COMPSCI 677 Spring 2022

Lab 1: Toy Store - Output

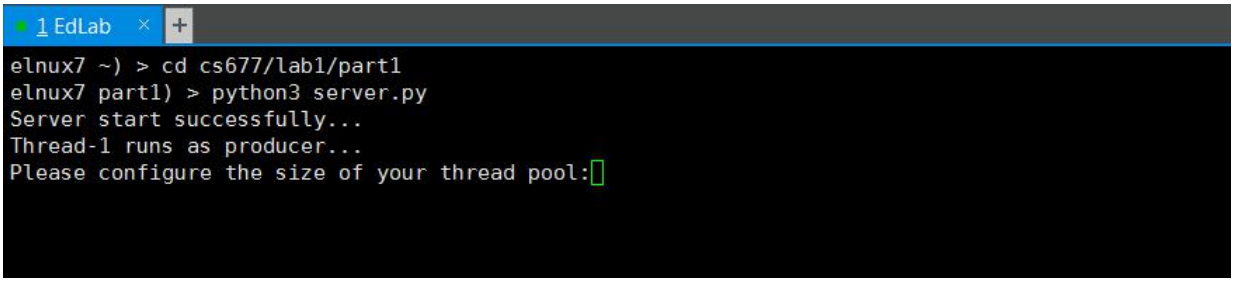
Team Members: Maoqin Zhu, Yixiang Zhang

Part 1 - Implementation with Socket Connection and Handwritten Thread Pool

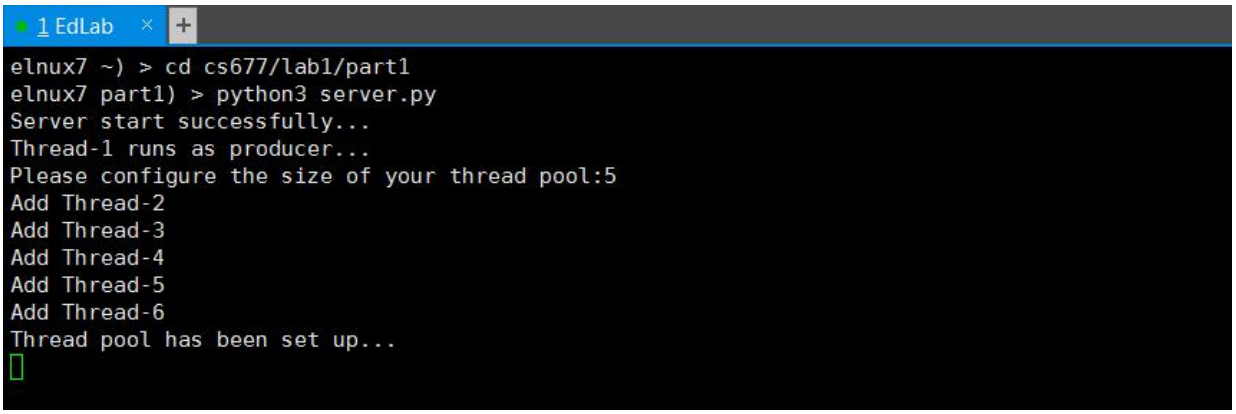
1. Server Startup Screenshots

EdLab View- Log in the UMass EdLab remote server, and upload our source code files “client.py” and “server.py” as follows.

Server Startup- After cd into current directory where you put your files (e.g cd cs677/lab1/part1), start up the server using command **$ python3 server.py**
Now, the server is running, and the “producer” thread starts correctly.



Configure the size of thread pool by simply typing an integer number **N**.
Then “consumer” threads will be added to our thread pool.



2. Functional Test Output

Client Terminal- Now we can look at the console at our local machine. Ensure the IP address & port number in file “client.py” are correct, and we can successfully connect to the server using command **$ python client.py**. Then tell the server which toy you are about to query.

Note that the price and stock of toys are static parameters in file “server.py”. Here we provide you screenshots of 3 different types of query requests.

```
6
7  # product catagory
8  products={'Tux':{'price':25.99, 'stock': 5},
9            'Whale':{'price':34.99, 'stock':0}}
10
```

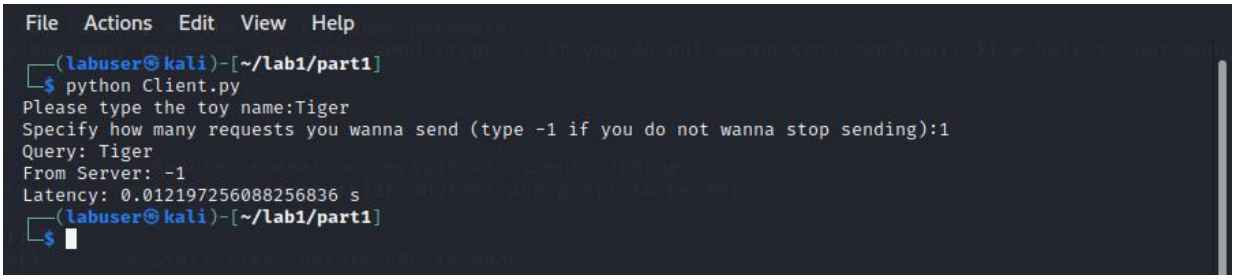
Query Tux: considering that Tux is one of the toys which is in stock, server returns the price of Tux, that is 25.99 successfully. And also we can check out the latency of each query as shown below.

```
File  Actions  Edit  View  Help
┌──(labuser㉿kali)-[~/lab1/part1]
└─$ python Client.py
Please type the toy name:Tux
Specify how many requests you wanna send (type -1 if you do not wanna stop sending):3
Query: Tux
From Server: 25.99
Latency: 0.011493444442749023 s
Query: Tux
From Server: 25.99
Latency: 0.011084794998168945 s
Query: Tux
From Server: 25.99
Latency: 0.009958982467651367 s
┌──(labuser㉿kali)-[~/lab1/part1]
└─$
```

Query Whale: considering that Whale is one of the toys which is out of stock, server returns 0 correctly. And also we can check out the latency of each query as shown below.

```
File  Actions  Edit  View  Help
┌──(labuser㉿kali)-[~/lab1/part1]
└─$ python Client.py
Please type the toy name:Whale
Specify how many requests you wanna send (type -1 if you do not wanna stop sending):3
Query: Whale
From Server: 0
Latency: 0.012356996536254883 s
Query: Whale
From Server: 0
Latency: 0.013092994689941406 s
Query: Whale
From Server: 0
Latency: 0.00991201400756836 s
┌──(labuser㉿kali)-[~/lab1/part1]
└─$
```

Query Tiger: considering that Tiger can not be found, server returns -1 correctly. And also we can check out the latency of each query as shown below.



Server Terminal- Now we can look at the console at remote EdLab server. Ensure the port number listening in file “server.py” are correct, and we can successfully set up a socket-based connection with the client.

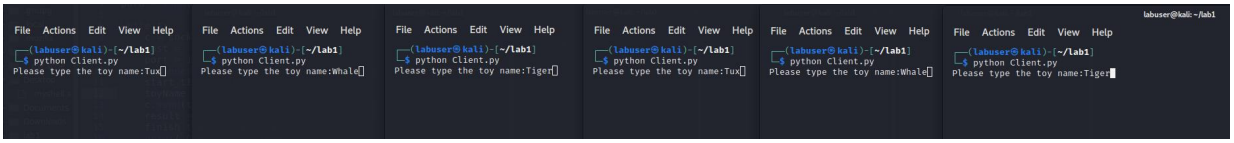
When query requests sequentially come from clients, the outputs of server are shown as follows. We can clearly see which “consumer” thread in the thread pool is handling with the current request from clients.

```
elnux7 part1) > python3 server.py
Server start successfully...
Thread-1 runs as producer...
Please configure the size of your thread pool:5
Add Thread-2
Add Thread-3
Add Thread-4
Add Thread-5
Add Thread-6
Thread pool has been set up...
('76.74.66.19', 51146) connected...
Thread-2 fetches a request from the queue!
Query has been done, Thread-2 is available!
('76.74.66.19', 51147) connected...
Thread-4 fetches a request from the queue!
Query has been done, Thread-4 is available!
('76.74.66.19', 51148) connected...
Thread-6 fetches a request from the queue!
Query has been done, Thread-6 is available!
('76.74.66.19', 51154) connected...
Thread-3 fetches a request from the queue!
Query has been done, Thread-3 is available!
('76.74.66.19', 51155) connected...
Thread-5 fetches a request from the queue!
Query has been done, Thread-5 is available!
('76.74.66.19', 51156) connected...
Thread-2 fetches a request from the queue!
Query has been done, Thread-2 is available!
('76.74.66.19', 51163) connected...
Thread-4 fetches a request from the queue!
Query has been done, Thread-4 is available!
('76.74.66.19', 51164) connected...
Thread-6 fetches a request from the queue!
Query has been done, Thread-6 is available!
('76.74.66.19', 51171) connected...
Thread-3 fetches a request from the queue!
Query has been done, Thread-3 is available!
```

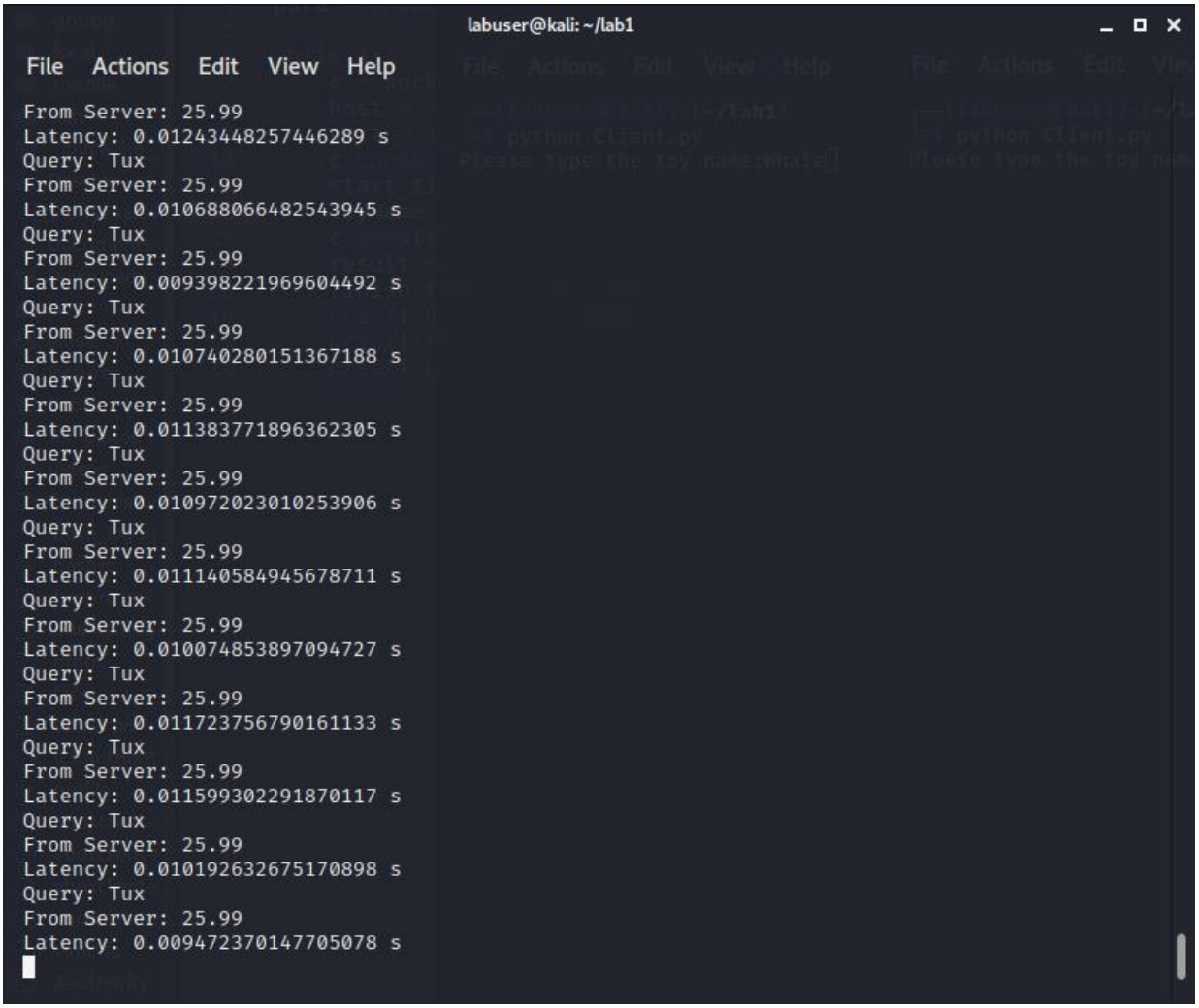
3. Load Test Output

Client Terminal- Similarly, before sending the request, all the configurations are the same as what we show you in functional test. But here we vary the number of clients from 1 to 6 and measure the latency as the load goes up.

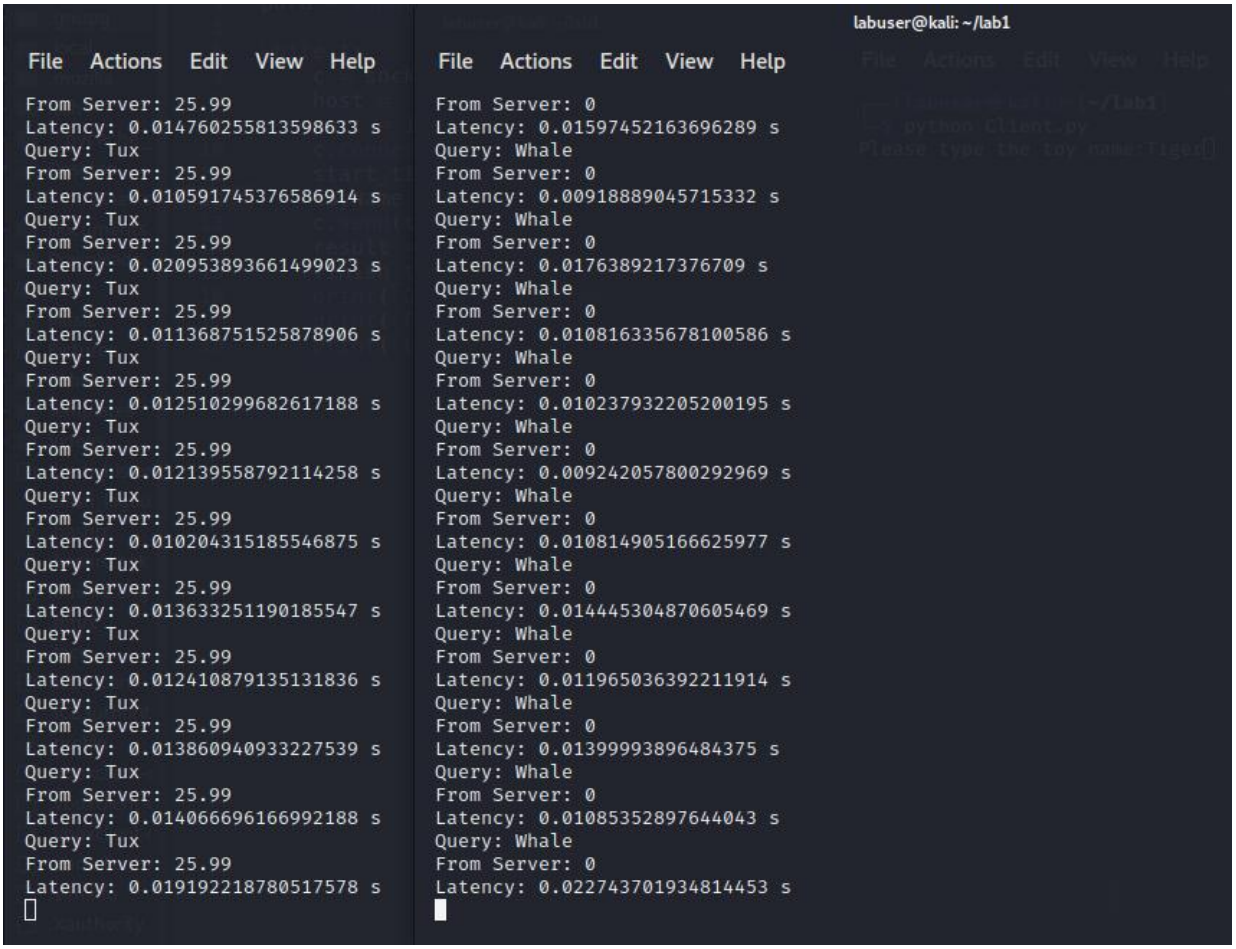
Open 6 terminals and type in the toy name as follows. Simulate different number of clients, and we can check out the latency of each query as shown below.



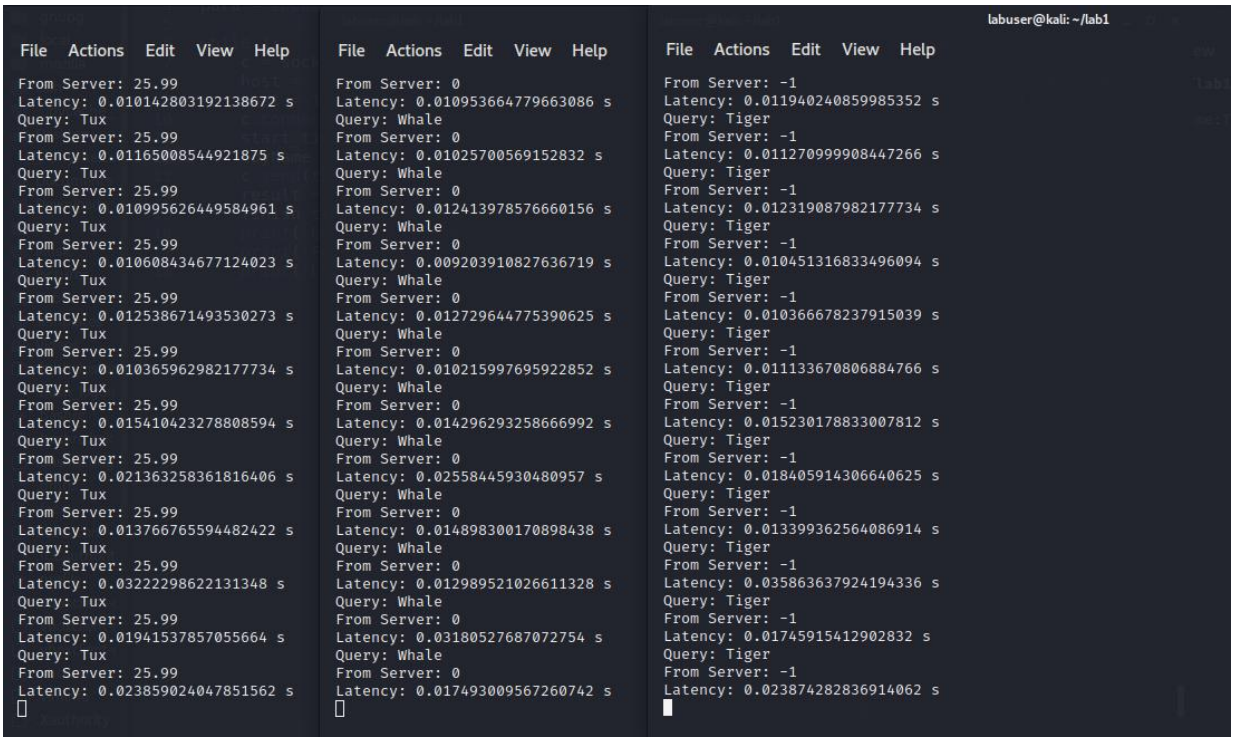
1 Client connected screenshot: only one client connects to the server.



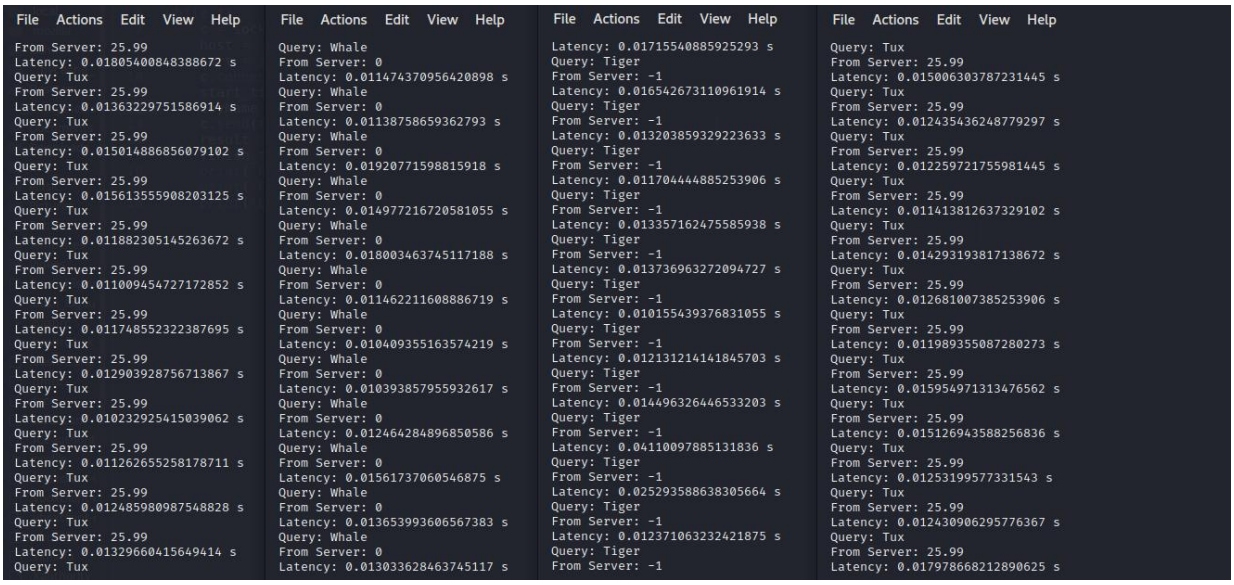
2 Clients connected screenshot: two clients connect to the server.



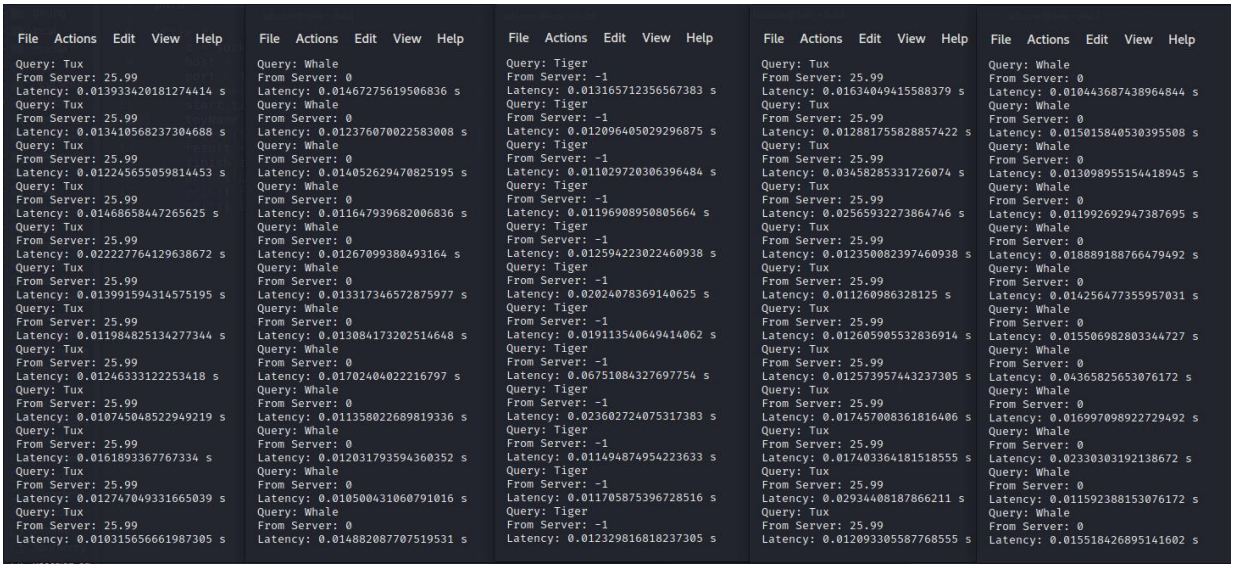
3 Clients connected screenshot: three clients connect to the server.



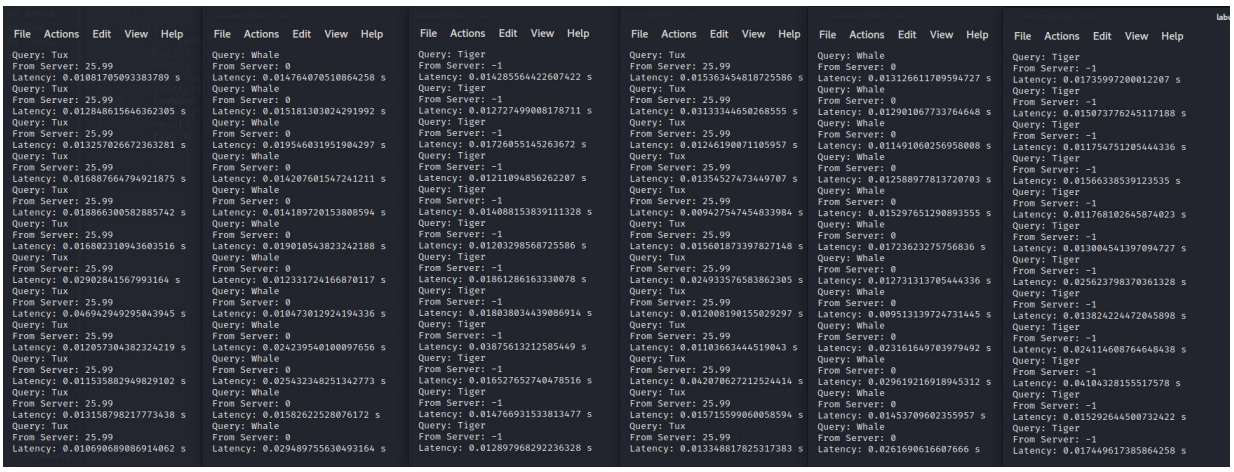
4 Clients connected screenshot: four clients connect to the server.



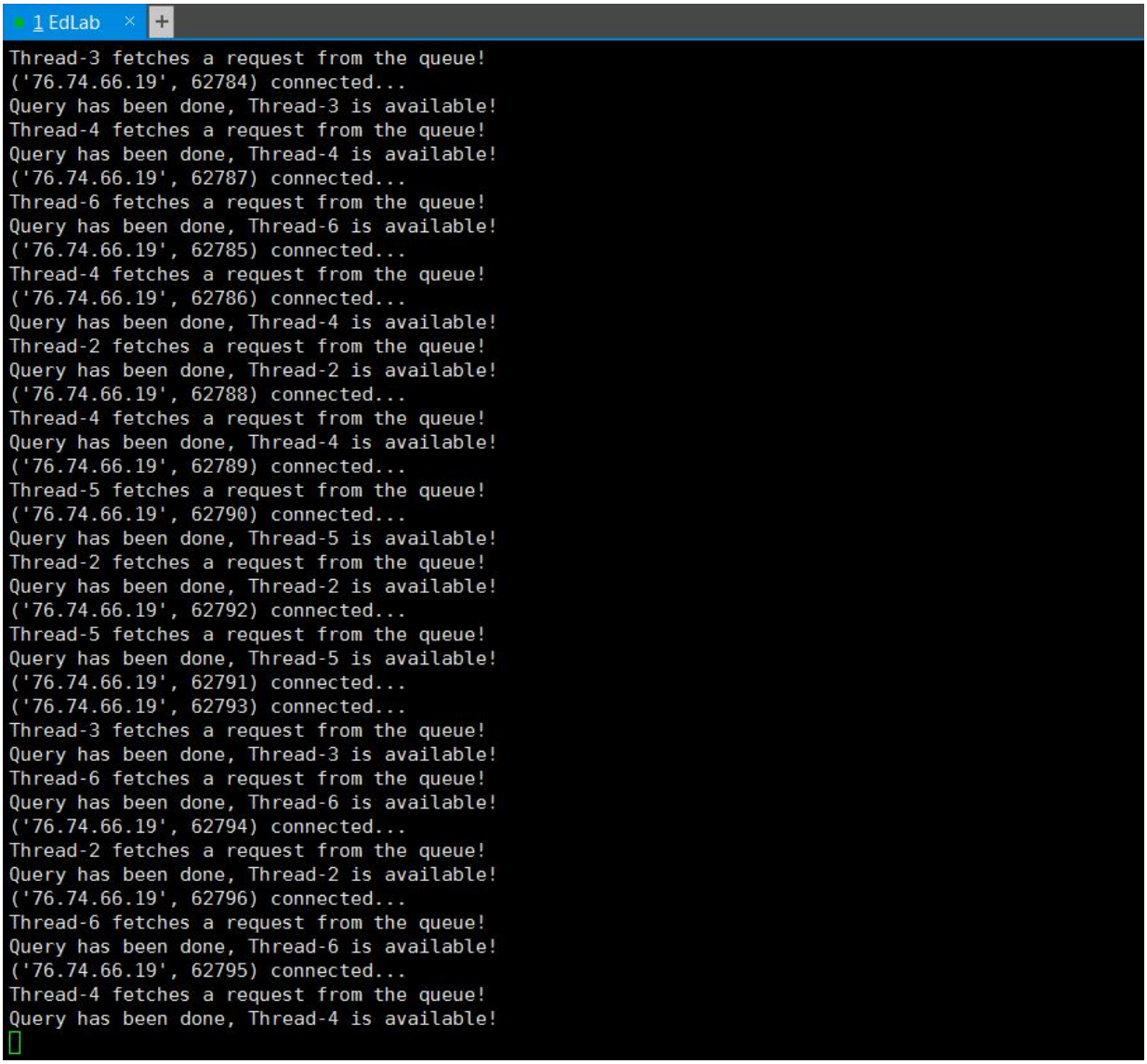
5 Clients connected screenshot: five clients connect to the server.



6 Clients connected screenshots: In this case, the number of clients(that is 6) is larger than the size of the thread pool(that is 5). We run all of the 6 clients, and we can check out the latency of each query as shown below.



Server Terminal- When multiple requests come, the outputs of server are shown as follows. We can clearly see which “consumer” thread in the thread pool is handling with request from clients.

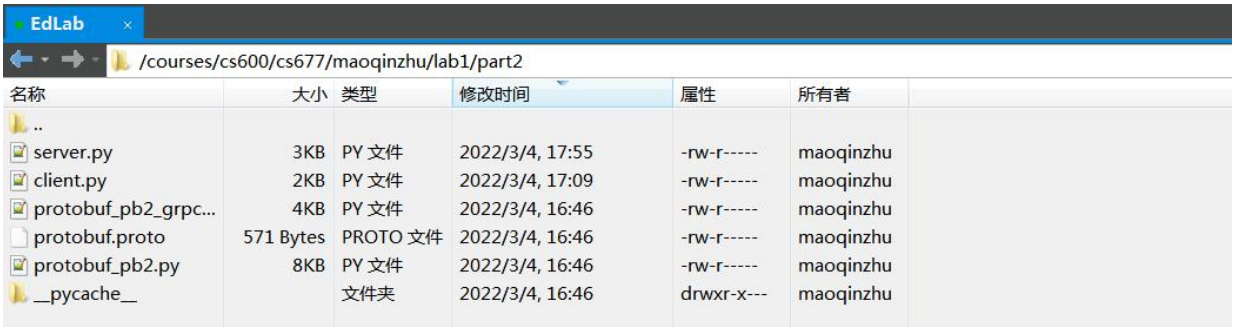


Part 2 - Implementation with gRPC and Built in Thread Pool

1. Server Startup Screenshots

EdLab View

Log in the UMass EdLab remote server, and upload our source code files as follows.



Server Startup

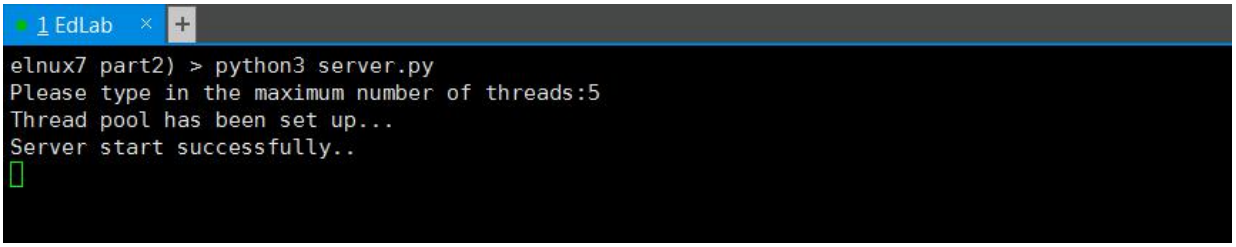
After cd into current directory where you put your files (e.g cd cs677/lab1/part2), you should install gRPC using **$ pip install grpcio_tools**

Then you start up the server using command **$ python3 server.py**

Now, the server is running, and the “producer” thread starts correctly.

```
elnux7 ~) > cd cs677/lab1/part2
```

```
elnux7 part2) > pip install grpcio_tools
Collecting grpcio_tools
  Downloading grpcio_tools-1.44.0-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.4 MB)
     |████████████████████████████████| 2.4 MB 5.8 MB/s
Requirement already satisfied: setuptools in /usr/lib/python3/dist-packages (from grpcio_tools) (45.2.0)
Collecting grpcio>=1.44.0
  Downloading grpcio-1.44.0-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (4.3 MB)
     |████████████████████████████████| 4.3 MB 10.7 MB/s
Requirement already satisfied: protobuf<4.0dev,>=3.5.0.post1 in /usr/local/lib/python3.8/dist-packages (from grpcio_tools) (3.12.2)
Requirement already satisfied: six>=1.5.2 in /usr/local/lib/python3.8/dist-packages (from grpcio>=1.44.0->grpcio_tools) (1.16.0)
Installing collected packages: grpcio, grpcio-tools
Successfully installed grpcio-1.44.0 grpcio-tools-1.44.0
```



Configure the size of thread pool by simply typing an integer number **N**.
Then “consumer” threads will be added to our thread pool.

2. Functional Test Output

Client Terminal- Now we can look at the console at our local machine. Ensure the IP address & port number in file “client.py” are correct, and we can successfully connect to the server using command **$ python client.py**. Then tell the server which toy you are about to query.

Note that the price and stock of toys are static parameters in file “server.py”. Here we provide you screenshots of different types of query requests.

```
54  # Set up toy database
55  # Static parameters(price and stock) to be set
56  def initCataData():
57      catagories = []
58      toy_Tux = protobuf_pb2.Toy(name='Tux', stock=100, price=25.99)          # Follow the message structure Toy defined in proto file
59      toy_Whale = protobuf_pb2.Toy(name='Whale', stock=100, price=34.99)
60      toy_elephant = protobuf_pb2.Toy(name='Elephant', stock=100, price=29.99)
61      toy_bird = protobuf_pb2.Toy(name='Bird', stock=50, price=39.99)
62      catagories.append(toy_Whale)
63      catagories.append(toy_Tux)
64      catagories.append(toy_elephant)
65      catagories.append(toy_bird)
66      return catagories
67
```

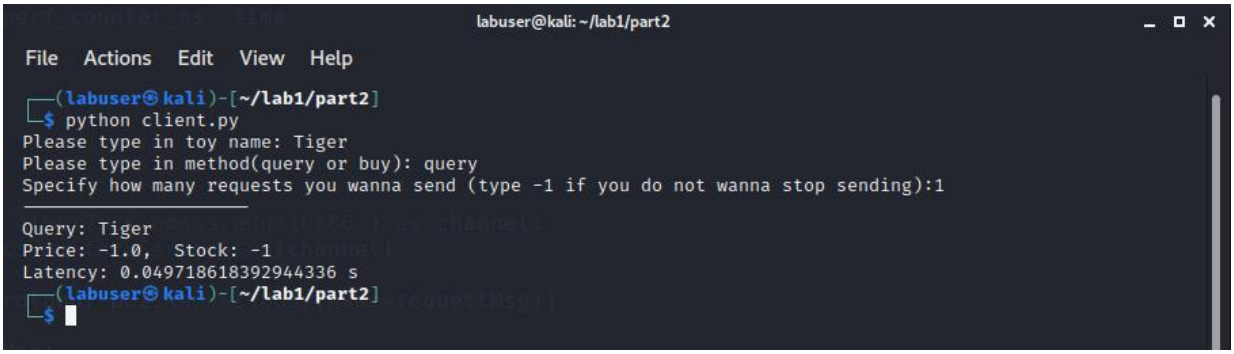
Query Tux: considering that Tux is one of the toys which is in stock, server returns the price of Tux, that is 25.99 successfully. And also we can check out the latency of each query as shown below.

```
File  Actions  Edit  View  Help

┌──(labuser㉿kali)-[~/lab1/part2]
└─$ python client.py
Please type in toy name: Tux
Please type in method(query or buy): query
Specify how many requests you wanna send (type -1 if you do not wanna stop sending):3
————————————
Query: Tux
Price: 25.99,  Stock: 100
Latency: 0.04188179969787598 s
————————————
Query: Tux
Price: 25.99,  Stock: 100
Latency: 0.03304338455200195 s
————————————
Query: Tux
Price: 25.99,  Stock: 100
Latency: 0.037835121154785156 s
┌──(labuser㉿kali)-[~/lab1/part2]
└─$
```

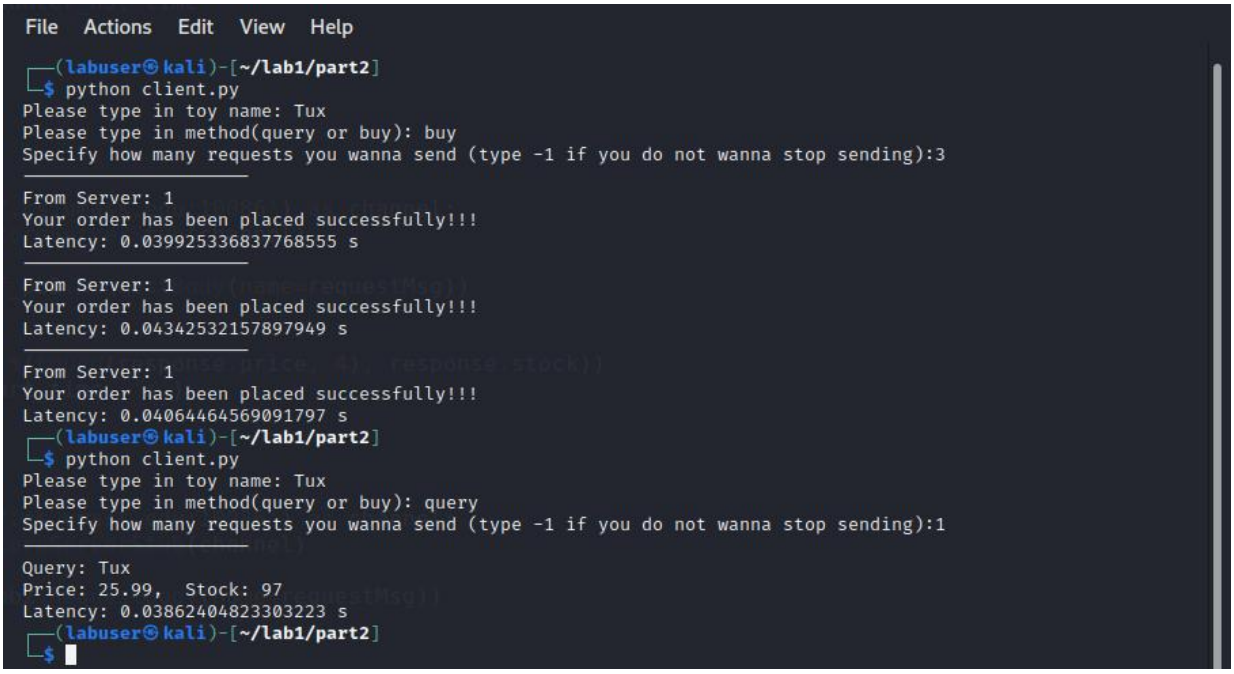
Query Bird: considering that Bird is one of the toys which is in stock, server returns the price of Bird, that is 39.99 successfully. And also we can check out the latency of each query as shown below.

```
File  Actions  Edit  View  Help

┌──(labuser㉿kali)-[~/lab1/part2]
└─$ python client.py
Please type in toy name: Bird
Please type in method(query or buy): query
Specify how many requests you wanna send (type -1 if you do not wanna stop sending):2
————————————
Query: Bird
Price: 39.99,  Stock: 50
Latency: 0.04297804832458496 s
————————————
Query: Bird
Price: 39.99,  Stock: 50
Latency: 0.05867505073547363 s
┌──(labuser㉿kali)-[~/lab1/part2]
└─$
```

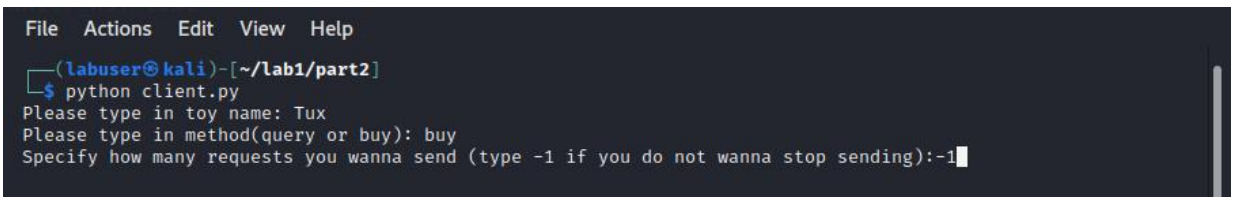
Query Tiger: considering that Tiger is not a valid input, server returns -1 and catch the exception correctly. And also we can check out the latency of each query as shown below.

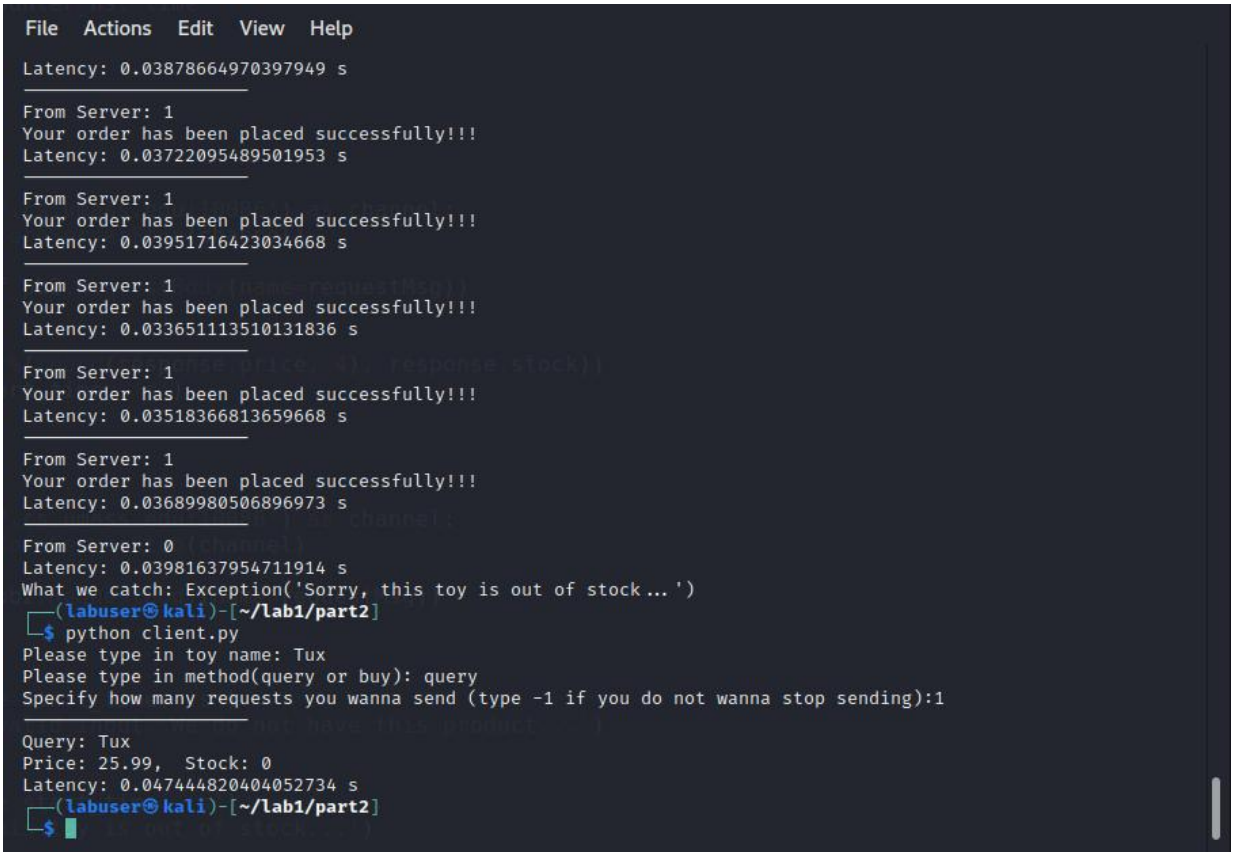


Buy Tux: considering that Tux is one of the toys which is in stock. If we send 3 request of buying, the stock is reduced by 3 correctly. And also we can check out the latency of each query as shown below.



Now if we input -1 when the console asks hou many requests we wanna send. It will not stop sending requests of buying until Tux is out of stock. And finally, we can catch a exception correctly.





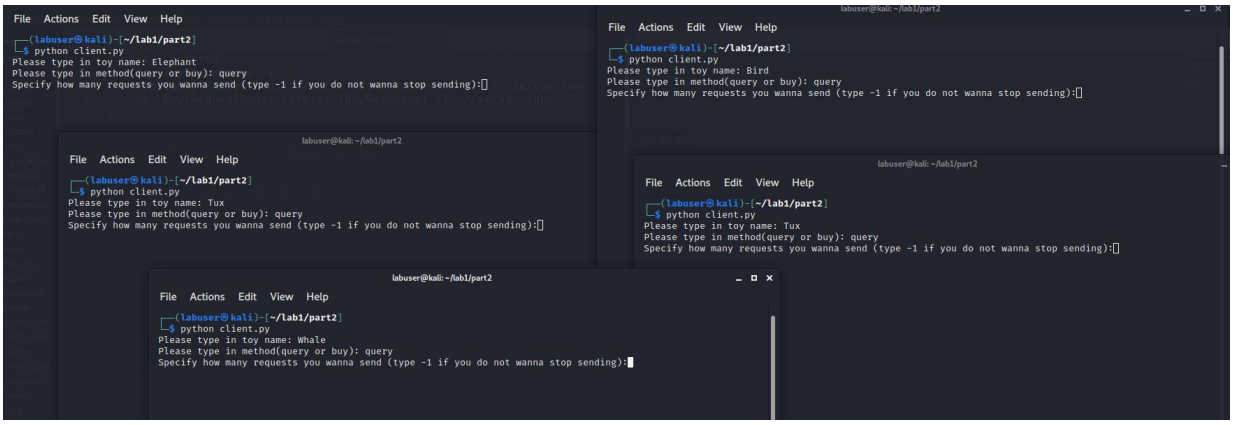
Buy Tiger: considering that Tiger is not a valid input, server returns -1 and catch the exception correctly. And also we can check out the latency of each query as shown below.

3. Load Test Output

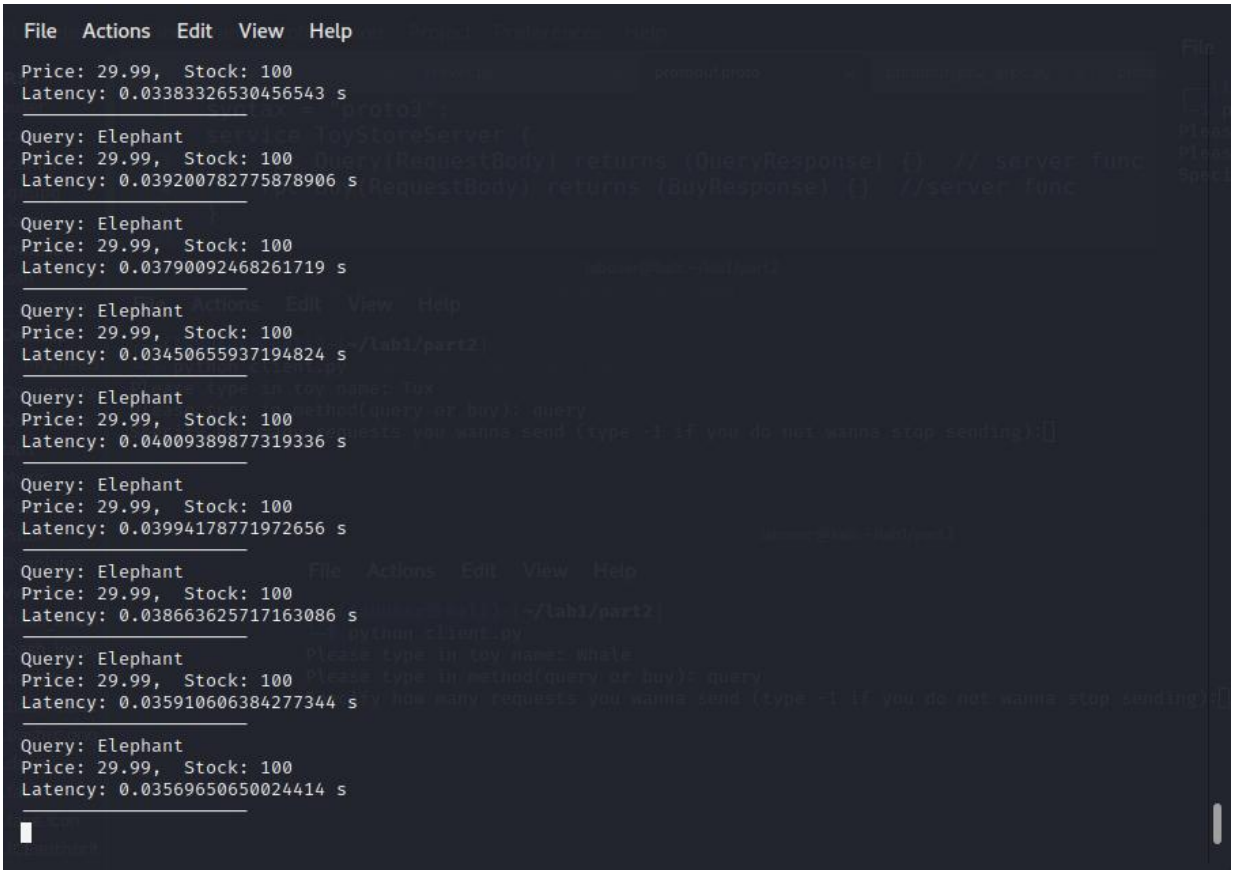
Queue Method

Similarly, here we vary the number of clients from 1 to 5 and measure the latency as the load goes up.

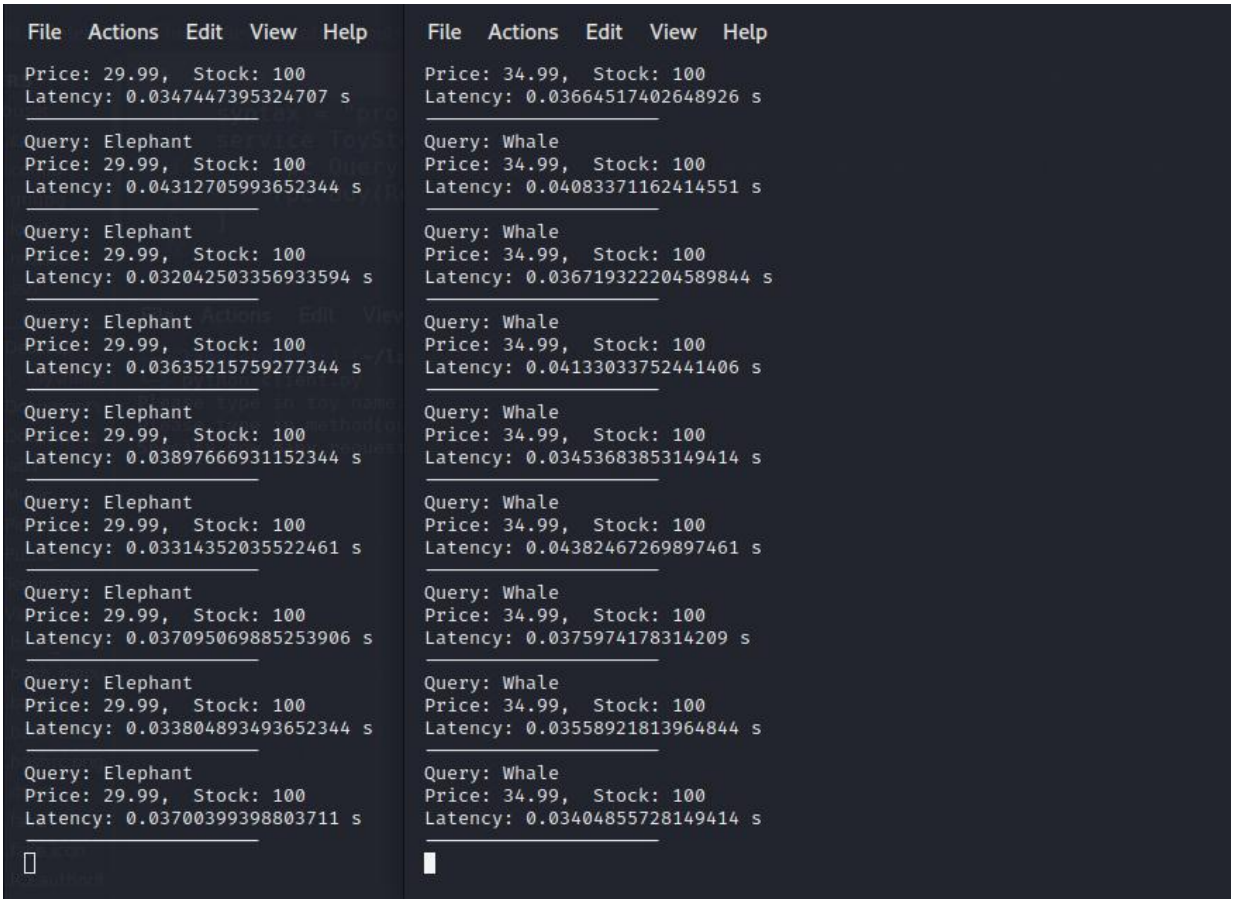
Open 5 terminals and type in the toy name, the method name and the number of requests as follows. Simulate different number of clients, and we can check out the latency of each query as shown below.



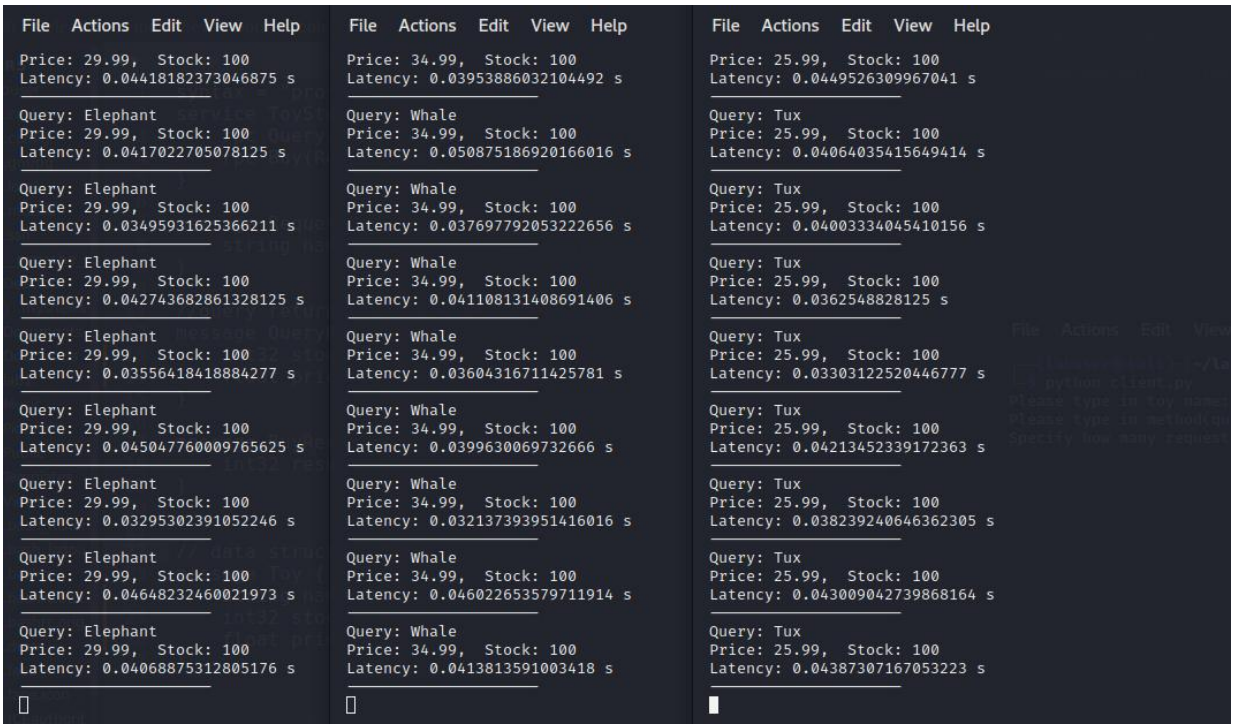
1 Client connected screenshot: only one client connects to the server.



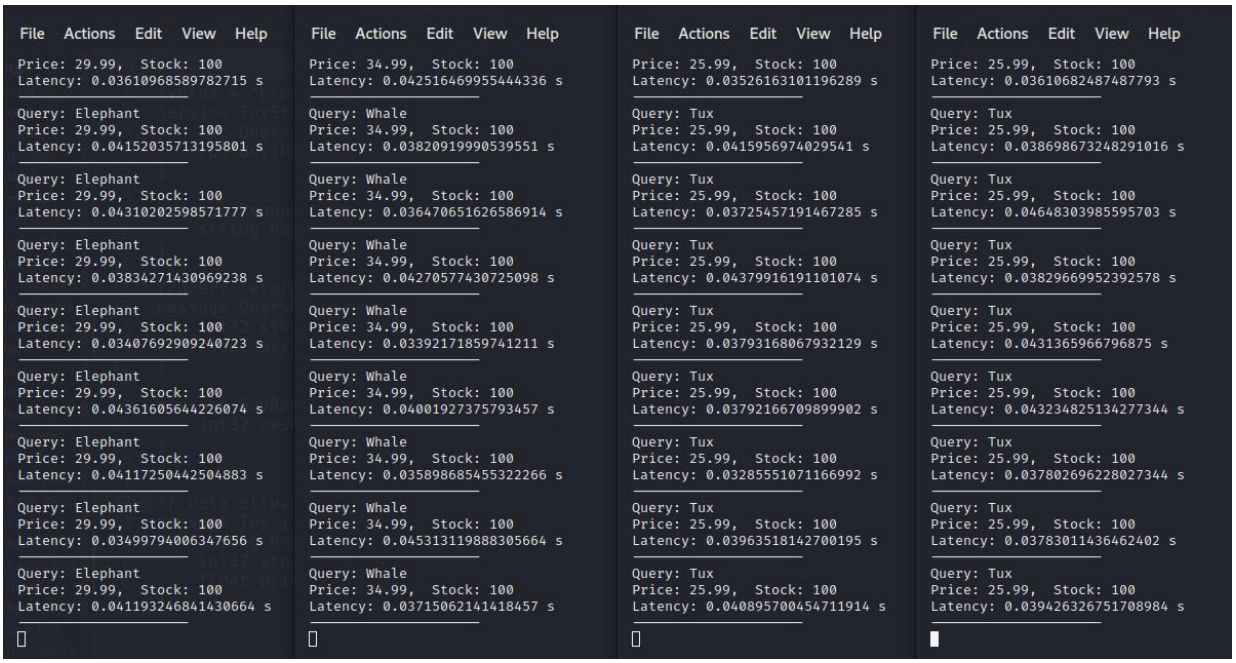
2 Clients connected screenshot: two clients connect to the server.



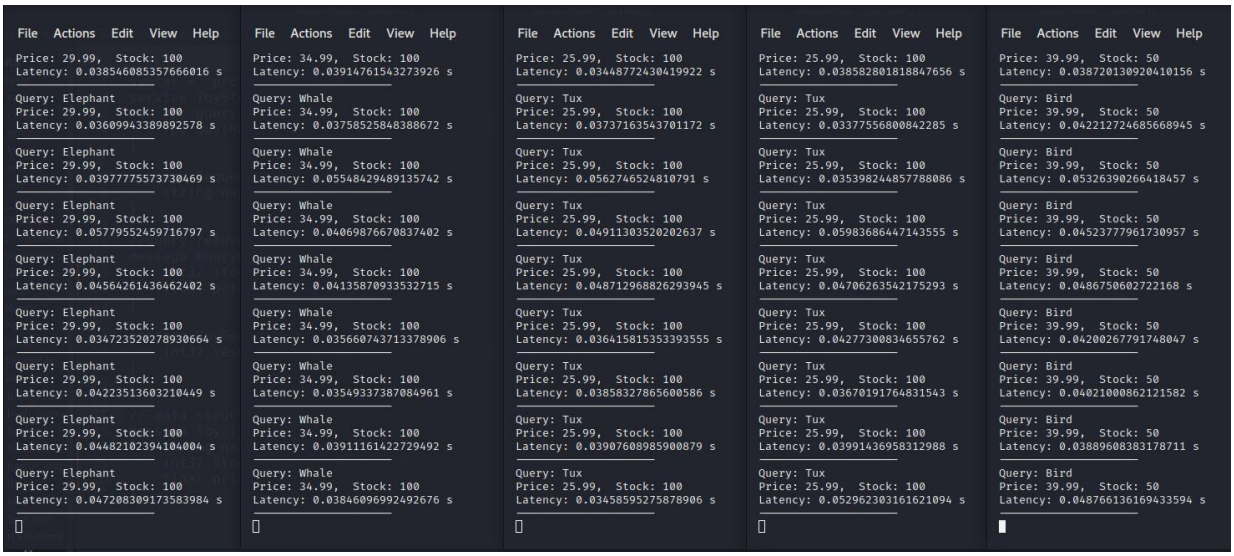
3 Clients connected screenshot: three clients connect to the server.



4 Clients connected screenshot: four clients connect to the server.



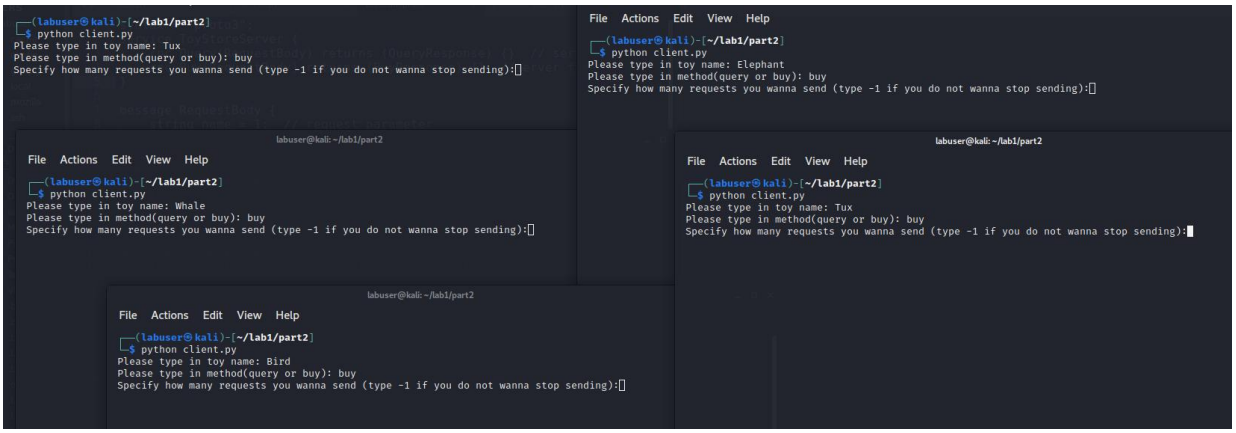
5 Clients connected screenshot: five clients connect to the server.



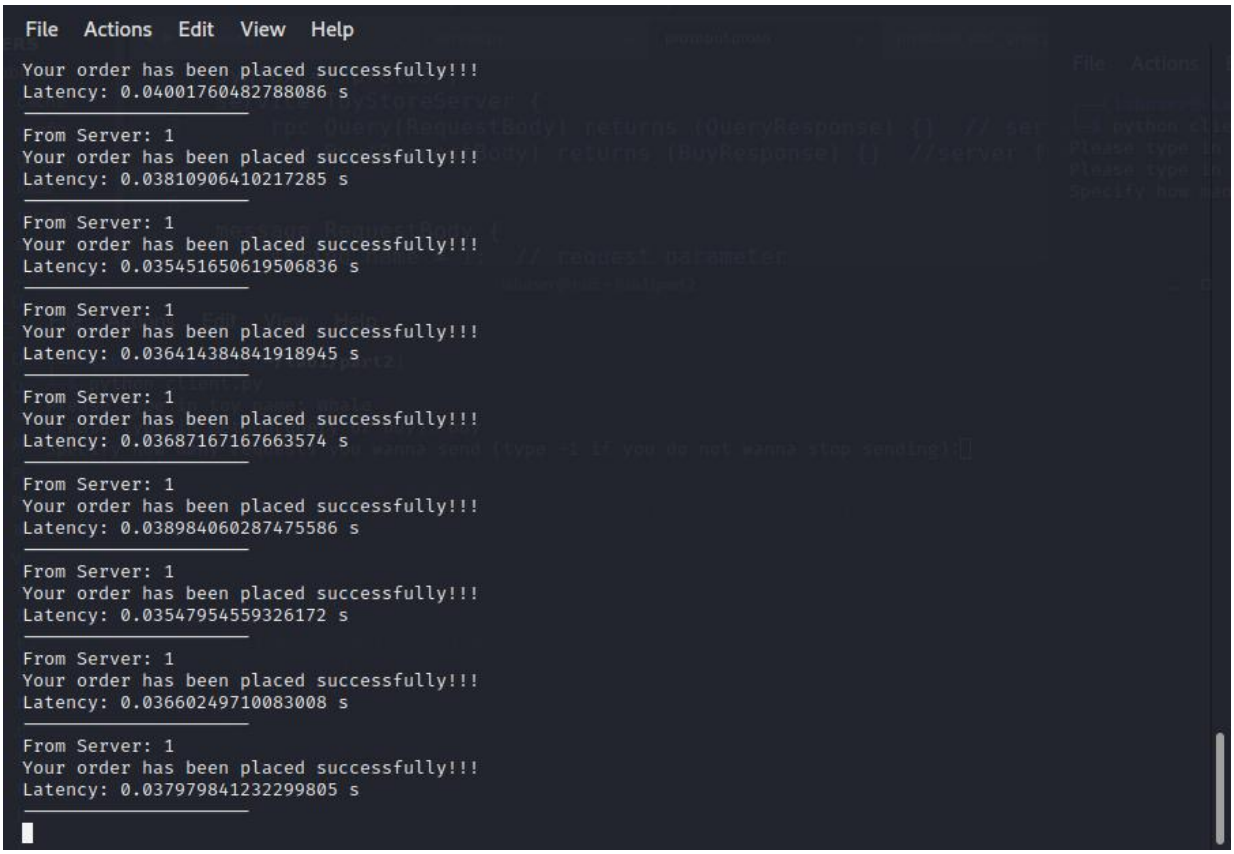
Buy Method

Similarly, here we vary the number of clients from 1 to 5 and measure the latency as the load goes up.

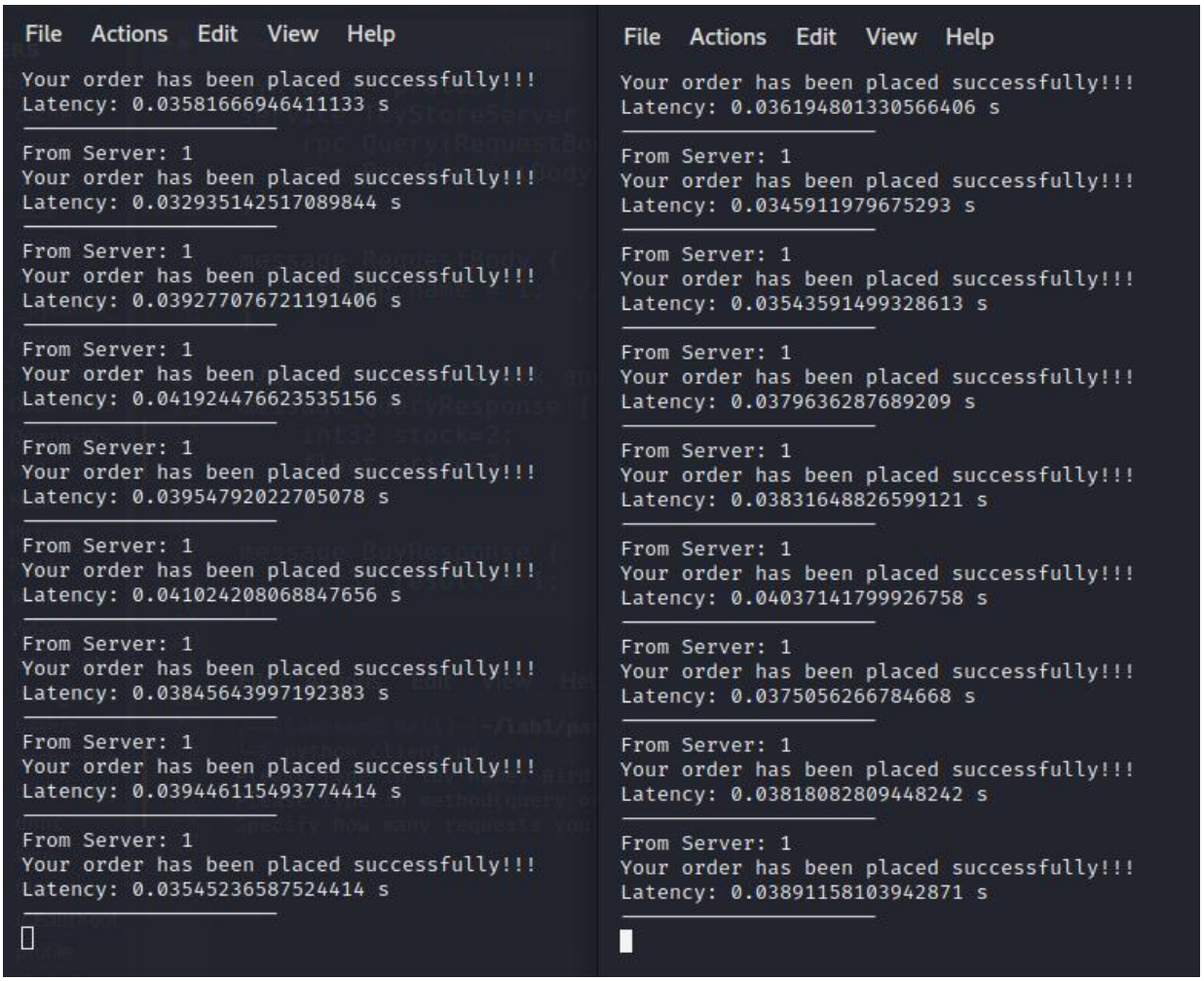
Open 5 terminals and type in the toy name, the method name and the number of requests as follows. Simulate different number of clients, and we can check out the latency of each query as shown below.



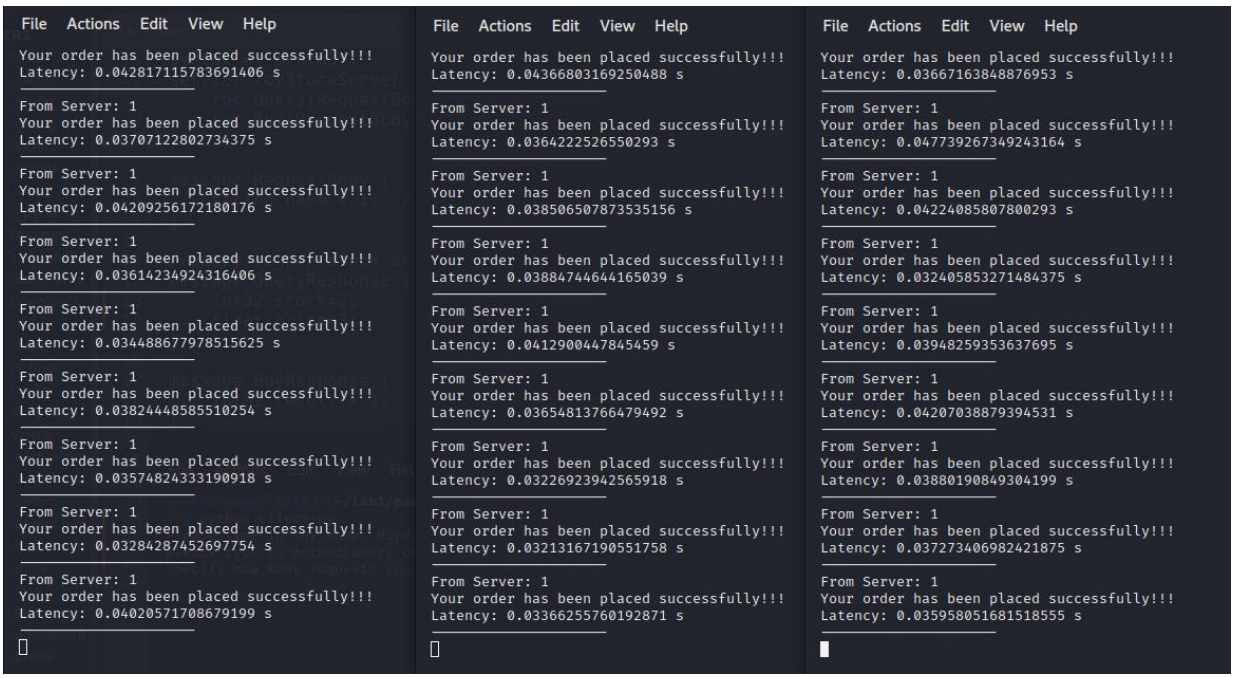
1 Client connected screenshot: only one client connects to the server.



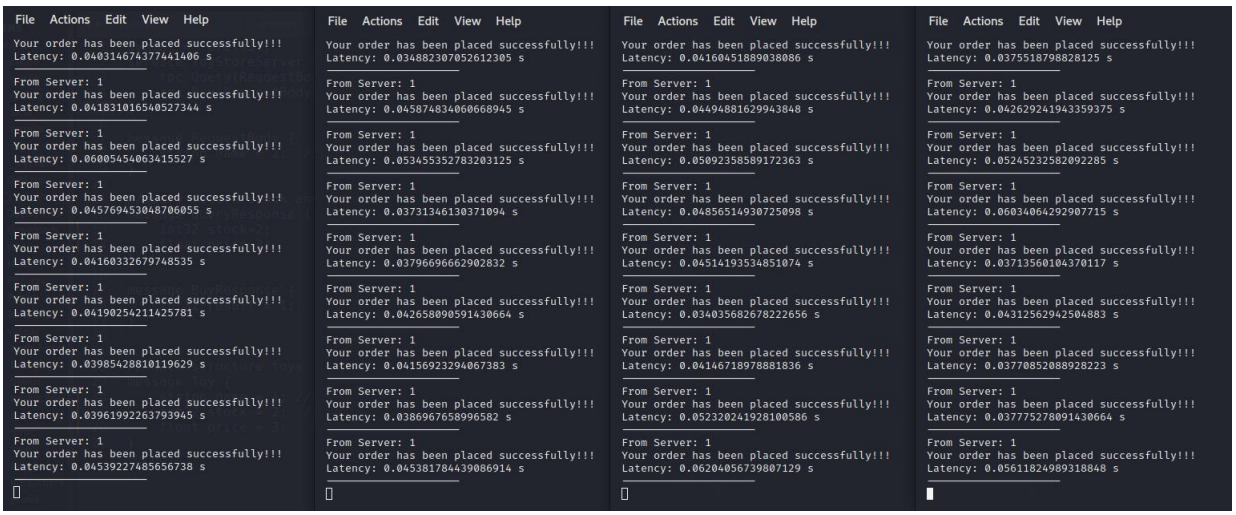
2 Clients connected screenshot: two clients connect to the server.



3 Clients connected screenshot: three clients connect to the server.



4 Clients connected screenshot: four clients connect to the server.



5 Clients connected screenshot: five clients connect to the server.