Name: _____ **Student ID #:** _____

Lab #5 Solutions: COMP 3000B (Operating Systems)
February 7th & 9th, 2006

1. (2 pts) From a graphical terminal (e.g. gnome-terminal), use the "ssh desktopX" command to log in to another machine in the lab, where desktopX is the name of another machine in the lab. Is the DISPLAY environment variable set? Can you run graphical programs (such as gnome-system-monitor) from that remote machine and have it display locally?

   Ans. By running "echo $DISPLAY" or the "env" command, it should have been clear that the DISPLAY environment variable was not set. As a result, it wasn't possible to run remote graphical programs, they would terminate with a "can't open display" error.

2. (2 pts) Run "ssh -X desktopX xclock", where desktopX again is the name of another machine in the lab. Which machine's clock is xclock checking—the local clock, or that of the remote machine? Explain.

   Ans. By adding the -X flag to the ssh command, the DISPLAY environment variable is set to a valid value, allowing remote X applications to be run and displayed locally. (SSH actually routes all graphical requests from the remote system through the encrypted connection, allowing remote applications to be executed securely.)

   Thus, the command as given will run xclock on the remote host. Because the xterm process is running on the remote host, the clock value it shows will be that of the remote host (i.e. the time displayed will be from the kernel on the remote host). (Note that you can test this fact by examining the list of active processes on the local and remote hosts.)

3. (2 pts) Run the command "xterm". Then, inspect the list of currently running processes using the command "ps axo pid,uid,gid,cmd". What group is xterm running as, and what group are your other processes running as? (You can convert the numeric IDs to text using the file /etc/group.) Why are they different? Explain.

   Ans. xterm should have been running as group utmp (#43), while (most) other student processes should have been running as group student (#1000). These groups are different because the xterm binary has the set group ID (setgid) bit set in its permissions; thus, when xterm is run via the execve system call, the kernel will give the process the effective group ID of the binary (in this case, 43); without the setuid/setgid mechanism, the effective group ID of all processes started by the user student is equal to the student account's default group ID of student (gid 1000).

   The setuid and setgid mechanisms are to allow users to invoke programs that have more privileges than they do. Thus, an xterm process invoked by the student user can change the contents of the file /var/run/utmp even though the student user normally cannot modify this file.

   (Side note: the file /var/run/utmp stores a record of who is currently logged into the system. This file is consulted by commands such as "who" and "w".)

4. (4 pts) Create five terminal windows. In three of them, run the command "worms". In the fourth, run top. In the fifth, use the command "renice" to change the priority of the worms processes. Close down any other running programs that you have started.

   Do the three worm processes execute at the same time when they run at full priority? What happens when you lower their priority to 20? (Be sure to lower them one at a time.) How is CPU time being divided between running processes? Experiment and explain your findings.

   Ans. This question was designed to show how priorities can affect CPU time allocation. On some systems, the worms will all appear to execute roughly at the same time; on others, the worms will take turns filling a terminal with characters. By lowing the priority of different worm processes, though, their relative speeds will change (with lower priority worms executing more slowly). When all the worms have the same (lowered) priority again, they will again also execute at roughly the same speed. The relative speed of the worms programs indicates the fraction of CPU time that each is receiving.

   To truly understand what was happening, it is important to note a few things. First, text I/O is buffered—in fact, there is effectively a bounded-buffer producer/consumer relationship between the terminal program (gnome-terminal or xterm) and the worms program. Also, the rate of I/O for the terminal is controlled by the producer/consumer relationship between the terminal program and the X display server (XFree86). Thus, the actual patterns observed depend upon the relative rate of execution of all of these programs, and on how the kernel decides to dynamically divide CPU time between these I/O-bound processes. Because the video cards, CPUs, RAM, and motherboards are different in the different machines in the lab, the concurrent worms programs do not execute in the same manner on every machine.