Assignment - 2

Blocking vs Non-blocking Sockets

Problem Description

The protocol between the client and the server is as follows:

1. The client connects to the server and then asks the user for input. The user enters a simple arithmetic expression string in infix form (e.g., "1 + 2", "(5 + 6) * 8.9"). The user's input is sent to the server via the connected socket.

2. The server reads the user's input from the client socket, converts it into a postfix expression and then evaluates the postfix expression, and sends the result and the converted postfix expression back to the client as well as writes the following in a file named “server_records.txt” (at the beginning create an empty file).
   <client_id> <infix_expr> <converted_expr><answer> <time_elapsed>
   Note: The time elapsed is the time elapsed (in seconds) from when the server connected to this client to returning the response to this query.

3. The client should display the server's reply to the user, and prompt the user for the next input until the user terminates the client program.

At the very minimum, the server should be able to handle addition, multiplication, subtraction, division, modulus, and power operations.

Here the server should be able to simultaneously chat with (i.e. serve requests for) multiple clients. Since “accept” is a blocking system call, it is not possible to wait for other connections while serving the requests for another. This has to be handled in 2 ways:

Problem 1: Using the “fork” system call

Write two separate C programs, one for a TCP server (handles requests for multiple users) and another one for a client.
The server program will “fork” a process for every new client it receives, such that each child process is responsible for handling the request of one client.

Problem 2: Using the “select” system call

Write two separate C programs, one for a TCP server (handles requests for multiple users) and another one for a client.
The server uses the “select” call to see which clients are making the requests (it can be a connection request or read/write request) and serve the requests of multiple clients.

Sample inputs:

User types: 1 + 2, server replies 1 2 +, 3
User types: 2 * 3, server replies 2 3 *, 6
User types: 4 + 7 - 3, server replies 4 7 + 3 -, 8
User types: 30 / 1.0, server replies 30 1.0 /, 30.0

Below is a sample run of the client.

```
$ gcc client.c -o client
$ ./client
Connected to server
Please enter the message to the server: 22 + 44
Server replied: 22 44 +, 66
Please enter the message to the server: 3 * 4
Server replied: 3 4 *, 12
...
```

In parallel, here is how the **sample run of the server** looks like this (you may choose to print more or less debug information).

```
$ gcc server.c -o server
$ ./server
Connected with client socket number 4
Client socket 4 sent message: 22 + 44
Sending reply: 22 44 +, 66
Client socket 4 sent message: 3 * 4
Sending reply: 3 4 *, 12
…
```

Algorithm for converting infix expression to postfix

1. Scan the infix expression from left to right.
2. If the scanned character is an operand, output it.
3. Else,
1 If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a '('), push it.
2 Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)
4. If the scanned character is an '(', push it to the stack.
5. If the scanned character is an ')', pop the stack and output it until a '(' is encountered, and discard both the parenthesis.
6. Repeat steps 2-6 until infix expression is scanned.
7. Print the output
8. Pop and output from the stack until it is not empty.

Algorithm for evaluating postfix expression

1) Create a stack to store operands (or values).
2) Scan the given expression and do the following for every scanned element.
    a) If the element is a number, push it into the stack
    b) If the element is an operator, pop operands for the operator from the stack. Evaluate the operator and push the result back to the stack
3) When the expression is ended, the number in the stack is the final answer.

Marking Scheme

Handling concurrency:

1. Using “fork”: 20 marks
2. Using “select”: 20 marks

Conversion from infix to postfix: 15 marks
Evaluation of postfix expression: 15 marks
Contents of “server_records.txt”: 10 marks
Error handling strategies: 10 marks
Coding style - 10 Marks.

Total: 100 marks

Submission Deadline: January 20, 2021 [2 PM]

Submission Guidelines

1. This is an individual assignment.
2. Create a folder where the folder name is “<roll_number>_Assgn2”.
3. Create two subfolders under the folder - problem1 and problem2.
4. Compress the parent folder (whose name is your roll number), upload the compressed folder at Moodle.

Reference Links

1- Blocking vs Non-blocking call
https://www.scottklement.com/rpg/socktut/nonblocking.html

2- Select system call
http://www.tutorialspoint.com/unix_system_calls/_newselect.htm

3- Beej Guide
https://beej.us/guide/bgnet/pdf/bgnet_usl_c_1.pdf