Assignment 2 (TCP and UDP Client-Server Programming)

Due: January 20, 2023, 2 pm

1. Write a UDP client-server system to make the time server you did in Assignment 1. However, the client should wait a maximum of 3 seconds for the result. If the time is not received within 3 seconds, the client tries again. The client tries a maximum of 5 times, after which if it still cannot get the time, it exits with an error message "Timeout exceeded". Use the poll() function to wait to receive on the UDP socket with timeout. Submit the 2 files *timeserv.c* and *timeclient.c*.

2. Your task will be to write two programs - one program corresponding to the server process and another program corresponding to the client process. The client and the server interacts using a communication protocol as follows.
   1. The client establishes a connection to the server using the connect() call.
   2. The server sends the string "LOGIN:" to the client
   3. The client displays the string to the user.
   4. The user enters a username (a string with max size 25) from the keyboard
   5. The client reads the username and sends the null-terminated string to the server
   6. The server checks the username against a list of usernames present in a file named *users.txt* in the server (assume in the current directory where the server is run from). The file has a list of usernames, one username per line. If the given username occurs in the fie, the server sends the string "FOUND" to the client, else it sends the string "NOT-FOUND" to the client.
   7. If the server has sent "NOT-FOUND", the client prints an error message saying "Invalid username", closes the connection, and exits. If the server has sent "FOUND", the client does the following steps.
   8. The client asks the user for a shell command to be executed on the server side.
   9. The user enters a command from the keyboard. The client reads the command and sends it to the server side as a null-terminated string for execution.
   10. The server receives the command and executes it. It sends the results of the execution to the client. If the command is invalid, the server sends the special string "$$$$". If the command is valid but there is an error in running it (for example a "cd" command run with a directory that does not exist), the server returns the special string "####".
   11. The client displays the result to the user and prompts the user for the next command. For an invalid command, this will just be a string "Invalid command". For a valid command with error in running, this will be the string "Error in running command".
   12. This continues until the user types the special command "exit", at which point the client closes the connection and exits.

For simplicity the set of valid shell commands accepted by the server (along with the corresponding C library function that will give you the result) are "pwd" (getcwd), "dir" (opendir + readdir), "cd" (chdir) only. You need not implement any options of the commands available in a standard Linux system. Do not use the "system" command to run the shell commands.

You need to ensure the following in your code:

1. The server should be a concurrent TCP server.
2. No buffer of size > 50 can be used in either client or server.
3. You should send only the bytes for the strings specified, the commands (and their arguments if any), and the results and not send any extra bytes in either direction.
4. All commands, results etc. should be sent as null-terminated strings only.

Submit 2 files: *sh_server.c* and *sh_client.c*