CMPE 156/L, Winter 2019

Programming Assignment 2
Due: _____

The purpose of this project is to develop the server and client sides of a simple remote shell application. The user of the client application can type various Unix shell commands, which execute on the server and return results back to the client. The command results need not be stateful (e.g., cd /tmp, pwd). The client will then display the results on *stdout*, as if the commands were run on the local machine.

Requirements

- The server code must accept its listening port number as the first command line argument (from argv[1]). For example, you should be able to run the server using the command line

  ```
  ./myserver 1234
  ```

- The server does **not** need to be concurrent (that is, it is not required to serve more than one client at a time). If you design it as concurrent, then it should not leave any zombie processes after servicing clients. The client need not accept any keyboard input from the user while it is processing a command, that is, until it has received the results for the previous command from the server and displayed them.
- The client code must accept the server IP address and listening port number as the first and second command line arguments (i.e., argv[1], argv[2]). For example, you should be able to run the client using the command line

  ```
  ./myclient 127.0.0.1 1234
  ```

- The client should terminate when the user types the special command *exit*.
- It is up to you to design the format of the messages exchanged between the client and server. The client is not required to do any formatting or conversions, and can display the information received from the server as such.
- The server design must be robust against errors and network failures. The server should be able to recover gracefully from client crashes, sends bad commands, etc. Someone should not be able to stop or crash your server by sending an invalid request, or by stopping/killing the client in the middle of the session. You are not required to protect the client from a misbehaving server, however.
- The memory use of the server should not get larger every time it processes a request (that is, no memory leaks).

What to submit?

You must submit all the files in a single compressed tar file (with `tar.gz` extension). The files should include

1. A specification of the application-layer protocol, describing the handshakes involved, message formats, error handling, etc. (Note: you need to describe only the handshakes above the socket layer, not the handshakes within the TCP layer).
2. A Makefile that can be used to build the client and server executable. Source files under the 'src' directory, executable under the 'bin'.
3. A README file including your name and a list of files in the submission with a brief description of each. If your code does not work completely, explain what works and what doesn't or has not been tested.
4. Documentation of your design in plain text or pdf. Do not include any Microsoft Word files. The documentation should describe how to use your application, including the internal design of your client and server implementations.
5. Organize the files into directories (*src*, *bin*, *doc*, etc.)

Grading

Each submission will be tested to make sure it works properly and can deal with errors. Grades are allocated using the following guidelines:

Basic Functionality:	25%
Basic Testing	25%
Dealing with Errors	20%
Documentation	20%
Style/Code structure, etc.	10%

Note that 20% of the grade will be based on how well your code deals with errors. Good practices include checking all system calls for errors and avoiding unsafe situations such as a buffer overflow.

The files must be submitted before midnight on the due date above.

Honor Code

All the code must be developed independently. All the work submitted must be your own.

Client implementation

The client program should show the prompt "client $ ", where there is a single space before and after the dollar sign. Don't add color or any other special characters to the prompt. For example,

```
./myclient 127.0.0.1 1234
client $ date
```

It is strongly recommended to use the readline library in the client program to process user input. Lookup usage of readline() and the header file #include <readline/readline.h>. Entering control-C would terminate the client program (but not the server).

Also, the client program should not modify the output returned by the server program. If a command fails on the server, the output on the client to should be clear about the error.

Example of commands

The server should be able to execute these commands (not necessarily piping commands):

```
cat /etc/resolv.conf
cat <filename>
date
df -h
du
echo $PATH
echo <environment variable>
free
last
ps aux
pwd
set
tty
uname -a
uptime
who

exit
```