if (served < ticket)
      service.Signal();
   lock.Signal();
}
```

Semaphore:

```
class CandyShop {
   public:
     void Enter(); void Finish();
   private:
     int ticket;
     Semaphore mutex;   // control access to shared variables
     Semaphore servers; // Sales people
}
CandyShop::CandyShop(int n) {
     ticket = 0;
     mutex.value = 1;
     servers.value = n;  // Number of Sales people
}
CandyShop::Enter() {
   mutex.Wait();
   ticket++;
   mutex.Signal();
   servers.Wait();
}
CandyShop:: Finish() {
   servers.Signal();
}
```

(b) (4 pts) Which of your solution is easier to understand and thus get correct?

Solution: The semaphore solution is better in this case because the counted semaphore exactly captures the multiple sales people, whereas in the monitor solution we need additional variables, counters, and conditional tests to track the sales people.

2. (10 pts) **Monitors.** Write a monitor solution to the north-south tunnel problem. Suppose a two-way (north-south), two-lane road contains a long one-lane tunnel. A southbound (or northbound) car can only use the tunnel if there are no oncoming cars in the tunnel. Because of accidents, a signaling system has been installed at the entrances to the tunnel. When a car approaches the tunnel, a sensor notifies the controller computer by calling a function `arrive` with the car's travel direction (north or south). When a car exits the tunnel, the sensor notifies the controller computer by calling `depart` with the car's travel direction. The traffic controller sets the signal lights: green means go, and red means stop. Construct an algorithm for controlling the lights such that they operate correctly even when most cars approach the tunnel from one direction. In the monitor solution, consider using a `CVar.broadcast` to release all the waiting cars from the north (or south) at once. In which cases, does this make sense?

Solution:

```
enum Direction {North, South};
class Tunnel {
    public:
        Arrive(Direction dir); Depart(Direction dir);
    private:
Lock lock;
        CVar goNorth, goSouth;
        int northWait;    // waiting to go north
        int southWait;    // waiting to go south
        int northBound;   // going north in Tunnel
```

```
        int southBound;   // going south in Tunnel
}
Tunnel::Tunnel() {
    lock.value   = 1;  // lock for shared variables is available
    northWait = 0;     // no one waiting
    southWait = 0;     // no one waiting
    northBound = 0;    // no one in tunnel
    southBound = 0;    // no one in tunnel
}
Tunnel::Arrive(Direction dir){
    lock.Wait();
    // If no one in the tunnel, car goes
    if ((northBound > 0) || (southBound > 0) {
       if (dir == North) {
          // if no one is waiting to go south, car goes north
          if ((southWait > 0) || (southBound > 0)) {
             // otherwise, we count north waiters, and wait for a signal
             northWait++;
             goNorth.Wait(lock);
             northWait--;
          }
       else {
          // if no one is waiting to go north, car goes south
          if ((northWait > 0) || (northBound > 0)) {
             southWait++;
             goSouth.Wait(lock);
             southWait--;
          }
       }
    }
    // count how many cars are in the tunnel
    if (dir == North) northBound++;
    else southBound++;
    lock.Signal();
}


// When a northbound car departs, we signal all southbound cars if any
// are waiting only if no northbound cars are in the tunnel.  Similarly,
// when a southbound car departs.  Note, in the Arrive routine above
// the car only waits if a car going the other direction is in the
// tunnel.

Tunnel::Depart(Direction dir) {

    lock.Wait();
    if (dir == North) {
        northBound--;
        if (southWait > 0) {
            if (northBound == 0)
               goSouth->Broadcast();
        }
    }
    else {
```

```
            southBound--;
            if (northWait > 0) {
                if (southBound == 0)
                   goNorth.Broadcast();
            }
        }
        lock.Signal();
    }
```

3. (10 pts) **Deadlock.** Short answer questions:

   (a) A system has six tape drives (a, b, c, d, e, f), with n threads competing for them. Each thread may need two of the drives. For what values of n is the system deadlock free?

   Solution: One thread. For two threads for example, we can get deadlock with the following:

   Example

```
Thread 1:
   a.Wait();
   b.Wait();
   ...
```

```
Thread 2:
   b.Wait();
   a.Wait();
```

   (b) Can a system be in a state that is neither deadlocked nor safe? If yes, give an example system.

   Solution: Yes. For example, given 3 units of resource A, if thread 1 has 2 units of A and its maximum is 3, and thread 2 has 1 and its maximum is 2. This state is not a safe, but if neither thread ever requests an additional unit of A, then it is not deadlocked.

4. (10 pts) **Deadlock.** Consider the following system snapshot using the data structures in the Banker's algorithm, with resources A, B, C, and D, and processes P_0 to P_4.

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

   Using Banker's algorithm answer the following questions.

   (a) How many resources of type A, B, C, and D are there?

   Solution: (allocation + available) {5,9,9,12} + {3,2,1,0} = {8.11,10,12}

   (b) What is the content of the *Need* matrix?

   Solution: See above table.

(c) Is the system in a safe state? Why?

Solution: Yes, P_0 can finish with its current resources and what's in available. When it finishes, avail becomes $\{6,2,1,2\}$. Now, P_2 can complete and then avail would be: $\{7,5,6,6\}$. Now, P_3 can complete and then avail would be: $\{7,11,9,8\}$. Then either P_1 or P_4 can complete, followed by the other.

(d) If a request from process P_4 arrives for additional resources of (1,2,0,0), can the Banker's algorithm grant the request immediately? Show the new system state, and other criteria.

Solution: No the request cannot be granted because all of none of the process are able to request their max number of resources, i.e., for all processes $i = 0, 4$, need(i) $>$ avail(i).

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