COMP 346 - Winter 2017
Theory Assignment 2

Due Date & Submission Format: *Please see course outline.*

Answer all questions

Question # 1

What are the differences between a process and a thread?

Question # 2

What resources are used when a thread is created? How do they differ from those used when a process is created?

Question # 3

What are the differences between user-level and kernel-level threads?

Question # 4

In a swapping/relocation system, the values assigned to the <base, limit> register pair prevent one user process from writing into the address space of another user process. However, these assignment operations are themselves privileged instructions that can only be executed in kernel mode.

Is it conceivable that some operating-system processes might have the entire main memory as their address space? If this is possible, is it necessarily a bad thing? Explain.

Question # 5

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.

Question # 6

Sometimes it is necessary to synchronize two or more processes so that all process must finish their first phase before any of them is allowed to start its second phase. For two processes, we might write:

```
semaphore s1 = 0, s2 = 0;

process P1 {              process P2 {
<phase I>                 <phase I>
V (s1)                    V (s2)
P (s2)                    P (s1)
<phase II>                <phase II>
        }                         }
```

a) Give a solution to the problem for three processes P1, P2, and P3.

b) Give the solution if the following rule is added: after all processes finish their first phase, phase I, they must execute phase II in order of their number; that is P1, then P2 and finally P3.

Question # 7

Explain why both P and V operation must be implemented as a critical section. Demonstrate through an example what may go wrong if any of them is not implemented as a critical section.

Question # 8

What is the potential problem of multiprogramming?

Question # 9

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.