Client Server Interactive shell Linux

Architecture

To start with, there is a server that can connect multiple clients. This connection between a client and server is a bi-directional communication channel. This means the client can send and retrieve data to the server similarly the server can send and retrieve data from a client.

To enable this communication sockets are used. Each time the server is run it takes control of a port which the clients must use to connect.

Once a client has connected to the server it prompts the user to input a command. There can be multiple commands in a single line separated by the delimiter ‘;’. Once the command is read it is written on the socket file descriptor aka msgsock. The server processes the command and returns the relevant result by writing on the same msgsock.

In the client code it uses two threads to get the job done. One thread to read input from the user and write on the msgsock, another thread to read msgsock and write the result on the screen. To terminate the connection the client needs to press only the enter key without any input.

On the server side of code when a connection has occurred it should wait to read from the msgsock, in other words it would have been blocked until client has sent some command.

In order to make the server interactive i.e. it can send commands when not accepting any, threads were used. To enable the server to read input from the user a separate thread was created once a connection is accepted. After doing this the client handler part of the server did not execute properly as it did not respond to client commands. In order to solve this a new thread was made inside the client handler part which would wait to read from the msgsock. Without this thread the msgsock was not being read.

This solves makes the commands print, send and list work fine. To execute command of listof, a communication medium was needed between the input thread and client handler part of the server. This made it possible to get list from individual clients.

Input thread processes print send and list commands itself and outputs the result. The command of listof requires the client handler. Once getting the input of listof 1234, the input thread searches for the required client in the cient table and then writes the command on the pipe of that client. The client handler thread read this pipe and prints the list of the programs that client has run.

The client handler also uses signals to help keep track of how a child was terminated. When using the kill command SIGTERM signal is sent to gracefully terminate the process. The status is changed to terminated. It also deals with external termination that is when the spawned process id terminated using the cross button. The signal handler is made for this pupose. The signal handler will catch SIGCHLD that is when a child is terminated the signal handler is called which then updates the status and end time of the process. Wait api is called to avoid zombie processes. It picks up the pcb of the terminated child to avoid memory wastage.

The code includes two structures that help maintain record of clients and processes. Both the structure instances are made global to be easily accessible anywhere in the code.

Limitations

- The server is not parallel.

- When multiple instances of the same program are running and we kill it by name, only the first instance is closed.
- If a process is not killed the elapsed time showed is incorrect.
- Only an single instance of the server can be run. I think this is a limitation as we can use another instance to keep backup. I have fixed the port number to try port forwarding for the demo.
- When we give an invalid program to run the exec fails but list is updated. I tried to fix this but started causing problem with other parts of the code.
- The msgsock gets corrupted on client side when changes are made in the run command part.
- The division functionality is a bit buggy when it encounters characters.
- Faced an unusual problem once only where the server exited upon recieving a command from user(input thread). Tried to encounter it again but couldnt.