18.1

Processes and threads are related to each other but are fundamentally different.

A process can be thought of as an instance of a program in execution. Each process is an independent entity to which system resources (CPU time, memory, etc.) are allocated and each process is executed in a separate address space. One process cannot access the variables and data structures of another process. If you wish to access another process’ resources, inter-process communications have to be used such as pipes, files, sockets etc.

A thread uses the same stack space of a process. A process can have multiple threads. A key difference between processes and threads is that multiple threads share parts of their state. Typically, one allows multiple threads to read and write the same memory (no processes can directly access the memory of another process) However, each thread still has its own registers and its own stack, but other threads can read and write the stack memory.

A thread is a particular execution path of a process; when one thread modifies a process resource, the change is immediately visible to sibling threads.

18.2

A context switch is the time spent switching between two processes (e.g., bringing a waiting process into execution and sending an executing process into waiting/terminated state). This happens in multitasking. The operating system must bring the state information of waiting processes into memory and save the state information of the running process.

In order to solve this problem, we would like to record timestamps of the last and first instruction of the swapping processes. The context switching time would be the difference in the timestamps between the two processes.

Let’s take an easy example: Assume there are only two processes, P1 and P2 P1 is executing and P2 is waiting for execution. At some point, the OS must swap P1 and P2—let’s assume it happens at the Nth instruction of P1. So, the context switch time for this would be Time\_Stamp(P2\_1) – Time\_Stamp(P1\_N).

One approximation could be to record the end instruction timestamp of a process and start timestamp of a process and waiting time in queue If the total time of execution of all the processes was T, then the context switch time = T – (SUM for all processes (waiting time + execution time)).

18.5

(i)

Semaphore s\_a(0);

Semaphore s\_b(0);

A

{

/\*\*\*/

s\_a.release(1);

}

B

{

s\_a.acquire(1);

/\*\*\*/

s\_b.release(1);

}

C

{

s\_b.acquire(1);

/\*\*\*/

}

(ii)

Semaphore s\_a(0);

Semaphore s\_b(0);

Semaphore s\_c(1);

A

{

s\_c.acquire(1);

/\*\*\*/

s\_a.release(1);

}

B

{

s\_a.acquire(1);

/\*\*\*/

s\_b.release(1);

}

C

{

s\_b.acquire(1);

/\*\*\*/

s\_c.release(1);

}

18.6

Java provides two ways to achieve synchronization: synchronized method and synchronized statement.

Synchronized method: Methods of a class which need to be synchronized are declared with “synchronized” keyword. If one thread is executing a synchronized method, all other threads which want to execute any of the synchronized methods on the same objects get blocked.

Synchronized statement: It provides the synchronization for a group of statements rather than a method as a whole. It needs to provide the object on which these synchronized statements will be applied, unlike in a synchronized method.

i) If you have two threads in one instance of a program, can they call A at the same time?

No.

ii) Can they call A and C at the same time?

Yes. Only methods of the same object which are declared with the keyword synchronized can’t be interleaved.