Homework #1

Professor Johnson

Claudia Gusti & Cash Cassidy

Questions:

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

Answer: An operating system is the software that interacts directly with the hardware. Middleware acts as a middle man between application programs and the operating system. Both middleware and the operating system contain rules that dictate when computer processes should interact with each other through persistent storage.The difference is that an operating system provides its services by making use of the features of the hardware, but the middleware provides its services by making use of the underlying operating system.

2. What is the relationship between threads and processes?

Answer: A thread is the fundamental unit of concurrency. It refers to any one sequence of programmed actions. A system typically will be running several threads, one for each ongoing program execution. A program is a container that holds the thread/threads that were started running and protects them from unwanted interactions with unrelated threads running on the same computer.

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

   Answer: We are looking forward to learning more about multi-threading because we want to use the processor as optimally as possible to run our programs more efficiently.

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?

Thread A complete 100 loops -> 100 * (10+1) = 1100 ms

Context Switch -> 1 ms

Thread B completes -> 10 * 100 ms = 10000ms

Total -> 1000ms + 1 ms + 1100 ms = 2101 ms = 2.101 seconds

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?

Thread A goes into I/O, and immediately context switch happens for thread B -> 1 ms

Thread B runs until I/O is complete -> 10 ms

Context switch happens from B to A -> 1 ms

Thread A does the computation -> 1 ms

The process loops for 100 times

Total time -> 100*(1+10+1+1) = 1.300 seconds

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

The second option is better, because running concurrent threads takes less time (1.3 seconds vs 2.1 seconds) and maximizes the resources of the computer’s processor and disk space.

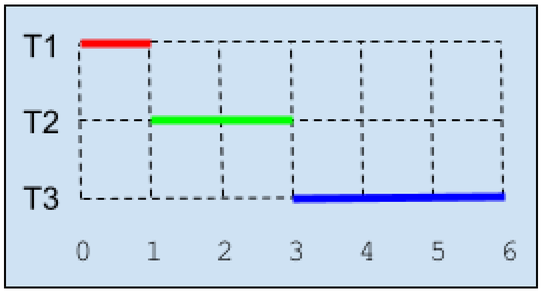
5. Find and read the documentation for `pthread_cancel[]`. Then, using your C programming environment, use the information and the model provided in Figure 2.4 on page 26 of the text book to write a program in which the initial [main] thread creates a second thread. The main thread should sit on a read call

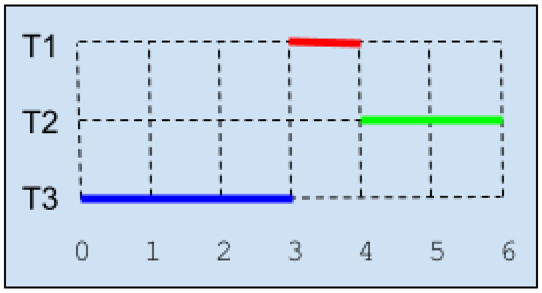
of some kind, waiting to read input from the keyboard, waiting until the user presses the Enter key. At that point, it should kill off the second thread and print out a message reporting that it has done so. Meanwhile, the second thread should be in an infinite loop; during each iteration it must sleep five seconds and then print out a message. Try running your program.

- Can the sleeping thread print its periodic messages while the main thread is waiting for keyboard input?
    - Yes, it can
- Can the main thread read input, kill the sleeping thread, and print a message while the sleeping thread is in the early part of one of its five-second sleeps?
    - Yes, it can

6. Suppose a system has three threads [T1, T2, and T3] that are all available to run at time 0 and need one, two, and three seconds of processing, respectively. Suppose each thread is run to completion before starting another. Draw six different Gantt charts, one for each possible order the threads can be run in. For each chart, compute the turnaround time of each thread; that is, the time elapsed from when it was *ready [time 0]* until it is completed [finished]. Also, compute the *average* turnaround time *for each order*. Which order has the shortest average turnaround time? What is the name of the scheduling policy that produces this order? {You can look this up in the textbook.]

Charts are numbered [1] [4]





[2] [5]
[3] [6]

Turnaround Times
Chart 1: T1 – 1s, T2 – 3s, T3 – 6s, Average – 3.33s
Chart 2: T1 – 1s, T2 – 6s, T3 – 4s, Average – 3.67s
Chart 3: T1 – 3s, T2 – 2s, T3 – 6s, Average – 3.67s
Chart 4: T1 – 4s, T2 – 6s, T3 – 3s, Average – 4.33s
Chart 5: T1 – 6s, T2 – 2s, T3 – 5s, Average – 4.33s
Chart 6: T1 – 6s, T2 – 5s, T3 – 3s, Average – 4.67s

Chart 1 has the lowest average turnaround time. The order of chart 1 is produced by the Shortest Job First scheduling policy.

7. Performa an Internet search of the C standard library API and find out how to get information from the command line first by using a **`printf[]`** call to display a prompt, then another function call [which you will look up] to get the user input. Write a program in C to prompt the user for their demographic information including name, age, class year, and any three other data items you wish. Structure the program as a call-and-response application such that each data item is a single question with a single answer entry. When all data has been obtained, display the data on the console. Each data item must be on a separate

line, and it must be appropriately labeled. The output must be done using a **single** `printf[]` statement.