Practical assignments - Socket programming

Note: You can use any programming language for this assignment. I recommend a scripting language like Python since it comes with a lot of useful functions built-in("batteries included"). In addition, Python is particularly useful to know since there are several useful computer security tools written in Python and thus require working knowledge of Python. Be sure to document in a README file which programming language you have used.

Introduction

On completing this assignment, you will become familiar with writing simple clients and servers using the socket interface of the operating system. We will be working with the socket interface provided by the Linux OS.

Question 1: Writing a TCP client

Writing a client is easier than a server to some extent so we'll start with that. In this assignment, you will write a TCP client that will answer a challenge given by the server. Here's a sample session demonstrating the challenge and the corresponding response:

Server(S) - "Welcome to challenge. Timeout: 3s, total trials: 50"
Server(S) - "Start: W, end: Z, period: 2"
Client(C) - "W Y"

i.e. the challenge is given a start value S_VAL, an end value E_VAL and a period P_VAL, return the set of values within [S_VAL, E_VAL] each P_VAL apart, starting from S_VAL. See if you can understand this problem statement with the help of the above example session before you proceed.

However, you must write a program that returns the challenge response because within 3 seconds, the server will terminate the connection if no data is received:

S - Welcome to challenge. Timeout: 3s, total trials: 50
S - Start: e, end: p, period: 9
(after 3 seconds; client sends no data to server)
S - Boo! You're too slow! Bye!
<connection terminated>

How we will test your program

1. You are required to create a run.sh file which will compile any source code if needed, invoke your challenge solver and pass arguments to the solver.

2. When we invoke the run.sh file, we will pass the IP address and TCP port number of the challenge server as arguments:

   ```
   $ ./run.sh 192.168.56.1 8000
   ```

3. Only the data written to the socket will be evaluated. Any output written to stdout and stderr by client will be ignored by autograder.

Sample test cases

In order to help make it easier for you to test your challenge solver, we provide a test server and few tests cases in folder “1_helpers” in the attachment. The following instructions will help you run the test server to evaluate your client.

1. Run the python test server for this question with the desired TCP port number and sample test cases file’s name as argument.
2. The server will automatically start listening for connections on the TCP port you specified above.
3. Once the test server starts running, you can connect to it by using “nc” command to connect to the port you specified on the host “localhost”. This will help you play around with server to understand the challenge better if needed.
4. After writing your challenge solver, you can test it using this test server as well.

Note that the actual grader will use a different set of test cases during evaluation and not just these samples test cases. These sample test cases and the test server help simulate the environment where your client will be tested so that you can be sure you implemented the client properly.

IMPORTANT: The server reads 1 line at a time and processes it. For this server, a line is defined as any sequence of characters ending with “\n”. So every time you send the challenge’s solution to server, it should end with “\n” else the server will not read your solution correctly and timeout thinking no data was sent.

Question 2: TCP server programming

In this assignment, you will implement a multi-forking server that can handle at most 5 clients at a time. The clients will request the Nth Fibonacci number multiple times and the server must provide the correct answer for every request to the client which submitted the request. If the client wishes to terminate, it will request the 0th Fibonacci number. A client can send any number of requests before terminating. A sample session is shown below:

```
C - 5
S - 3
C - 7
```

S - 8
C - 0
S - (exits)

We will ask your server to generate very high fibonacci numbers which may not fit into 4 byte data types(eg: int). If you use a compiled language like C/C++/Java, use some bignum library to avoid errors. Python and most other similar scripting languages should be able to handle large numbers easily. Additionally, make sure your program is fast because clients will disconnect after 5 seconds if no data is received.

How we will test your program

1. You are required to create a run.sh file which will compile any source code if needed, invoke your challenge solver and pass arguments to the solver.
2. We will pass a port number as argument to the server program you write. The server must listen for incoming connections on that port. Sample invocation below need not be the actual invocation - we may use a different port number for evaluating.

   ```
   $ ./run.sh 8000
   ```

3. We will then run multiple clients, each of whom will request a random Fibonacci number random number of times. The clients might sleep for a random duration between requests so wait for client to reply.
4. Remember to accept at most 5 clients after which server must wait till a client disconnects before accepting new connection.
5. The server must respond correctly for every incoming request else the client will terminate and you will lose grades.
6. Finally, we will automatically kill the server after the tests are completed.

TIP: To save time, you can save the numbers already computed in an associative array(eg: map data structure).

Sample test cases

In order to help make it easier for you to test your challenge solver, we provide a test client and few tests cases in folder “2_helpers” in the attachment. The following instructions will help you run the test client to evaluate your server.

1. Run your solver program(preferably using run.sh to make sure the shell script is working correctly).
2. Run the python test client for this question with the host address and port number of the server as well as sample test cases file’s name as arguments.
3. The client will automatically send test cases to the server and display the result.

Note that the actual grader will use a different set of test cases during evaluation and not just these samples test cases. These sample test cases and the test server help simulate the environment where your client will be tested so that you can be sure you implemented the client properly.

<u>IMPORTANT</u>: The client reads 1 line at a time and processes it. For this client too, a line is defined as any sequence of characters ending with “\n”. So every time you send the challenge’s solution to the client, it should end with “\n” else the client may not process it and timeout.