Lab2

Last modified: 10/17/2022

Try to implement the same problems as for lab 1 in the situation the server needs to be concurrent – i.e. be able to accept multiple clients at the same time.
Also try to solve the following problems in the same context. What approach would be the best in the following scenarios? (Threads, processes)

1. The client takes a string from the command line and sends it to the server. The server interprets the string as a command with its parameters. It executes the command and returns the standard output and the exit code to the client.
2. The client sends the complete path to a file. The server returns back the length of the file and its content in the case the file exists. When the file does not exist the server returns a length of -1 and no content. The client will store the content in a file with the same name as the input file with the suffix *–copy* appended (ex: for f.txt => f.txt-copy).
3. The server chooses a random float number <SRF>. Run multiple clients. Each client chooses a random float number <CRF> and send it to the server. When the server does not receive any incoming connection for at least 10 seconds it chooses the client that has *guessed* the best approximation (is closest) for its own number and sends it back the message *"You have the best guess with an error of <SRV>-<CRF>"*. It also sends to each other client the string *"You lost !".* The server closes all connections after this.
4. The clients send an integer number N and an array of N float values. The server will merge sort the numbers received from all clients until it gets an empty array of floats (N=0). The server returns to each client the size of the merge-sorted array followed by the merge-sorted arrays of all floats from all clients.
5. The client sends a domain name taken from the command line (Ex: www.google.com) to the server. The server opens a TCP connection to the IP address corresponding to the received domain name on port 80 (called HTTP-Srv). It sends on the TCP connection the string: *"GET / HTTP/1.0\n\n"* and relays the answer back to the client. When *HTTP-Srv* closes connection to the server, the server closes the connection to the client at its turn.
6. The server chooses a random integer number. Each client generates a random integer number and send it to the server. The server answers with the message "larger" if the client has sent a smaller

number than the server's choice, or with message "smaller" if the client has send a larger number than the server's choice. Each client continues generating a different random number (larger or smaller than the previous) according to the server's indication. When a client guesses the server choice – the server sends back to the winner the message "You win – within x tries". It also sends back to all other clients the message "You lost – after y retries!" (x and y are the number of tries of each respective client). The server closes all connections upon a win and it chooses a different random integer for the next game (set of clients)

7. The client reads a username and a password from the standard input. It sends the username to the server. The server uses the *getpwent* system call repeatedly to find the password information about the username. If the entry for the username is found, the password field from the *struct passwd* is returned to the client. The client recovers the salt of the password and checks the input password with the received encrypted version using the *crypt* system call. If there is no user *username*, the server returns back to the client the empty string and closes the connection.
8. Change the Python Example Concurrent Number Guess Bellow to transmit with each answer to a client – the number of total clients that are competing. Also change the client to read the numbers from the standard input and launch a contest in the class during the lab

Example – Concurrent Client-Server (C/C++)– Character Count

Example – Python Concurrent Number Guess – threaded

Note: Run the server and the client on different hosts. For this you need to know the IP address of the machine where the server runs (ifconfig/ipconfig)