Matthew Buchanan

Networks and Data Communication

CS - 447

Programming Assignment 1

Introduction

The goal of this project is to create a client/server pair using UDP. This pair needs many things UDP inherently lacks, like reliable data delivery, connections (each client on a separate dedicated socket), and of course it must be multi-threaded. The server acts as an interactive calculator with only three functions. Different users may log into the server at any time or choose to leave.

Design Decisions

The language of choice was python 3, since I'm new to network programming and didn't want to face any issues. I faced issues.

My initial design was to create a list of client sockets, and use those to communicate with the threads. Since python's sockets are not thread safe, I settled on a new plan.

The main server thread would listen for new clients on its port. Once one was detected, a thread is created for the new client, and that's the last the "main" thread hears of it.

The thread calls a function which retrieves a new port for establishing a dedicated communication channel with the client. Code on the client changes the server port to this new port provided by the server, effectively giving it sole access.

The thread listens on its port, calling a request handler to handle user input and server output. When the client signals it wishes to close, the thread is ended, severing the connection and freeing the port.

Design Specifics

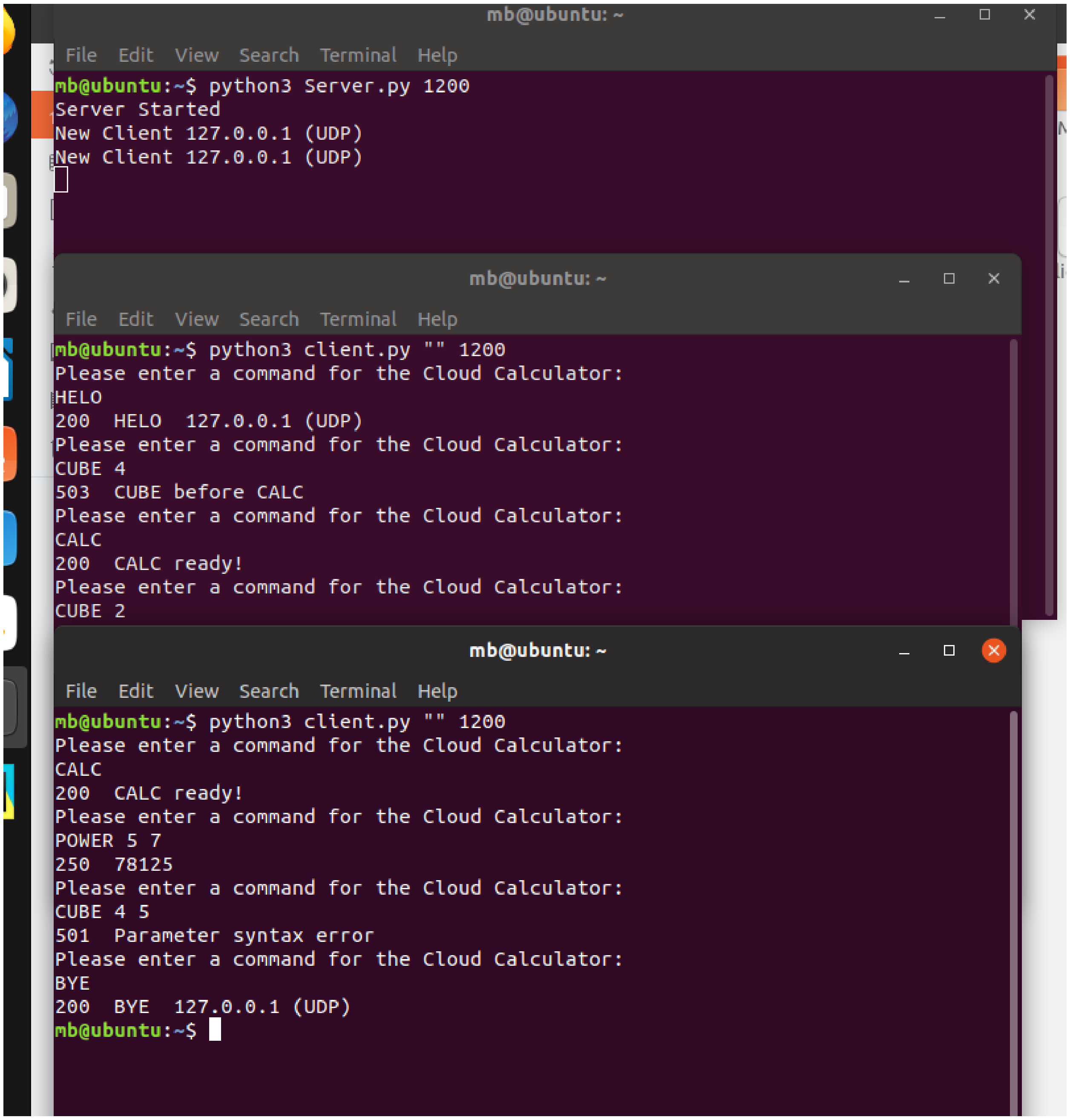
- The UDP was required
- Multithreading was implemented with python's threading library
- Each client receives its own thread for the duration of the session
- Each client receives its own port on the server
- All socket transfers use a three step process similar to TCP. One host sends some data, and the other sends the data back. Once this data is verified, a short accept code is transmitted to the receiving host.

Design Compromises

- No time outs were implemented, meaning severed communication results in a hung client and a server with a ghost client occupying resources
- No mechanism was implemented to end a thread from outside the thread itself, such as for timed out clients.

Test Sample Output

A screen of the program showing two clients connected to a server


mb@ubuntu: ~
File Edit View Search Terminal Help

```
mb@ubuntu:~$ python3 Server.py 1200
Server Started
New Client 127.0.0.1 (UDP)
New Client 127.0.0.1 (UDP)
```

mb@ubuntu: ~
File Edit View Search Terminal Help

```
mb@ubuntu:~$ python3 client.py "" 1200
Please enter a command for the Cloud Calculator:
HELO
200  HELO  127.0.0.1 (UDP)
Please enter a command for the Cloud Calculator:
CUBE 4
503  CUBE before CALC
Please enter a command for the Cloud Calculator:
CALC
200  CALC ready!
Please enter a command for the Cloud Calculator:
CUBE 2
```

```
mb@ubuntu:~$ python3 client.py "" 1200
Please enter a command for the Cloud Calculator:
CALC
200  CALC ready!
Please enter a command for the Cloud Calculator:
POWER 5 7
250  78125
Please enter a command for the Cloud Calculator:
CUBE 4 5
501  Parameter syntax error
Please enter a command for the Cloud Calculator:
BYE
200  BYE  127.0.0.1 (UDP)
mb@ubuntu:~$
```

Summary

I learned a lot writing this program, and its final implementation leaves much to be desired. Even though I’m several hours past due date, I feel good with how my first attempt at network programming worked out.