CSCI 631 **Computer Assignment 1** **Spring 2015**
(30 points)

Daytime Client–Server Model

The source codes of a TCP daytime client and server programs are given in your textbook. After compiling your source files of your programs and linking the resulting object files with the system routines, execute your programs. Modify your programs so that they will use the port number supplied on the command line instead of using the registered port number for daytime service.

The usage of the server is: server [port-number], where port-number is the port number of the server that runs in the background. The port number is optional, and if it's not supplied, then the default port number SERV_PORT (as defined in the header file 631.h in directory /home/courses/631/common) is chosen. A valid (ephemeral) port number should be between 9880 and 9890, but you don't need to validate a port number in your program. That is the job for the shell scripts. The client also accepts an optional port number with the default value SERV_PORT. Its usage is: client server-address [port-number].

For error checking, use the I/O routines such as fprintf ( ) or perror ( ), and remember that all error messages are supposed to be printed out on stderr. For error handling, use wrapper functions for all system calls that you use in your programs, as explained in your textbook.

Name the source file of your client as prog1_client.c, the source file of your server as prog1_server.c, and the source file of your wrapper functions as wrapper.c (however, you can give them any names you like as far as they have proper extensions; you may also need header files for prog1_client, prog1_server.c, and wrapper.c). When your programs are ready, login both the **tiger** (1st machine) and **lambda** (2nd machine) on separate windows, and do the followings:

1. On the 2nd machine, make a link to the script P1_2: ln –s ~courses/631/bin/P1_2 and start the script utility: script prog5_2.out. When you are in the script utility, execute the command: P1_2 prog1_server.exe, which will execute your server in the background. (If your executable file has a different name, then use it as an argument to P1_2.) If you don't supply an argument to P1_2, the script will prompt for that. When the script asks for a port number for the server on the 2nd machine, enter a port number from the available pool of port numbers or just enter for the default port number. That port number will be displayed on the screen to be used for the client on the 1st machine. Then, the script will ask you to enter from the keyboard to continue, but do not enter anything at this point.

2. On the 1st machine, make a link to the script P1_1: ln –s ~courses/631/bin/P1_1 and start the script utility: script prog1_1.out. When you are in the script utility, execute: P1_1 prog1_client.exe, which will execute your client in the foreground (If your executable files have different names, then use them as arguments to P1_1.) If you don't supply the arguments to P1_1, the script will prompt for that. When the script asks for a port number, you must enter the same port number that you used in the 2nd machine. After displaying the current time and date on the screen, the script will

ask you to enter from the keyboard to continue, but do not enter anything at this point.

3. To resume the operation on the 2nd machine, just hit the enter key from the keyboard. Then the script P1_2 will display all active processes on the system, and after terminating the server program, it will display all active processes one more time. Terminate the script utility on the 2nd machine by the control character <ctrl>–D.

4. To resume the operation on the 1st machine, just hit the enter key from the keyboard. Then the script P1_1 will list all active processes on the system. Terminate the script utility on the 1st machine by the control character <ctrl>–D.

5. Just before logout from either the 1st or the 2nd machine, execute the command: /bin/ps –f –u $USER. This will display a list of all processes generated by your programs – some of those processes might even be created in one of your previous terminal sessions. In that list, if you see any processes except your login shells, terminate all those processes by the kill command: kill pid1 pid2 ... pidn, where pid1 pid2 ... pidn is the list of process ids of the processes to terminate. If some of those processes are still alive after executing the kill command, then try to terminate their father processes.

The correct outputs for this program can be found in files:

~courses/631/progs/p1/prog1_1.out (1st machine) and
~courses/631/progs/p1/prog1_2.out (2nd machine).

You need to include proper documentation in all files you submit for grading. For the client, you need wrappers for the following system calls: close ( ), connect ( ), fputs ( ), inet_pton ( ), read ( ), and socket ( ). For the server, you need wrappers for the following system calls: accept ( ), bind ( ), close ( ), connect ( ), fputs ( ), inet_pton ( ), listen ( ), read ( ), socket ( ), time ( ), and write ( ).

If the inet_pton ( ) function flags an error by returning an integer value of <= 0, the return value -1 sets errno but not the return value 0. Thus, for the return value 0, do not use the function perror ( ) to print out the error message. Instead, use the I/O function fprintf ( ) on stderr without using the function strerror ( ).

Since a socket is a file in the UNIX system, do not forget to close it at the end of your program, and remember that the close ( ) function is a system call and it needs a wrapper.

When your programs are ready, submit the files of your client program on the 1st machine and the files of your server program on the 2nd machine to your instructor by the shell script mail_prog.631.