DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING
CONCORDIA UNIVERSITY

COMP 346: Operating Systems
Fall 2019
Theory Assignment 2
Due date: Wednesday Oct 16th before 11:59 pm

Total marks: 80

Q.1. [15 marks] Consider a single processor, single core environment. Somebody suggested the following solution to the critical section problem involving two processes P_0 and P_1. It uses two shared variables *turn* and *flag*. Note that this is not the Peterson's solution discussed in class, but looks similar:

```
boolean flag [2]; // Initially False
int turn; // Initially 0
do {
      flag[i] = True;  // i == 0 for P0 and 1 for P1
      while (flag[j] == True) { // j = 1- i
           if (turn == j) {
                 flag[i] = False;
                 while (turn == j) ;  // Do nothing: just busy wait
                 flag[i] = True;
           }
      }
      // Critical section code here
      // …
     turn = j;
     flag[i] = False;
    // Remainder section code here
    // …
 } while (True)
```

The above is the code for process P_i, i = 0 or 1. The other process is P_j, where j = 1 – i. Now answer the following questions:

a) Will the solution satisfy mutual exclusion of the critical section? You must prove or argue (in a way similar to we did in class for Peterson's solution) your answer.
b) Will the solution satisfy the "progress" requirement? You must prove or argue your answer.
c) Will the solution satisfy the bounded waiting requirement? If so, what is the bound? You must prove or argue your answer.

Q.2. [20 marks] Answer the following questions:

a) Consider three concurrent processes A, B, and C, synchronized by three semaphores *mutex*, *goB*, and *goC*, which are initialized to 1, 0 and 0 respectively:

```
Process A          Process B          Process C
--------------     -------------      ---------------
wait (mutex)       wait (mutex)       wait (mutex)
…                  …                  …
signal (goB)       wait (goB)         wait (goC)
…                  signal (goC)       …
```

```
signal (mutex)          …          signal (mutex)
                        signal (mutex)
```

Does there exist an execution scenario in which: (i) All three processes block permanently? (ii) Precisely two processes block permanently? (iii) No process blocks permanently? Justify your answers.

b) Now consider a slightly modified example involving two processes:

```
Process A                          Process B
----------                         ----------
for (i = 0; i < m; i++) {          for (i = 0; i < n; i++) {
  wait (mutex);                      wait (mutex);
  …                                  …
  signal (goB);                      wait (goB);
  …                                  …
  signal (mutex);                    signal (mutex);
}                                  }
```

(i) Let $m > n$. In this case, does there exist an execution scenario in which both processes block permanently? Does there exist an execution scenario in which neither process blocks permanently? Explain your answers. (ii) Now, let $m < n$. In this case, does there exist an execution scenario in which both processes block permanently? Does there exist an execution scenario in which neither process blocks permanently? Explain your answers.

Q.3. [15 marks] Consider the below implementations of a semaphore's *wait* and *signal* operations:

```
wait () {
    Disable interrupts;
    sem.value--;
    if (sem.value < 0) {
        save_state (current) ; // current process
        State[current] = Blocked; //A gets blocked
        Enqueue(current, sem.queue);
        current = select_from_ready_queue();
        State[current] = Running;
        restore_state (current);  //B starts running
    }
    Enable interrupts;
}
```

```
signal(){
    Disable interrupts;
    sem.value++;
    if (sem.value <= 0){
          k = Dequeue(sem.queue);
          State[k] = Ready;
          Enqueue (k, ReadyQueue);
      }
     Enable interrupts;
    }
```

a) What are the critical sections inside the *wait* and *signal* operations which are protected by disabling and enabling of interrupts?
b) Give example of a specific execution scenario for the above code leading to inconsistency if the critical sections inside implementation of *wait()* and *signal()* are not protected (by disabling of interrupts).
c) Suppose that process A calling semaphore *wait()* gets blocked and another process B is selected to run (refer to the above code). Since interrupts are enabled only at the completion of the wait operation, will B start executing with the interrupts disabled? Explain your answer.

Q.4. [20 marks] A file is shared between several reader and writer threads. Design a monitor to control the access of the file by the different threads so that the following constraints are satisfied: (i) at most one writer can be active on the file at a particular time. (ii) When a writer is writing to the file, no reader can read from the file. (iii) More than one reader can be reading from the file simultaneously. (iv) When a writer is waiting to write, no more **new** reader should be allowed to read. (v) When a writer is writing and some other writer is waiting to write, then the writer is given more preference over a reader waiting to read. The general structure of each reader and writer thread is shown in the following:

```
Monitor FileControl { // Definition of the monitor to be filled in by you
    …
    …
}

FileControl fc;   // An instance of the monitor
```

```
Writer Thread:                          Reader Thread:
while (True) {                          while (True) {
    ...                                     ...
    fc.WriterEntry();                       fc.ReaderEntry();
    Write (file);                           Read (file);
    fc.WriterExit();                        fc.ReaderExit();
    …                                       …
}                                       }
```

Fill in the pseudo-code for the monitor FileControl as shown above.

Q.5. [10 marks] Someone wrote the following solution to solve the *First Reader Writer Problem*, where multiple readers can access the file at the same time; a writer however has exclusive access to the file (i.e. when a writer accesses the file, no other reader or writer can access the file).

```
semaphore writeBlock = 1, mutex = 1;
int numOfReaders = 0;

//Reader Code – runs in a loop
If  (numOfReaders ==0 ) {
     wait(writeBlock);
     wait(mutex);
     numOfReaders++;
     signal(mutex);
}
else
{    wait (mutex);
     numOfReaders++;
     signal (mutex);
}
read (file, …)
wait (mutex);
numOfReaders--;
if(numOfReaders == 0)
     signal(writeBlock);
```

```
//Writer Code – runs in a loop

wait (writeBlock);
write (file, …)
signal (writeBlock);
```

signal (mutex);

Does this solution suffer any problem(s)? Whether *yes* or *no*, explain your answer very clearly and in full details (preferably with an example scenario of what can go wrong, if any).