COMP 346 - Winter 2020
Theory Assignment 2

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

Answer all questions

Question # 1

What are the main differences between the user-kernel threads models? Which one of these models is likely to trash the system if used without any constraints?

Question # 2

Why threads are referred to as "light-weight" processes? What resources are used when a thread is created? How do they differ from those used when a process is created?

Question # 3

Does shared memory provide faster or slower interactions between user processes? Under what conditions is shared memory not suitable at all for inter-process communications?

Question # 4

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
-----------
wait  (mutex)
...
signal  (goB)
...
signal  (mutex)
```

```
Process B
-----------
wait  (mutex)
...
wait  (goB)
      signal  (goC)
...
signal  (mutex)
```

```
Process C
-----------
wait  (mutex)
...
wait  (goC)
...
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
--------------
for (i = 0; i < m; i++) {
  wait (mutex);
  …
  signal (goB);
  …
  signal (mutex);
}
```

```
Process B
-------------
for (i = 0; i < n; i++) {
  wait (mutex);
  …
   wait (goB);
  …
   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 # 5

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 # 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

Generally, both P and V operation must be implemented as a critical section. Are there any cases when any of these two operations can safely be implemented as a non-critical section? If yes, demonstrate through an example when/how this can be done without creating any violations. If no, explain why these operations must always be implemented as critical sections.

Question # 8

What is the potential problem of multiprogramming?