Com S 352 Fall 2018 Project 2:

SIMPLE SECURE REMOTE SHELL

100 points

Due: Friday, November 16, 2018, 11:59pm

This project is extension to Project 1, where you are required to add support for additional features.

So, this project has all the functionalities of Project 1, while adding the following new features to it.

A. **Encryption/Decryption of Command:** Client Encrypts the Command entered by the user before sending it to the Server. Server, upon receiving the encrypted command, decrypts it before executing the command.

You are required to write the Encryption and Decryption functions for the following simple algorithm:

Encryption:

Add a number "N" to the ASCII values of all the characters of the command to get the Encrypted command

Decryption:

Subtract the number "N" from the ASCII values of all the characters of the command to get the original command

Where,

N……is the last digit of your ISU ID (mention it in the code using comments) (If the last digit of your ISU ID is 0, use the number before that)

B. **"jobs" command:** You are to implement the jobs command, so that when the command "jobs" is entered by user, it displays the list of the process IDs (PIDs) of all "live" child processes (i.e. the processes that have not terminated yet)

C. **Reaping of Zombie Processes:** Every time, when input is received at server, you should check for any Zombie processes, and if found, reap them. You can use the list of child processes PIDs to check for zombie processes.

D. "History" Command:

a) You are to implement a history feature in your program. When the user enters the command "History", it displays up* to 10 most recent commands entered by the user. The commands are numbered from 1 to 10, 1 being the oldest and 10 being the most recent command.
* if the user has so far entered less than 10 commands, say 6, display all the 6 commands).

Next your program should support two techniques for retrieving commands from the command history:

b) When the user enters **!!** as a command, the most recent command in the history (i.e. the command just before entering !!) is executed.
c) When the user enters **!N** as a command, the Nth command in the history is executed, (a single ! followed by an integer N, e.g. !3 means 3rd command in history) (where N = 1,2,3,……,10)
d) The program should also manage basic error handling.
1. If there are no commands in the history, entering !! should result in a message "No commands in history".
2. If there is no command corresponding to the number entered with !N, the program should output "No such command in history."

E. Handling Multiple commands

In actual Linux shell multiple commands can be entered in the same line separated by "**;**".

In Your Program:

The user (on client side) should be able to enter multiple commands in the same line separated by **;** , the client sends all the commands to the server, receives output for these commands from the server, and displays the outputs on its screen.

Example, if the user wants to enter 3 commands in the same line,

-bash-4.3$ date;cal;who

with the following **output** (outputs of the three commands):

```
Wed Sep 13 12:55:51 CDT 2017

   September 2017
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9 10 11
12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27
28 29 30

abi      pts/0    2018-01-28 13:24 (10.64.222.89)
abi      pts/1    2018-01-28 13:31 (10.64.222.89)
mabdulah pts/2    2018-01-28 14:39 (10.25.71.69)
jcu      pts/8    2018-01-19 17:14 (10.24.46.63)
```

NOTE: The basic working of the project remains the same with "jobs", "History", and "multiple commands" features. Meaning that all commands are entered on client side, client sends the commands (including jobs, history, and the multiple commands entered by the user) to the server, the server executes the commands and sends the outputs back to client, and then the client displays the output on its screen.

You will write the Server and Client program (as this project is extension of project 1, some steps are the same):

Server: The server will run on the remote machine.

- It will bind to a **TCP** socket at a port known to the client and waits for a Connection Request from Client.
- When it receives a connection, it forks a child process to handle this connection. The Server must handle multiple clients at a time.
- The parent process loops back to wait for more connections.
- The command received is **Decrypted**.
- The child process executes the given shell command (received from the client), returning all **stdout** and **stderr** to the client. (Hence, the server will **not** display the output of the executed command)
- The server can assume that the shell command does not use **stdin**.

Client: The client will run on the local machine.

- From the command line, the user will specify the host (where the server resides) and the command to be executed.
- The client will then connect to the server via a TCP socket.
- The Client **Encrypts** the Command entered by the user.
- The Client sends the command to the server.
- The client will display any output received from the server to the **stdout**.
- After displaying the output, the client waits for next command from the user.
- The client will **not** close/exit until the user enters "quit" command.

Submission

- You should use C to develop the code.
- You need to turn in electronically by submitting a zip file named: Firstname_Lastname_Project2.zip.
- **Source code must include proper documentation to receive full credit (you will lose 10% of your score, if the code is not well documented).**
- All projects require the use of a **make file** or a certain script file (accompanying with a **readme file** to specify how to use the script/make file to compile), such that the grader will be able to compile/build your executable by simply typing "make" or some simple command that you specify in your readme file.
- **Source code must compile and run correctly on the department machine "pyrite", which will be used by the TA for grading. If your program compiles, but does not run correctly on pyrite, you will lose 15% of your score. If your program doesn't compile at all on pyrite, you will lose 50% of your score (only for this issue/error, points will be deducted separately for other errors, if any).**
- You are responsible for thoroughly testing and debugging your code. The TA may try to break your code by subjecting it to bizarre test cases.
- You can have multiple submissions, but the TA will grade only the last one.

Start as early as possible!