1. What is the difference between an operating system and middleware?

An operating system is a software that uses the hardware resources of a computer system to provide support for the execution of other software. Middleware is software occupying a middle position between application programs and operating systems. Operating systems and middleware both are software used to support other software. However, they rely upon different underlying providers of lower-level services. An operating system provides the services in its API by making use of the features supported by the hardware. Middleware, on the other hand, provides the services in its API by making use of the features supported by an underlying operating system.

2. What is the relationship between threads and processes?

A thread is the fundamental unit of concurrency. Any one sequence of programmed actions is a thread. A process is a container that holds the thread or threads that you started running and protects them from unwanted interactions with other unrelated threads running on the same computer.

3. Of all the topics previewed in chapter one of the text book, which one are you most looking forward to learning more about? Why?

Of all the topics previewed in chapter one, the one we are most looking forward to learning about is the scheduling problem. The author said there are many ways to solve the problem of scheduling threads' execution. We want to know how the different scheduling approaches fit different contexts of system usage and achieve differing goals.

4. Suppose thread A goes through a loop 100 times, each time performing one disk I/O operation, taking 10 milliseconds, and then some computation, taking 1 millisecond. While each 10-millisecond disk operation is in progress, thread A cannot make any use of the processor. Thread B runs for 1 second, purely in the processor, with no I/O. One millisecond of processor time is spent each time the processor switches threads; other than this switching cost, there is no problem with the processor working on thread B during one of thread A's I/O operations. (The processor and disk drive do not contend for memory access bandwidth, for example.)

a. Suppose the processor and disk work purely on thread A until its completion, and then the processor switches to thread B and runs all of that thread. What will the total elapsed time be?

Each time thread A takes 10ms + 1ms = 11ms. It goes through a 1oop 100 times, so it takes a total of 11 * 100 = 1100ms. Switching takes 1ms. Thread B takes 1sec = 1000ms. The total elapsed time will be 1100ms + 1ms + 1000ms = <u>2101ms</u>.

b. Suppose the processor starts out working on thread A, but every time thread A

performs a disk operation, the processor switches to B during the operation and then back to A upon the disk operation's completion. What will the total elapsed time be?

Threads A & B are running at the same time. Working on thread B can be done while thread A is going through the loop 100 times. The time it takes is 10ms + 1ms(switch A to B) + 1ms(switch B to A) + 1ms(thread A computation) = 13ms. The total elapsed time will be 13 * 100 = 1300ms.

c. In your opinion, which do you think is more efficient, and why?

It is more efficient to make use of the processor by switching to and running thread B while the disk operation is in progress. Working only on thread A until its completion and switching to and running thread B are much slower than running the threads concurrently. It is a waste of time to not run them concurrently when thread B can make use of the processor during thread A's I/O operation.