HOMEWORK 2

CSE 120 OPERATING SYSTEMS
FALL 2006

DUE ON **10/17/2006 (TUESDAY)** AT THE BEGINNING OF CLASS

1) What is the difference between Multitasking, Multithreading and Multiprogramming?

2) What would be a possible problem if you executed the following program? How can you solve it?

```
#include <signal.h>
#include <sys/wait.h>

main()
{
        for(;;)
        {
                if( !fork() )
                {
                        exit(0);
                }
                sleep(1);
        }
}
```

3) Does exec() return? What happens if the command passed to exec() doesn't exist? Does exec() create a new process?

4) "The separation of fork() and exec() is probably the most brilliant innovation in the design of Unix." Justify this statement. Why do we need fork() and exec() as separate system calls? Why not a single one that does both?

5) If one thread in a program calls fork(), does the new process duplicate all threads, or is the new process single-threaded? If a thread invokes exec(), will the program specified in the parameter to exec() replace the entire process – including ALL the threads?

6) [Silberschatz Galvin Gagne – 7th Edition – Chapter 6 – Problem 6.16] How does the signal() operation associated with monitors differ from the corresponding operation defined for semaphores?

7) Illustrate with an example how the implementation of a semaphore with a waiting queue may result in a deadlock. Another problem related to deadlocks is indefinite blocking or starvation. What is indefinite blocking or starvation and how might this be caused?

8) Operating systems often distinguish between two kinds of semaphores – 'binary' and 'counting'. What scenarios do you think each of them is most suitable for?

9) Consider the swap() procedure given below:

```
void Swap(boolean *a, boolean *b)
{
        boolean temp=*a;
        *a=*b;
        *b=temp;
}
```

Assuming that this swap() procedure is executed atomically as a single instruction, write a piece of code that demonstrates how this can be used to provide mutual exclusion that satisfies the bounded-waiting requirement. Use the following skeleton code:

```
while(TRUE)
{

        /* your code */

                //critical section

        /* your code */

                //remainder section

}
```

(Hint: Use common data structures initialized to FALSE, like:

```
        boolean waiting[n];
        boolean lock;
```

)

10) [Silberschatz Galvin Gagne – 7th Edition – Chapter 6 – Problem 6.10] Show that, if the wait() and signal() semaphore operations are not executed atomically, then mutual exclusion may be violated.