Mohammad Ali Zahir

ID: 40077619

COMP 346: Theory Assignment 2

1. User Threads:

   Require no context switching, which makes them the faster model in terms of switching. Since they are only made by system calls, the OS can not see them , which would lead to poor scheduling done by the CPU

   Kernel Threads:

   Kernel threads require context switching, which would mean that it would take more time in terms of switching. Since the kernel oversees the scheduling, the CPU can allocate enough memory for all the processes to run smoothly.

   The only model that can trash the system without any constraints would be the one-to-one relationship model. It will create many kernel threads which would make the CPU unable to allocate memory properly and thus crashing the system.

2.

   Kernel threads are light weight processes. Since they share the memory with the process that creates, we call them light-weight.

A process not only is created by itself, it also creates the PCB. This allows the process to be an independent entity unlike the kernel threads which relies on the process is it derived from. Basically they have to do less work.

3. Shared memory provides a faster interaction between user processes because a shared memory space allows for faster communication from the OS. Shared memory should not be used for inter-processed communications since it might cause synchronization problems, hence making it unsuitable.

4.

a)

i) If we start with process B, it will signal the mutex, goes to process A or C which then will block it completely because it will either keep on doing goB or goC. The same result can be achieved by starting with process C.

ii) By starting at process A, we will go to C, which will lead for A to finish, However, since C will be blocked by waiting for goC, this will also lead for B to wait for the mutex to be released. Hence this will block B and C.

iii) To ensure this, we have to start with A, which then goes to B, which then goes to C.

b)

i) m>n just means that process A will run more than process B. This allows the process to create more permits for the semaphore goB, and since it will always create permits for goB, these processes will never be blocked permanently.

ii) Since process A is not waiting on anything, there is no way it will be blocked permanently, However, once it reaches process B, it will block completely because it will keep on waiting on goB constantly.

5.

A kernel always exists in the memory. This means that a single memory is never bearer of the size of the whole address. Addresses are also reserved for the system sometimes. This would mean that no process can be the sole inhabitant of the memory space. By going through with this, we can cause major problems not allowing the system to have its reserved space, which can cause major problems.

6.

a) Process P1{

<phase 1>

signal(s1)

wait(s2)

<phase 2>

}

Process P2{

<phase 1>
signal(s2)
wait(s3)
<phase 2>
}

Process P3{
<phase 1>
signal(s3)
wait(s1)
<phase 2>
}

b)

Process 1{

<phase 1>

wait(s3)

signal(s1)

<phase 2>

}

Process P2{

<phase 1>

wait(s1)

signal(s2)

<phase 2>

}

Process P3{

<phase 1>

wait(s2)

signal(s3)

<phase 2>

}

7. We will irregularities wit the threads if we do not maintain the operation as atomic.

   Ex:

   Let s(4) = 3

   Let p1 executes signal(), but suddenly it gets switched to wait() by p2. By the time we reach to completing signal() for p1, s(4) =3 will be rendered false.

8. In multiprogramming, there is a lot of overflows of the connected hardware. If two people try to connect to a different device with their Bluetooth devices but both hardware are capable to be paired with each separate devices. With Bluetooth speakers to a computer, multiprogramming may cause unnecessary data exchange from connecting to the wrong computer and hence creating unnecessary problems.