COP 4635 SYS & NET II – TCP CLIENT-SERVER COMMUNICATION

OVERVIEW

This assignment will delve into simple client-server network programming and network design. You will create a total of three programs that will communicate with each other over a network. A server program will be designed to accept multiple requests from clients and simultaneously respond to those requests. Client programs will make requests of the server process and display the returned results. You are required to implement two different client programs, one in C and one in Java to test client-server communication.

THE PROGRAM

When completed, your server program (`TCPserver.c`) will perform the following tasks:

1. Create a server socket and display the needed information about the socket. That information should include the host name and IP address of the machine where the server program is running and the *port number* used to create the socket.

2. Wait for incoming requests from client programs. When a request arrives, the server will perform the following steps:
   a) Create a new detached thread to handle the request.
   b) Interpret the request that was sent by parsing the message.
   c) Depending on the message received, perform the following actions:
   [1] If the message starts with the tag *<echo>* ends with the tag </echo>, the child thread will read the text message between the two tags and return it as a reply in a new message that starts with the tag *<reply>* followed by the received text message and ends with the tag *</reply>*. There should be no additional whitespaces introduced by the server between *<reply>* and *</reply>* and the text message received by the client unless the original message from the client includes whitespace. For example, when the client sends the message *<echo>Hello, World!</echo>* the server must respond with *<reply>Hello, World!</reply>*.
   [2] If the message is identical with *<loadavg/>*, the child thread will compute its load average for 1, 5, and 15 minutes using the library function `getloadavg()`. Values returned by the function represent the number of processes in the system run queue averaged over the last 1, 5, 15 minutes. The three numbers must be represented as a string with each value separated by a ":". The complete message returned by the server should start with the tag *<replyLoadAvg>* and end with the tag *</replyLoadAvg>*. In the return message, there should be no whitespaces between the tags *<replyLoadAvg>* and *</replyLoadAvg>* and the actual encoded values for the load average.
   [3] In all other cases, when the message is not formatted as described above, the server responds with the error message *<error>unknown format </error>*.
   [4] The child thread will close the socket connection to the client and terminate.

3. The main (parent) thread will continually wait for requests until `CTRL-C` is pressed to terminate the server process. You may or may not capture the CTRL-C keys to shutdown the server.

The server must handle messages according to the protocol above in a robust manner. This means that the server needs to respond to the requests correctly and handle incorrectly formatted messages gracefully without crashing or failing to respond. You may assume that the total size of the message exchanged between server and client exceeds no more than 256 bytes.

When completed, your client programs (`TCPclient.c` and `TCPclient.java`) provide functions to perform the following tasks:

1. Create a streaming socket and connect to a server.
2. Send a request for service to the server.
3. Receive a response from the server.
4. Close the socket.

Use the provided header and stub files uploaded to the Content area in *eLearning* to implement the functionality of the clients. For C, write the testing code in a separate main program `TCPmain.c`. Because the tester will use a separate Java and C main program to evaluate your solution, it is important that you comply with the interface definitions of the clients as provided by the header and Java stub files. A sample main program in C and Java is provided. Your `TCPmain.c` program in C and the main() method in `TCPclient.java` should execute the following sequence of steps:

1. Accept a hostname and a port number from the command line in the format "`<hostname> <portnum>`" where `<hostname>` is the name of a host, and `<portnum>` is a port number.
2. Create a socket and connect to the server at the specified location.
3. Send a request to the server (user entered or a fixed message).
4. Receive a reply from the server and display it in its entire length.
5. Close the socket and terminate the program.

The client must call the corresponding function implemented in `TCPclient.c` to connect, send, receive messages, and close the socket.

IMPLEMENTATION SUGGESTIONS

I suggest reading back through the lecture notes for details on each step in completing this project. You should also consider examining the reading material and the sample program posted on *eLearning*. Both sources could save you a lot of time. For your implementation you need to use the following system calls for the C client-server program and the corresponding library calls for the Java client program:

socket()	**gethostbyname()**	**gethostname()**	**bind()**
listen()	**accept()**	**close()**	**getloadavg()**
pthread_create()	**write() or send()**	**read() or recv()**	
pthread_detach()	**getsockname()**	**phthread_exit()**	

DELIVERABLES & EVALUATION

Submit your complete solution as a single zip file (only zip files will be accepted) containing A) source code, B) a single makefile to compile the client and server programs, and C) a README file (if applicable) to the corresponding dropbox in *eLearning*.

The README file should only be included if you submit a partial solution. In that case, the README file must describe the work you did complete. You must follow the *Project Submission Instructions* posted in *eLearning*

under Course Materials. The submission requirements are part of the grading for this assignment. If you do not follow the requirements, 5 points will be deducted from your project grade. Keep in mind that documentation of source code is an essential part of programming. If you do not include comments in your source code, points will be deducted. We also require you to refactor your code to make it more manageable and to avoid memory leaks. Points will be deducted if you don't refactor your code or if we encounter memory leaks in your program.

Your solution needs to compile and run in the computing environment provided on the CS department's Linux servers. Testers will upload your solution to a server and compile and test it running several undisclosed test cases. Therefore, to receive full credit for your work it is highly recommend that you test & evaluate your solution on the servers to make sure that graders will be able to run your programs and test them. You may use any of the 5 SSH servers available to you for programming and testing and evaluation. Use *ssh.cs.uwf.edu* to log into any of the 5 servers.

GRADING

This project is worth 100 points in total. A grade sheet posted in *eLearning* outlines the distribution of the points and grading criteria. Keep in mind that there will be a substantial deduction if your code does not compile.

DUE DATE

The project is due on February 1st by 11:00 pm. Late submissions will not be accepted and the grader will not accept any emailed solutions. The syllabus contains details in regards to late submissions. Groups that submit this project by January 25th at 11:00 pm will receive 6 bonus points. However, any group that chooses to submit their solutions before January 25th at 11:00 pm cannot resubmit and updated version at a later time.