CS3251 Computer Networks I

Fall 2010

Programming Assignment 1

Assigned: Sept. 14, 2010

Due: Sept. 23, 2010, 11:59pm

TCP and UDP Applications Programming

For this assignment you will design and write your own application program using network sockets. You will implement two different versions of the application. One version will be based on TCP. The other will use UDP. You will write both the client and server portions of this application.

The application you are designing is a four-function reverse polish notation (or postfix) calculator. For more information about RPN see http://en.wikipedia.org/wiki/Reverse_Polish_notation

Your program should be able to 1) read arbitrary postfix expressions from the client command line, 2) have the client transmit the expression to a server where the computation will be performed, 3) have the server return the answer to the client and 3) have the client print the answer to the screen. The client will not do any calculation but will instead send a request to the server to be calculated. Also, **the server can only perform a single operation with each request message.**

You should support addition, subtraction, multiplication and division. You can limit the application to support only integer (of arbitrary value) input and output. You may impose (nontrivial) limits on the size of the integer and/or the number of input "tokens" (operands or operations).

You must allow the IP address and port number of the server to be specified on the command line of the client. For instance, the command:

```
rpncalc 127.0.0.1 13001 "245 549 +"
```

would produce the output:

```
794
```

This query would have been sent to the server application running on the local host (loopback address). The only output is the resulting value or an error message if it fails.

Another example is:

```
rpncalc 127.0.0.1 5600 "25 5 * 60 +"
```

would produce the output:

In this case, the client will send two separate queries to the server. The first will calculate 25 * 5. The second will calculate 125 + 60. Only the final result should be printed. (For the TCP implementation, only a single connection should be used.)

You will need to develop your own "protocol" for the communication between the client and the server. While you should use TCP or UDP to transfer messages back and forth, you must determine exactly what messages will be sent, what they mean and when they will be sent (syntax, semantics and timing). Be sure to document your protocol completely in the program writeup.

Your server application should listen for requests from *rpncalc* clients, process the client and return the result. After handling a client the server should then listen for more requests. You are NOT required to implement a concurrent server for this assignment. The server application should be designed so that it will never exit as a result of a client user action.

Focus on limited functionality that is fully implemented. Practice defensive programming as you parse the data arriving from the other end. Do not work on a fancy GUI, but focus on the protocol and data exchange. Make sure that you deal gracefully with whatever you or the TAs might throw at it.

You are welcome (encouraged even) to use external sources as you develop the RPN algorithm (such as the one shown in the Wikipedia reference above). You may also use client and server templates including the ones discussed in class. However, **you must include a citation (text or web site) in your source code and project description if you use an external reference or existing code templates.**

Notes:

1. You are implementing two complete, separate versions of the program. A client and server using TCP and another client and server using UDP. Your final submission should include 4 separate programs. For simplicity in grading, let's call them *RPNcl-tcp*, RPN*srv-tcp*, *RPNcl-udp* and RPN*srv-udp*.

2. Your TCP and UDP versions will share much of the same code for command line parsing and user interaction. However, your UDP implementation will have to deal with lost request messages. Make sure you consider what happens when a message is lost. You will need to handle this in some way such as using a timer to retransmit your request as well as a reasonable limit on the number of retransmissions. A good way to test your client in this situation is to run it without the server. Does your client handle this gracefully?

3. Your programs are to be written in standard C using the standard Socket libraries we have been covering in class. Your programs should run on the CoC Unix (or Linux) systems.

4. You should select a port for your *RPN* service. I recommend something between say 13000 and 14000. You will need to make sure nobody else in the class is running their server on the same system using the same port as you. (Consider using the value 13000 + your prism account number). *Make sure you kill your server when you are done working.* See the *netstat* command for checking on busy network ports.

5. Use explicit IP addresses in the client for specifying which server to contact. Do not use DNS for host name lookups. We will do that in a later assignment.

6. Make sure that you do sufficient error handling such that a user can't crash your server. For instance, what will you do if a user provides invalid input?

7. You must test your program and convince us it works! Provide us with a sample output (you can use *script* or cut-and-paste) that shows all of your functions working.

Submission Format:

Submit your programming assignments as a gzipped tarball (.tar.gz extension) with your Last Name as the file name. You submit everything using t-square.

Your archive(.tar.gz) should include: your **well-documented source code**, a README file, a Makefile and sample output file called Sample.txt. The README file must contain:

1. Detailed instructions for compiling and running your client and server programs including a Makefile.

2. A description of your application protocol (1/2 to 1 page) with sufficient detail such that somebody else could implement your protocol

3. Any known bugs, limitations of your design or program

Grading Guidelines:

The grade will be split equally between the UDP and the TCP programs. We will use the following guidelines in grading:

0% - Code is not submitted or code submitted shows no attempt to program the functions required for this assignment.

25% - Code is well-documented and shows genuine attempt to program required functions but does not compile properly. The protocol documentation is adequate.

50% - Code is well-documented and shows genuine attempt to program required functions. The protocol documentation is complete. The code does compile properly but fails to run properly (crashes, or does not handle properly formatted input or does not give the correct output).

75% - Code is well-documented. The protocol documentation is complete. The code compiles and runs correctly with properly formatted input. But the program fails to behave gracefully when there are user-input errors.

100% - Code is well-documented. The protocol documentation is complete. The code compiles and runs correctly with properly formatted input. The program is totally resilient to input or network errors.