CMPSCI 377: Operating Systems

Solution to homework 2: Scheduling and Synchronization

1. (10 pts) **Scheduling.** Given the following mix of job, job lengths, and arrival times, assume a time slice of 15 and compute the completion and average response time of each job for the FIFO, RR, and SRTF algorithms. Please use the following table format for your solution.

   Solution: Note, to get response times, we must subtract start times from finish times.

			Scheduling Algorithms					
Job	length	arrival time	FIFO	RT	RR	RT	SRTF	RT
0	75	0	75	75	190	190	205	205
1	40	10	115	105	110	100	80	70
2	25	10	140	130	85	75	40	30
3	20	80	160	80	160	80	100	20
4	45	85	205	120	205	120	145	60
		Avg. RT		102		113		77

2. (10 pts) **Scheduling.** Given 3 jobs of length 10, 30, and 20 seconds with the same arrival time, schedule them in job number order. The 10 sec job has 1 sec of I/0 every other sec starting at 1 second (assume the I/O happens just before the time slice). The context switch time is 0 sec, and there are 2 queues. The first has 1 sec time slice; the second has a 2 sec time slice. Using the Multilevel Feedback Queues Algorithm, fill in the following tables with the average response, execution, and completion times of these jobs. Use the notation from class: make the superscript on the job number the progress of the job, and the subscript on the job number the system time. For comparison, also compute the job completion and average response times for the RR algorithm.

   Solution:

Job	length	Completion Time RR	MLFB
1	10	28	28
2	30	60	60
3	20	50	51
avg. RT		46	46.33

Queue	Time Slice	Job
1	1	$1^1_1, 2^1_2, 3^1_3, 1^2_4, 1^3_7, 1^4_{10}, 1^5_{13}, 1^6_{16}$ $1^7_{19}, 1^8_{22}, 1^9_{25}, 1^{10}_{28}$
2	2	$2^3_6, 3^3_9, 2^5_{12}, 3^5_{15}, 2^7_{18}, 3^7_{21}, 2^9_{24}, 3^9_{27}, 2^{11}_{30}, 3^{11}_{32}$ $2^{13}_{32}, 3^{13}_{36}, 2^{15}_{38}, 3^{15}_{40}, 2^{17}_{42}, 3^{17}_{44}, 2^{19}_{46}, 3^{19}_{48}, 2^{21}_{50}, 3^{20}_{51}, 2^{30}_{60}$

3. (5 pts) **Scheduling.** What is the effect on the Round Robin Algorithm of increasing the time slice to arbitrarily large values.

   Solution: Very large time slices decreases Round Robin's ability to improve fairness, and makes it look more and more like FIFO.

4. (10 pts) **Synchronization:** What advantages does the test&set instruction have over enabling and disabling interrupts? In which circumstances may we still perfer enabling and disabling interupts?

   Solution: test&set is less error prone because the OS does not have to remember to enable interrupts, and it does not let the user control interrupts in any way. Inside kernel code short sequences with interrupts disabled will be better since it eliminates busy waiting.

5. (15 pts) **Semaphores.** 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.

   Solution: The solution below enforces alternation if cars are arriving in both directions regularly, and it lets multiple cars going in the same direction in the tunnel at once. It only switches directions, if all cars are out of the tunnel and there are cars waiting to go in the opposite direction. Every time it switches directions (say from north to south), it lets all the waiting (southbound) cars go at once. It checks the northWait/southWait variables to determine if a car is waiting, otherwise if no car is waiting, the arriving car goes. The northBound/southBound variables track the number of cars in the tunnel. In this solution, we only signal to waiting cars. As a result, some cars may skip waiting altogether if no cars are in the tunnel or no car is waiting from the opposite direction.

```
enum Direction {North, South};
class Tunnel {
```

```
    public:
        Arrive(Direction dir); Depart(Direction dir);
    private:
Semaphore mutex, 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() {
    mutex.value   = 1; // mutex for shared variables is available
    goNorth.value = 0; // Cars do not wait if tunnel is empty - Depart
    goSouth.value = 0; //   signals only when car(s) wait (n/sWait > 0)
    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){
    mutex.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++;
             mutex.Signal();   // release mutex before waiting!
             goNorth.Wait();
             mutex.Wait();
             northWait--;
          }
       else {
          // if no one is waiting to go north, car goes south
          if ((northWait > 0) || (northBound > 0)) {
             southWait++;
             mutex.Signal();   // release mutex before waiting!
             goSouth.Wait();
             mutex.Wait();
             southWait--;
          }
       }
    }
    // count how many cars are in the tunnel
    if (dir == North) northBound++;
    else southBound++;
    mutex.Signal();
}

// When a northbound car departs, we signal a southbound car if one is
// 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
```

```
Tunnel::Depart(Direction dir) {

    mutex.Wait();
    if (dir == North) {
        northBound--;
        if (southWait > 0) {
            if (northBound == 0)
               for (int i = 0; i++; i < southWait)
                     goSouth.Signal();
        }
    }
    else {
        southBound--;
        if (northWait > 0) {
            if (southBound == 0)
               for (int i = 0; i++; i < northWait)
                     goNorth.Signal();
        }
    }
    mutex.Signal();
}
```