PA5: The Server Moved Out!

Introduction

In this programming assignment, you will be adding a class called `TCPRequestChannel` to extend the IPC capabilities of the client-server implementation in PA3 using the TCP/IP protocol. The client-side and server-side ends of a `TCPRequestChannel` will reside on different machines.

Since the communication API (not just the underlying functionality and features) through TCP/IP is different from FIFO, you will also need to restructure the server.cpp and client.cpp as part of this programming assignment:

- The server program must be modified to handle incoming requests across the network request channels using the TCP/IP protocol. Specifically, the server must be able to handle multiple request channels from the client residing on a different machine.

- You must also modify the client to send requests across the network request channels using the TCP/IP protocol.

The Assignment

In this assignment, you will restructure your PA3 `client.cpp` in order to accomplish data point and file transfers using TCP/IP request channels. You will also rewrite your `server.cpp` so that multiple instances of the client program can connect to the server simultaneously.

The client will take in two additional mandatory arguments compared to PA3.

- ***"-a"*** which denotes the IP address the server is running on.
- ***"-r"*** which denotes the port number the server is deploying its services to.

The following command line options are valid for PA5:

- For data point transfers
  - *a, r* [with optional *n, w, b, p, h, m* arguments]

```
./client -n <# data items> -w <# workers> -b <BoundedBuffer size> -p
<# patients> -h <# histogram threads> -m <buffer capacity> -a <IP
address> -r <port no>
```

- For file transfers
  - *a, r, f* [with optional *w, b, m* arguments]

```
./client -w <# workers> -b <BoundedBuffer size> -m <buffer capacity>
-f <filename> -a <IP address> -r <port no>
```

The server must be called in the following form:

```
./server -r <port no> -m <buffer capacity>
```

Note that *"-m"* is an optional argument whereas *"-r"* is a mandatory argument for the server. Also note that the server no longer runs using the `exec()` function from the client. Rather, it is run separately from a terminal on a different machine. The server executable does not need the ip address argument because it runs the service in the current host. It just needs to deploy its services on the user-assigned port. The client on the other hand needs to know where the server is running (i.e., IP address) and at which port number.

Implementation Note

A class has been added called `TCPRequestChannel` which will replace `FIFORequestChannel` from PA3. The following matches the class declaration (comments explaining functions in starter code):

```
class TCPRequestChannel {
private:
      int sockfd;

public:
      TCPRequestChannel (const std::string _ip_address,
                         const std::string _port_no);
      TCPRequestChannel (int _sockfd);

      ~TCPRequestChannel ();

      int accept_conn ();

      int cread (void* msgbuf, int msgsize);
      int cwrite (void* msgbuf, int msgsize);
};
```

You should provide definitions for the functions declared above in a **`TCPRequestChannel`**`.cpp` file. In addition, you will need to make changes in the `server.cpp` and the `client.cpp` programs to implement the desired functionality for realizing TCP/IP connection. For instance, the server should run an infinite loop to `accept()` incoming connections from the client and create a new thread with the accepted socket to process subsequent communication using the `handle_process_loop` function (which should stay mostly unchanged except for the input argument type - it should now be `TCPRequestChannel*` instead of `FIFORequestChannel*`).

The client on the other hand, can simply call the respective constructor ***`w`*** times to create all the channels. **There is no need for sending a `NEWCHANNEL_MSG`** to the server because calling the constructor results in a separate dedicated channel. Due to this reason, **the client does not need a separate control channel** either.

- Create `TCPRequestChannel.h/.cpp` from template code above.
- Fulfill requirements of network socket connections in `TCPRequestChannel`
- Modify `client.cpp/server.cpp`

1. Server creates `TCPRequestChannel(ip_address="", port_no=r)`
2. Server enters infinite loop to establish connections with `TCPRequestChannel::accept_conn()` and `TCPRequestChannel::TCPRequestChannel(int sockfd)`
3. Client creates channels with `TCPRequestChannel(ip_address=a, port_no=r)`

Getting Started

This assignment will be hosted using the GitHub Classroom platform. You must visit https://classroom.github.com/a/idAEcOyB to create a repository for your project. You must have a functioning PA3 as a baseline for this project. Talk to your TA if your PA3 code is non-functional.

Rubric Instructions:

- **Code Requirements [85 points]**
  - **[15 points]** Datapoint requests into histograms over sockets
  - **[25 points]** File transfers with CSV and binary files and varying message capacity over sockets
  - **[20 points]** Remote network communication over sockets
    a. NOTE: Not tested by autograder
  - **[25 points]** Secret tests
  - View the *PA5 FAQ* for methods of testing over a remote connection

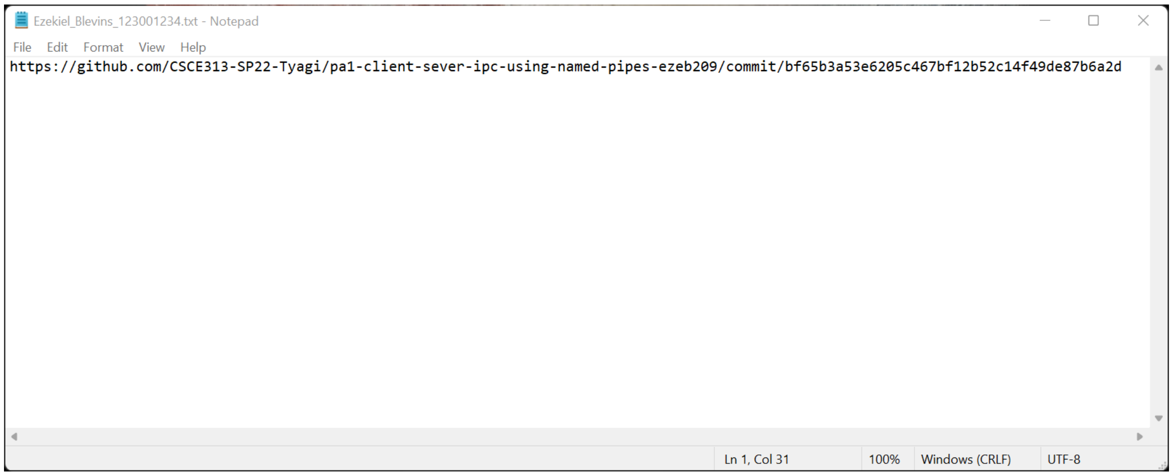
- **Report [15 points]**
  - **Design (½-1 page):** Briefly describe your implementation design as it pertains to setting up the client and server to communicate over the network.
  - **Data Requests**: Make at least two graphs for the performance of your client program with varying numbers of worker threads **w** (try [50, 100]), varying size of request buffer **b** (try [256, 2048]), varying size of histogram threads **h** (try [5, 50]) for patients **p** = 15 and ecg data count **n** = 15K. Discuss how performance changes (or fails to change) with each of them, and offer explanations for both. Do we see scaling on any of the parameters?
  - **File Request**: Make two graphs for the performance of your client program

with varying numbers of worker threads **w** (try [50, 100]) and varying buffer capacity **m** (try [256, 2048]). Discuss how performance changes (or fails to change) with each of them, and offer explanations for both. Do you see a scaling? Why or why not?

- **Insight:** Share any unique insights that you gained while setting up the communication parameters. Document any new features/findings that you sought/implemented while working on this assignment. Key discretionary contributions may qualify for a small subjective credit from your teaching staff.
- **Questions to Consider:**
    a. How does the TCP/IP method differ from FIFO in terms of speed?
    b. What is the maximum number of connections you can create without changing the ulimit parameter? Is the number the same as in PA3?
    c. What happens to the point of diminishing return? Does it change from what you saw in PA3?

What to Turn In

Once you have completed the assignment and submitted your final code to GitHub Classroom, on Canvas, turn in a text document named *FirstName_LastName_UIN.txt* containing the link to the commit hash that you are submitting (PA1 example below) and your report named *PA5_FirstName_LastName_UIN.pdf*.


Ezekiel_Blevins_123001234.txt - Notepad
File Edit Format View Help
Ln 1, Col 31
100%
Windows (CRLF)
UTF-8

```
https://github.com/CSCE313-SP22-Tyagi/pa1-client-sever-ipc-using-named-pipes-ezeb209/commit/bf65b3a53e6205c467bf12b52c14f49de87b6a2d
```