**Computer Systems Technology**

British Columbia Institute of Technology

COMP 8005 - Assignment2- Testing

Albert Huang&

Aiyan Ma

Mar 5, 2018

Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Date** | **Version** | **Revised by** | **Remarks** |
| 1 | 2018-Feb-26 | V0.1 | Aiyan,Ma | The initial draft for Test plan and test case |
| 2 | 2018-Mar-05 | V1.0 | Albert, H. | Adding all test case contain. |

# Table of Contents

[Table of Contents 2](#_Toc752581601)

[1. General Client Testing 5](#_Toc56119058)

[1.1 Test Outline 5](#_Toc1008404177)

[1.2 Test Case Descriptions 6](#_Toc507739819)

[1.2.1 Test 1 6](#_Toc141743915)

[1.2.2 Test 2 7](#_Toc1153072673)

[1.2.3 Test 3 8](#_Toc716315972)

[1.2.4 Test 4 9](#_Toc1306847237)

[1.2.5 Test 5 10](#_Toc222097032)

[1.2.6 Test 6 12](#_Toc385064151)

[1.2.7 Test 7 15](#_Toc346191977)

[1.2.8 Test 8 16](#_Toc830245117)

[1.2.9 Test 9 17](#_Toc1223570525)

[2. Multi-Thread Testing 18](#_Toc1117716117)

[2.1 Test Outline 18](#_Toc646085382)

[2.2 Test Case Descriptions 19](#_Toc551342860)

[2.2.1 Test 1 19](#_Toc1884965313)

[2.2.2 Test 2 21](#_Toc2116248472)

[2.2.3 Test 3 23](#_Toc81929831)

[2.2.4 Test 4 25](#_Toc998040427)

[3. Select Mode Server Testing 26](#_Toc880851549)

[3.1 Test Outline 26](#_Toc1106881280)

[3.2 Test Case Descriptions 27](#_Toc427861737)

[3.2.1 Test 1 27](#_Toc1335666467)

[3.2.2 Test 2 28](#_Toc1619215528)

[3.2.3 Test 3 30](#_Toc1867741805)

[3.2.4 Test 4 32](#_Toc1065727926)

[4. Epoll Mode Server Testing 33](#_Toc1144347112)

[4.1 Test Outline 33](#_Toc2010909694)

[4.2 Test Case Descriptions 34](#_Toc771195337)

[4.2.1 Test 1 34](#_Toc897501021)

[4.2.2 Test 2 35](#_Toc827314395)

[4.2.3 Test 3 37](#_Toc1123747466)

[4.2.4 Test 4 39](#_Toc911494223)

# General Client Testing

## 1.1 Test Outline

| Rule # | Test Description | Tool Used | Expected Results | Pass/Fail |
| --- | --- | --- | --- | --- |
| 1 | Program successfully connects to the multi-threaded server | Server/Client application | Server receives connection request from amount of clients. | Pass. Detailed results are attached. |
| 2 | Program successfully connects to the select server. | Server/Client application | Server receives connection request from amount of clients. | Pass. Detailed results are attached. |
| 3 | Program successfully connects to the epoll server. | Server/Client application | Server receives connection request from amount of clients. | Pass. Detailed results are attached. |
| 4 | Client creates processes  Server/Client equal to the number of cpu of the client host. | Server/Client application | In task manager processes = to the number of total cpus  on the PC and log file will show that. | Pass. Detailed results are attached. |
| 5 | Client creates the correct  amount of connections  specified by the user. | Server/Client application | Client set the total connections of one process, we can find the connection number by netstat command. | Pass. Detailed results are attached. |
| 6 | Client resent the data to server many times which is specified by the user. | Server/Client application | Client set the repeating times to resent the data to server, we can find from log file. | Pass. Detailed results are attached. |
| 7 | Client would sleep a while during sending work and the time interval is specified by the user. | Server/Client application | Client set the sleeping time of the gap between two sending data events, we can find the proof from log file. | Pass. Detailed results are attached. |
| 8 | Client shuts down after all child processes have  finished. | Server/Client application. | Client finishes and will exit. | Pass. Detailed results are attached. |
| 9 | Client print a brief report | Server/Client application. | When client close all connections will print a report and exit, we can find the report at the terminal. | Pass. Detailed results are attached. |

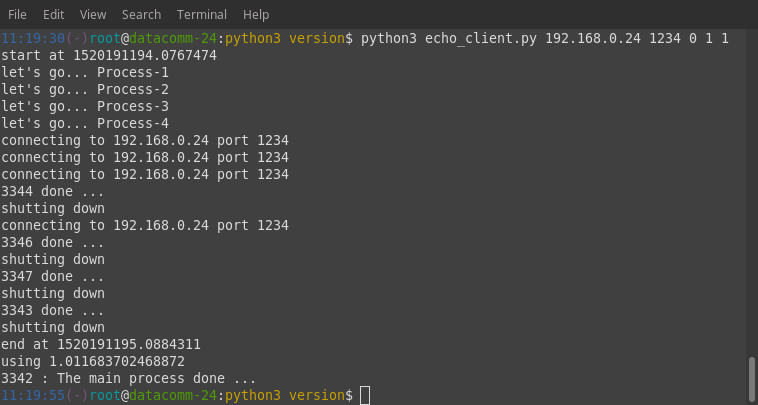
## 1.2 Test Case Descriptions

### 1.2.1 Test 1

Client successfully connects to the multi-threaded server. Server receives connections requests send by the client.

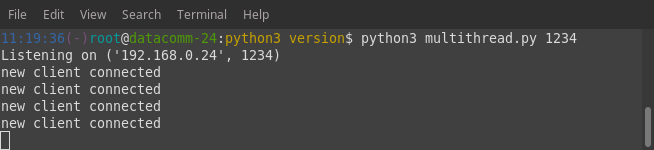
# **Run the Python Version:**

The client side:



We can see that under the python echo\_client.py command, there are some info shows that it generates 4 process to connect with the server, then disconnect.

The server side:



We can find that the server side shows that there are 4 connections established.

Since Java version almost the same as python’s so I won’t paste the similar screen shot for this test.

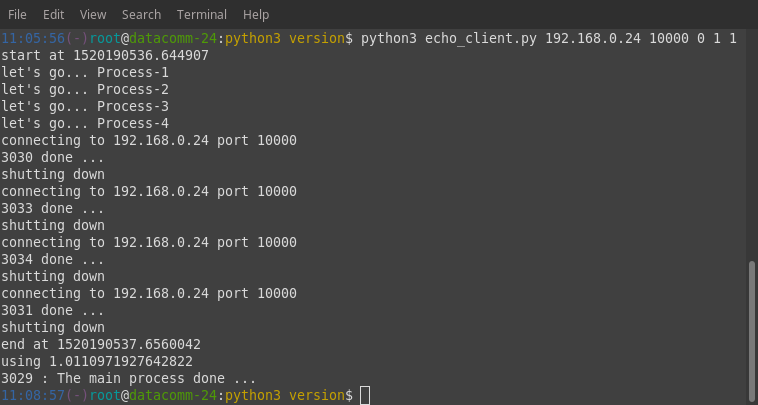
Pass.

### 1.2.2 Test 2

This Client successfully connects to the select server. Server receives connections requests send by the client.

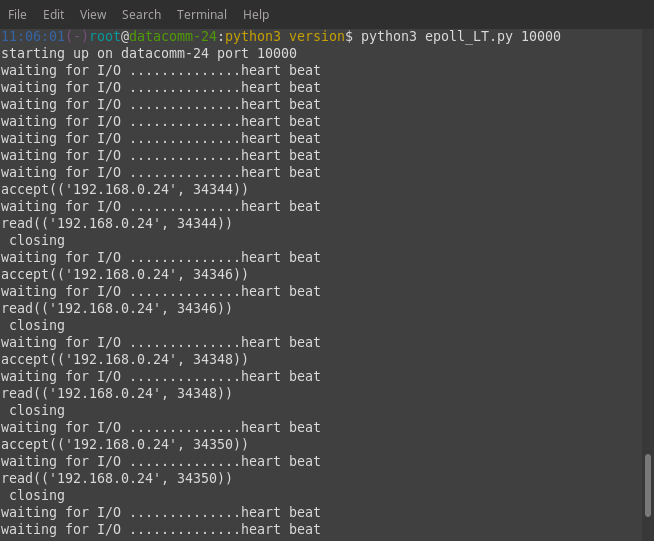
# **Run the Python Version:**

The client side:



We can see the same as previous one, the only different is port number...while

The server side:



We can find that the server side output is different from the multi-thread one, and we can find the client ports number here, and the then all these connection are closed since no data coming as excepted.

Test passes, all connections successfully connect the server.

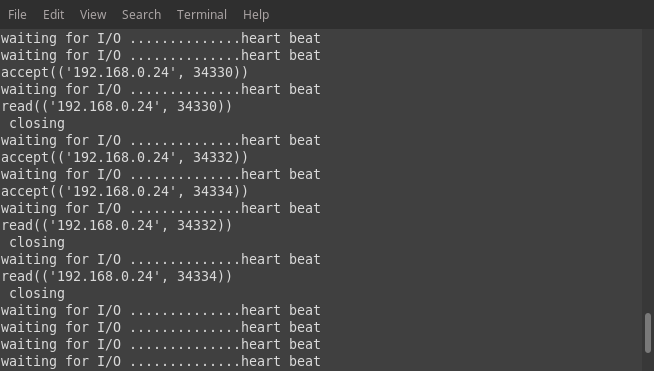
### 1.2.3 Test 3

Client successfully connects to the epoll server. Server receives connections requests send by the client.

# **Run the Python Version:**

Since client side command is the same as previous one, the only different is port number...while

The server side:

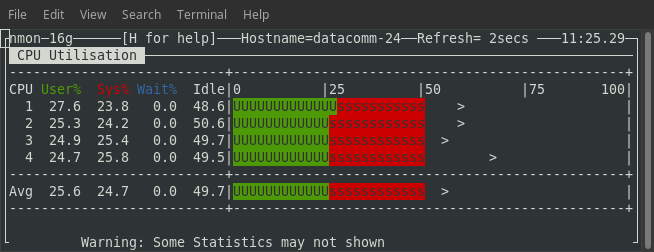


We can find that the server side output is still very similar to select mode version, the only difference is the connection ports number

Test passes, all connection have successfully established on the epoll server.

### 1.2.4 Test 4

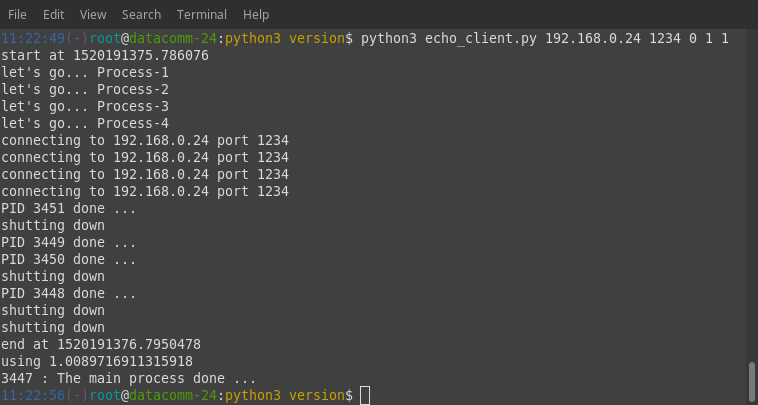
Client creates processes equal to the total CPU’s on the PC running the client. Expected will be 4 processes as the computer tested on has 4 CPU’s.



Just shows that here are 4 CPUs here.

# **Run the Python Version:**

The client side:



We can find that there are 4 of processes and 4 connections are created here;

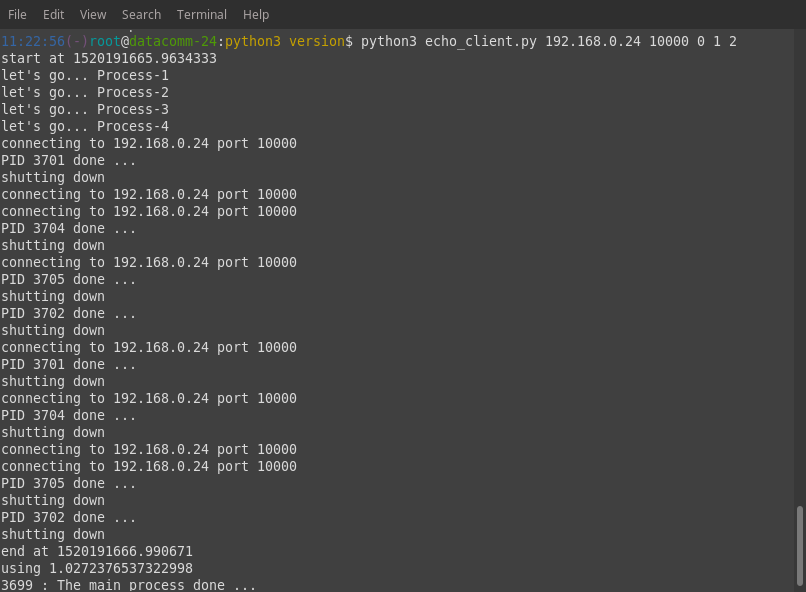
Test passes, client generate 4 processes.

### 1.2.5 Test 5

Client creates the correct amount of connections specified by the user.

# **Run the Python Version:**

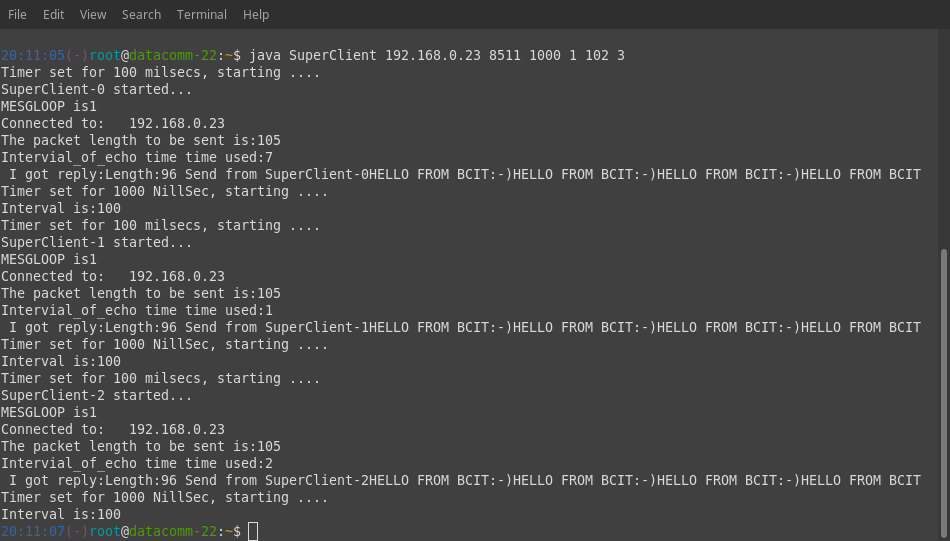
The client side:



In my command, the last argument is the number of working thread that each process will generate. So here the number is 2 means that there will generate 8 working threads. And it shows 8 PID finally shutdown since no data sending.

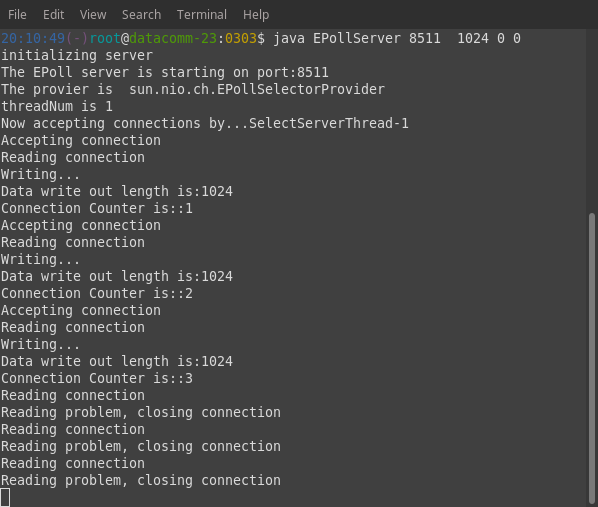
# **Run the Java Version:**

The client side:



The final argument in the command is the number of the workers, here is 3 means there will be 3 connections, and then they sent the same length of data and received the echo back.

The server side:



The info showed in server side proof the client side activity;

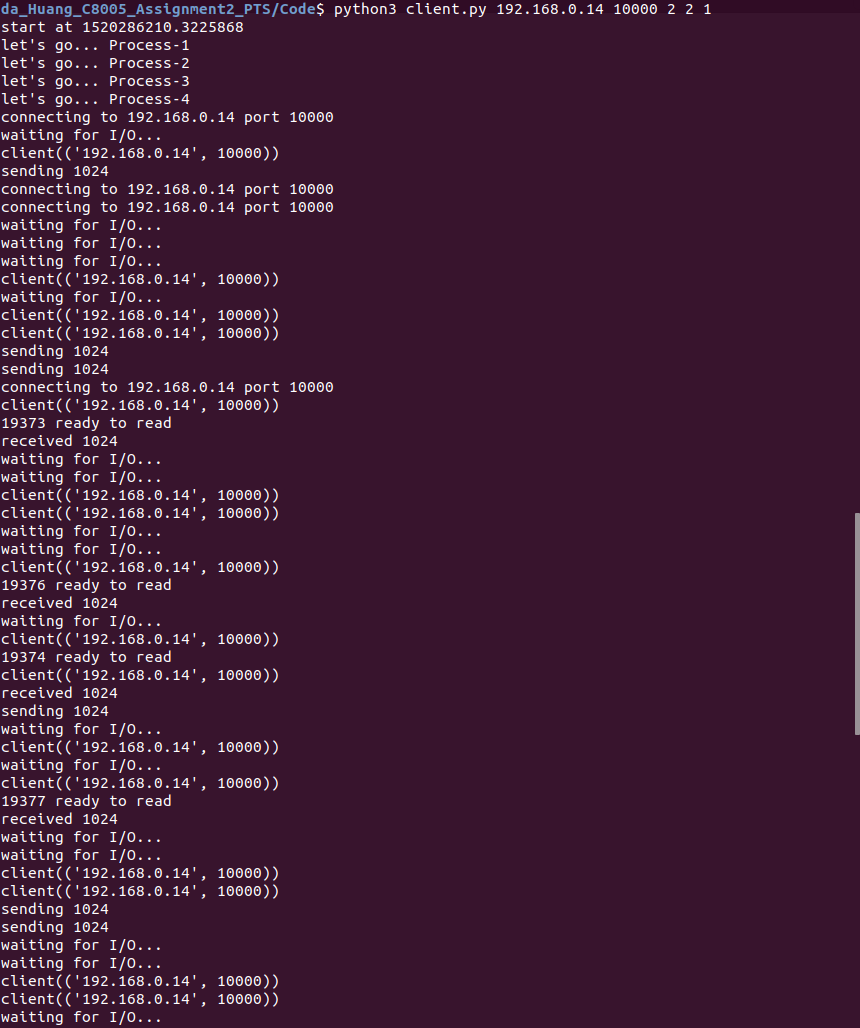
Test passed.

### 1.2.6 Test 6

Client resent the data to server many times which is specified by the user.

# **Run the Python Version:**

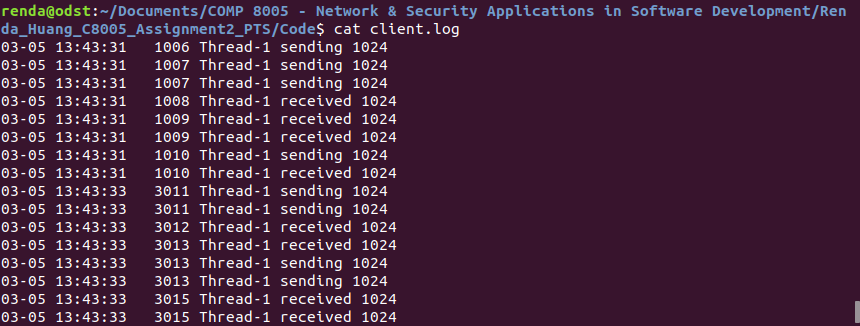
The client side:



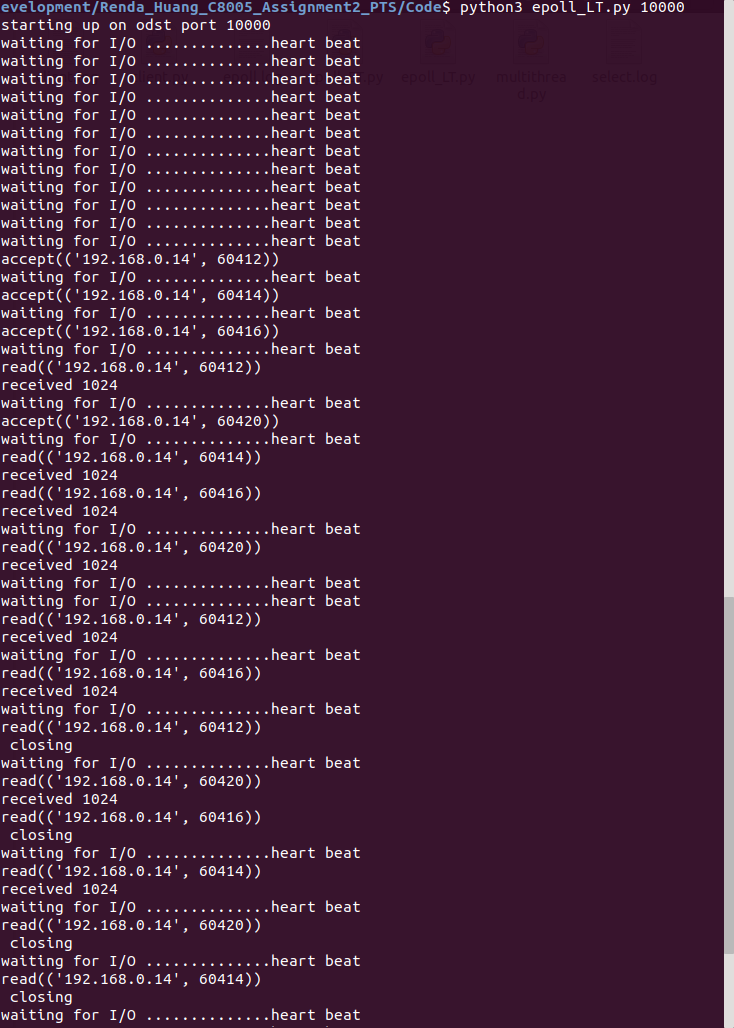
Since my command format is : python3 client.py <server ip> <server port> <repeating times> <sleeping time> <worker thread number>

So we can find the repeating time number is 2 and worker thread is 1 since there are 4 processes therefore 4 workers in all and each worker would send 2 times.

Client log shows:



The server side:



# **Run the Java Version:**

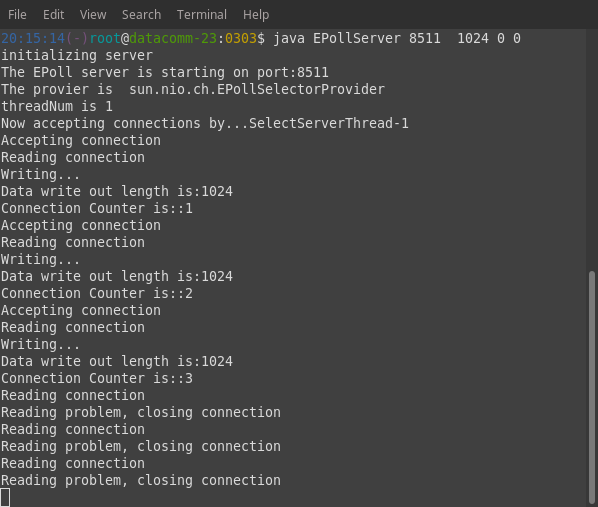
The client side:



Since the command format is : java SuperClient <server ip> <server port> <sleeping time while generating threads> <repeating times> <sleeping time while sending data> <workers>

So that, here should generate one worker sending three times.

The server side:



Server received and echo back

Test pass.

### 1.2.7 Test 7

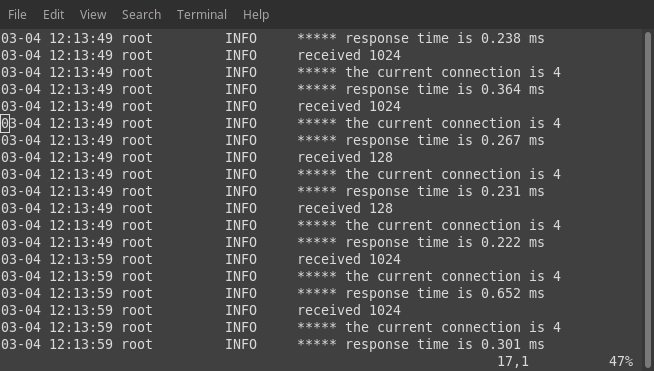
Client would sleep a while during sending work and the time interval is specified by the user.

# **Run the Python Version:**

Command: python3 client.py 10000 1 10 1

While I set the interval to 10

The server side log:



We can find that there are two bunch of data send by client, one bunch of data was sending on 12:13:49; the other bunch is on 12:13:59

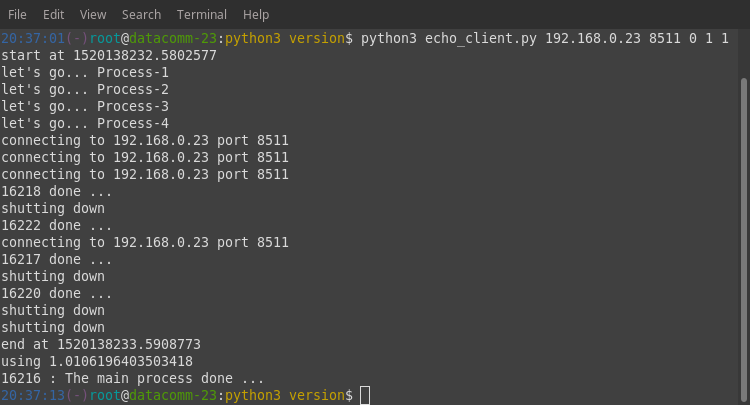
Test pass.

### 1.2.8 Test 8

Client shuts down after all child processes have finished.

# **Run the Python Version:**

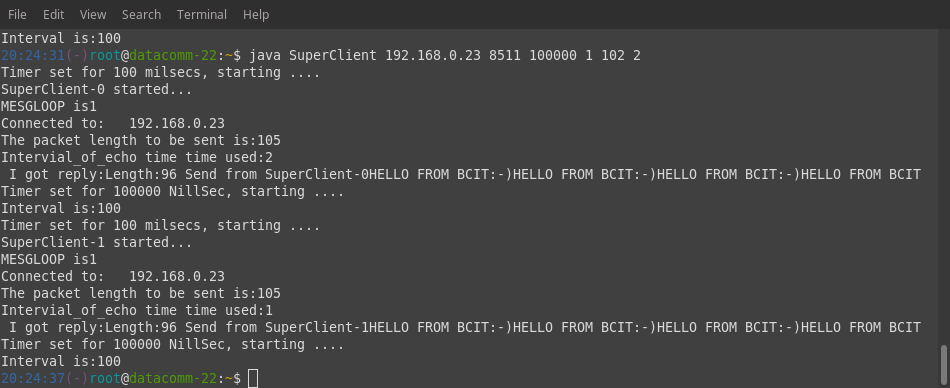
The client side:



We can find that when all the connections are closed then the client would shutdown.

# **Run the Java Version:**

The client side:



In java version, it s simpler than python version, it just done the job and exit, no more information printed.

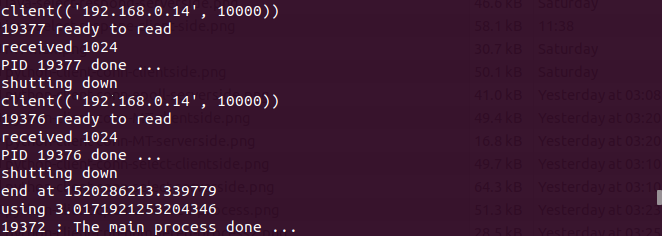
Test pass.

### 1.2.9 Test 9

Client print a brief report

# **Run the Python Version:**

The client side:



The python version client’s report is brief just list the time elapse, more information would be in log file.

Test pass.

# Multi-Thread Testing

## 2.1 Test Outline

| Rule # | Test Description | Tool Used | Expected Results | Pass/Fail |
| --- | --- | --- | --- | --- |
| 1 | Program successfully receives connection request from clients. | Sever/Client  application. | When server receive connection request, it would picks it up. | Pass. Detailed results are attached. |
| 2 | Program successfully responds to the client | Sever/Client  application. | The byte size sent to Server. See  the server is the test case same size of message sent back. | Pass. Detailed results are attached. |
| 3 | Program successfully listens on the port user specified test | Sever/Client  application. | Change a port to start the service and make client side change the destination port number | Pass. Detailed results are attached. |
| 4 | Server logs print expected information | Sever/Client  application. | Log shows several level of log data. | Pass. Detailed results are attached. |

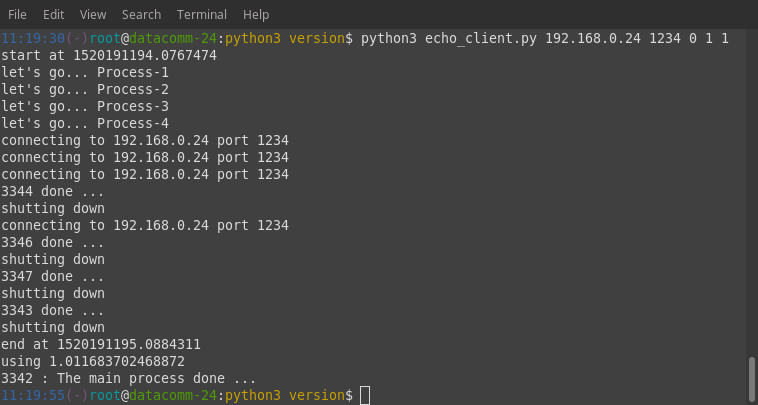
## 2.2 Test Case Descriptions

### 2.2.1 Test 1

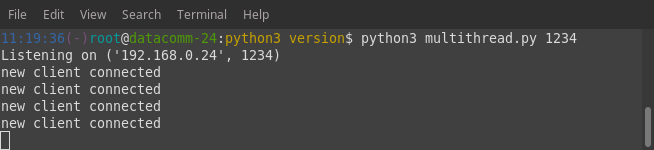
The Multi-thread server successfully receives connection request from clients, and establish connections

# **Run the Python Version:**

The client side:



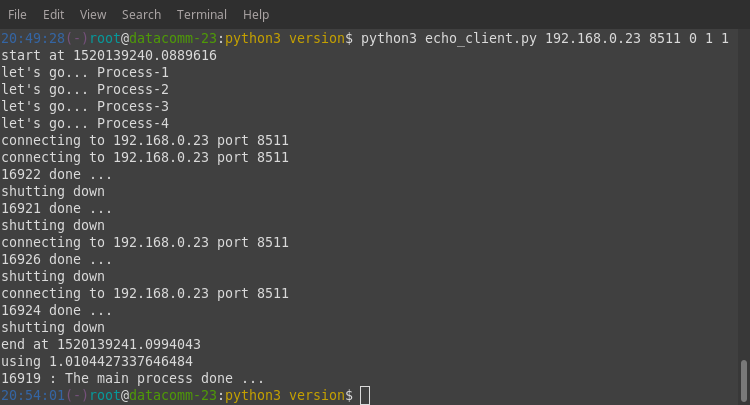
The server side:



All the above we already seen in client test

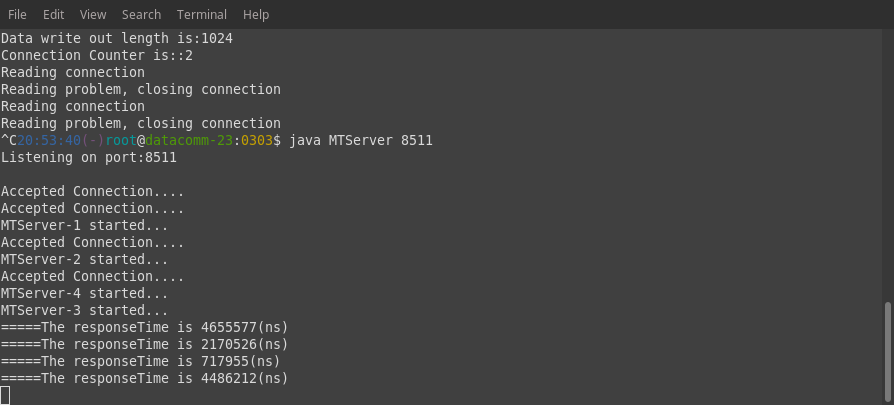
# **Run the Java Version:**

The client side:



I use the python version client to test java version server;

The server side:



Now we see the connection coming.

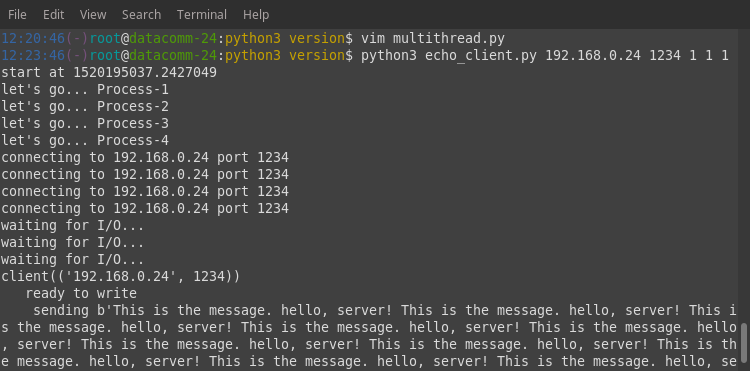
Test pass.

### 2.2.2 Test 2

This Multi-thread server successfully responds to the client.

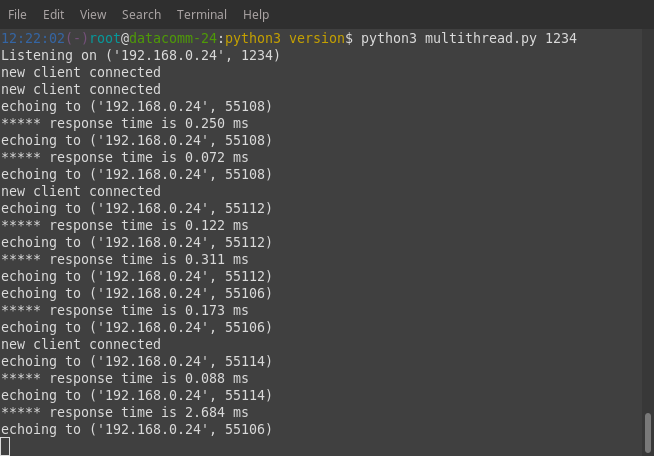
# **Run the Python Version:**

The client side:



Here client generate 4 workers to send data

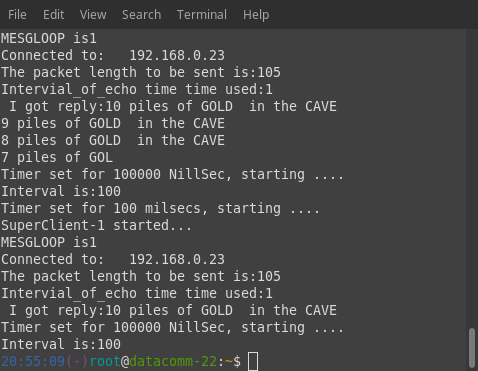
The server side:



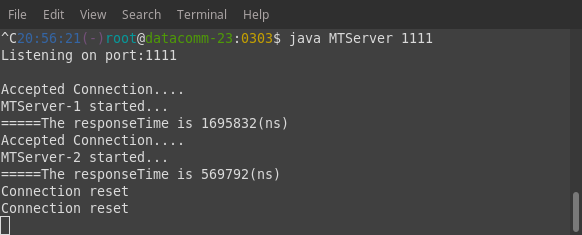
The server side respond to the client, echo back the data they sent;

# **Run the Java Version:**

The client side:



The server side:



We can see there are two threads send data and get respond from server

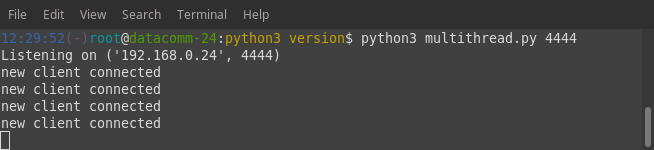
Test passes

### 2.2.3 Test 3

Program successfully listens on the port user specified test

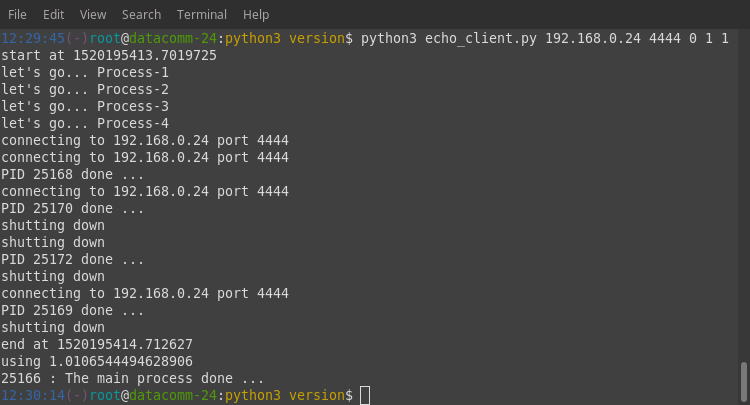
# **Run the Python Version:**

The server side:



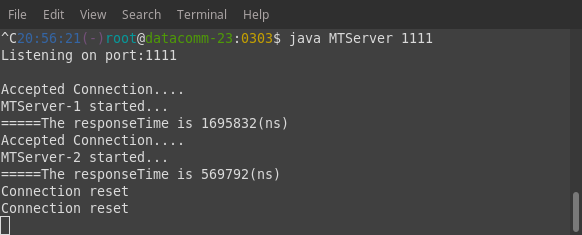
We can see that here I set server port to 4444, and get connections from client.

The client side:



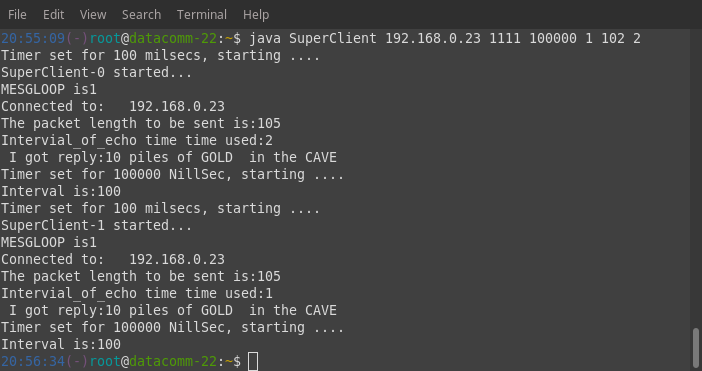
# **Run the Java Version:**

The server side:



We set server port to 1111, and get client connections

The client side:



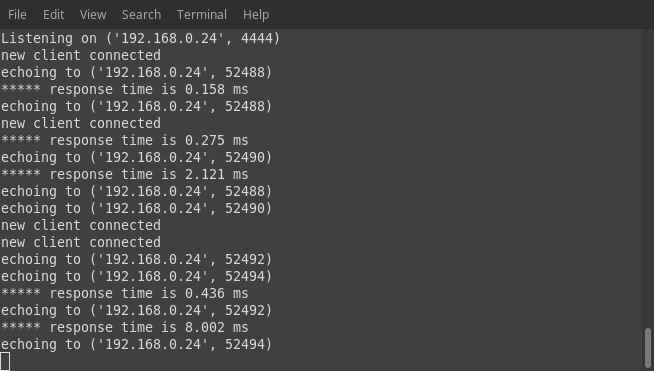
Test passes.

### 2.2.4 Test 4

Server logs print expected information.

# **Run the Python Version:**

The server side:



In python version, server print out the client info and respond time of each request

Test passes.

# Select Mode Server Testing

## 3.1 Test Outline

| Rule # | Test Description | Tool Used | Expected Results | Pass/Fail |
| --- | --- | --- | --- | --- |
| 1 | Program successfully receives connection request from clients. | Sever/Client  application. | When server receive connection request, it would picks it up. | Pass. Detailed results are attached. |
| 2 | Program successfully responds to the client | Sever/Client  application. | The byte size sent to Server. See  the server is the test case same size of message sent back. | Pass. Detailed results are attached. |
| 3 | Program successfully listens on the port user specified test | Sever/Client  application. | Change a port to start the service and make client side change the destination port number | Pass. Detailed results are attached. |
| 4 | Server logs print expected information | Sever/Client  application. | Log shows several level of log data. | Pass. Detailed results are attached. |

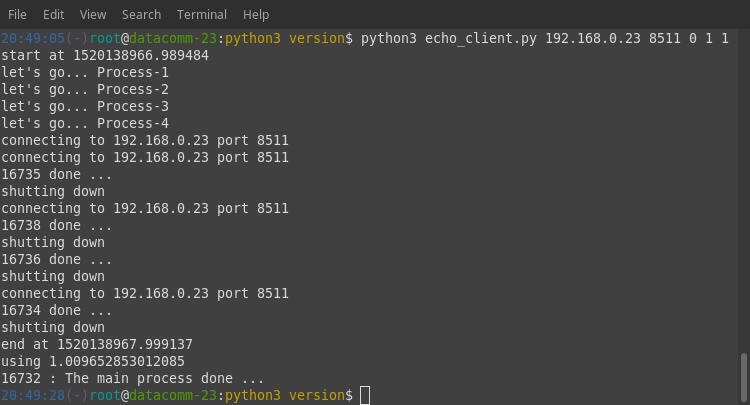
## 3.2 Test Case Descriptions

### 3.2.1 Test 1

The select mode server successfully receives connection request from clients, and establish connections

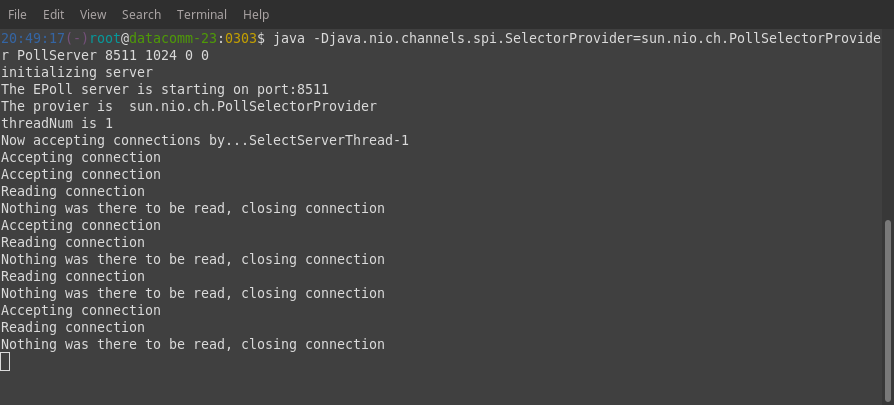
# **Run the Java Version:**

The client side:



Here I use python client connect java server

The server side:



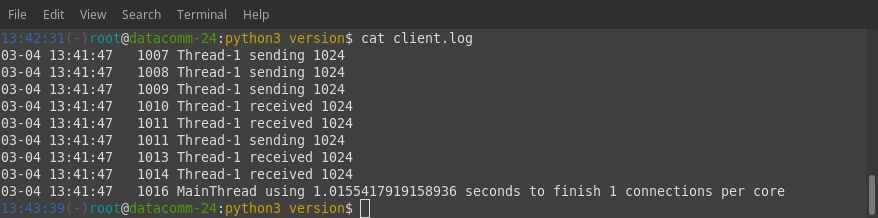
Test passes, since the server establish the connections successfully.

### 3.2.2 Test 2

This select mode server successfully responds to the client.

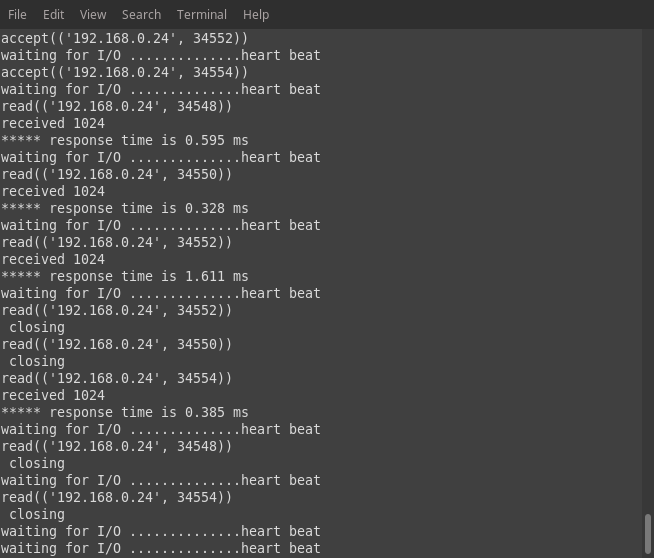
# **Run the Python Version:**

The client side:



This is from client log, show the information of client sending and receiving.

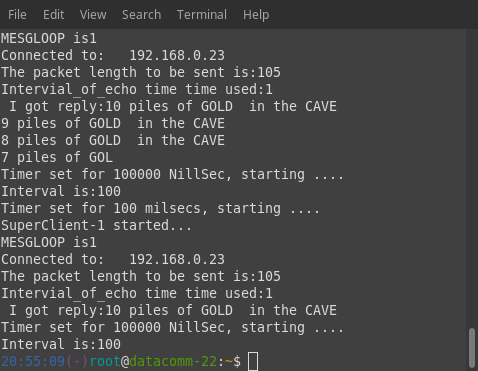
The server side:



Server side shows the respond to each request.

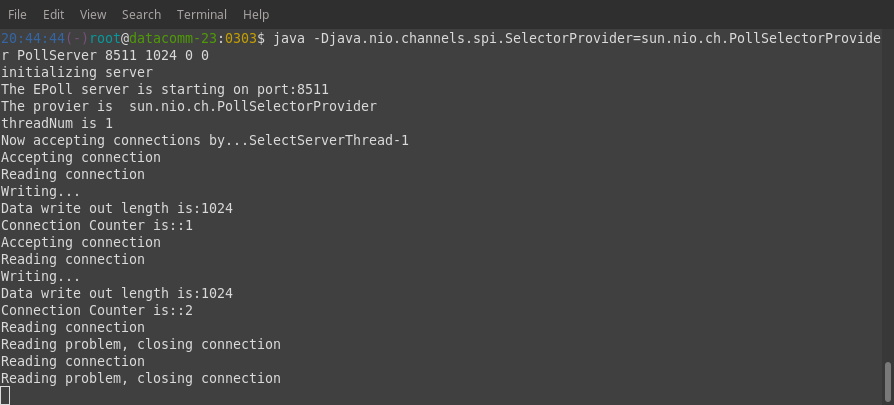
# **Run the Java Version:**

The client side:



The client generate two threads, and each thread send a packet to server;

The server side:



We can see the server received the two packet and echo back then close the connection.

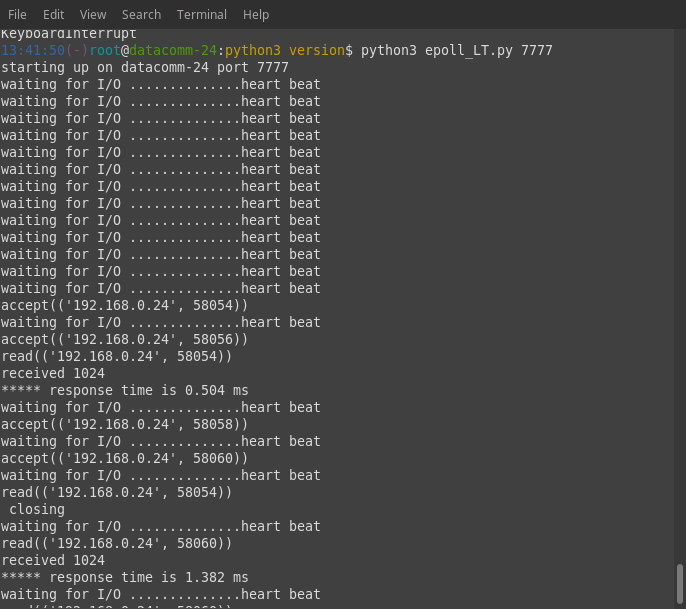
Test passes.

### 3.2.3 Test 3

Program successfully listens on the port user specified test

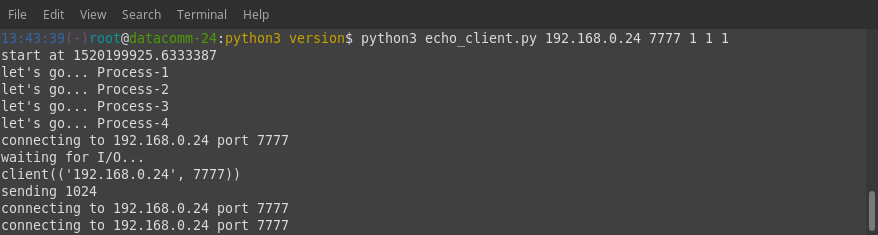
# **Run the Python Version:**

The server side:



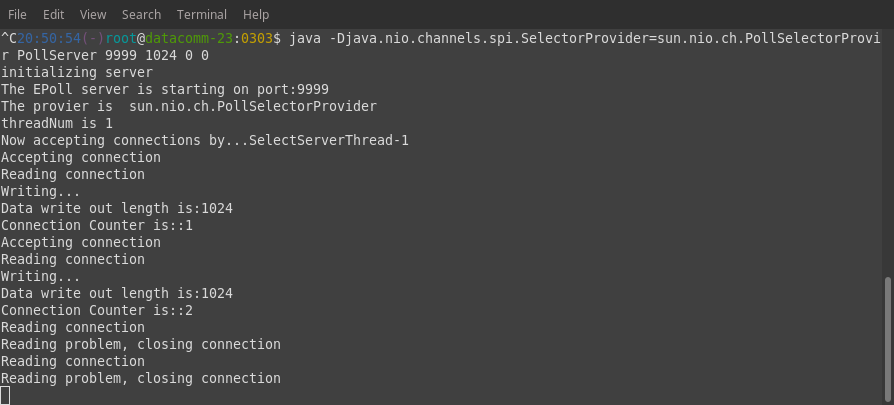
Here I set the server port to 7777, and clients connect successfully

The client side:



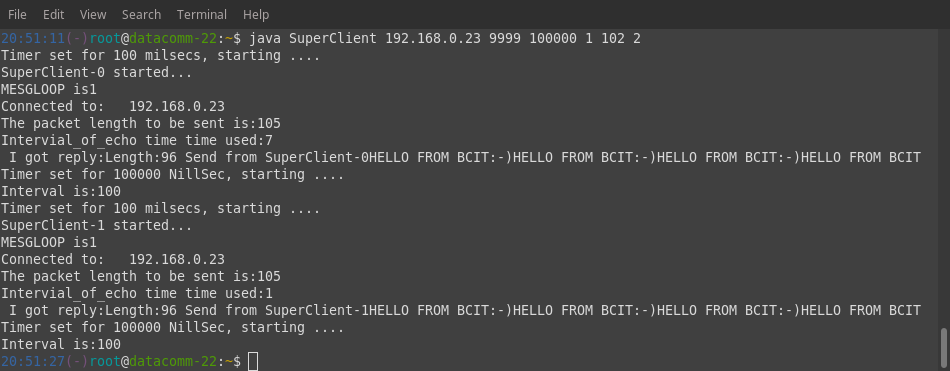
# **Run the Java Version:**

The server side:



Here we set the select server port to 9999

The client side:



The clients successfully connect to port 9999.

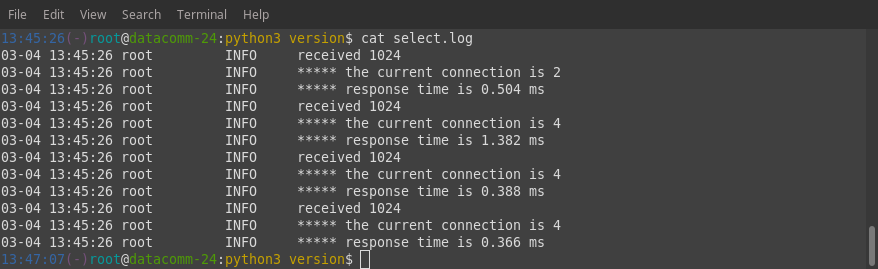
Test passes.

### 3.2.4 Test 4

Server logs print expected information.

# **Run the Python Version:**

The server side:



The select mode server log contains the data length it received and connections it hold, the respond time for each request.

Test passes.

# Epoll Mode Server Testing

## 4.1 Test Outline

| Rule # | Test Description | Tool Used | Expected Results | Pass/Fail |
| --- | --- | --- | --- | --- |
| 1 | Program successfully receives connection request from clients. | Sever/Client  application. | When server receive connection request, it would picks it up. | Pass. Detailed results are attached. |
| 2 | Program successfully responds to the client | Sever/Client  application. | The byte size sent to Server. See  the server is the test case same size of message sent back. | Pass. Detailed results are attached. |
| 3 | Program successfully listens on the port user specified test | Sever/Client  application. | Change a port to start the service and make client side change the destination port number | Pass. Detailed results are attached. |
| 4 | Server logs print expected information | Sever/Client  application. | Log shows several level of log data. | Pass. Detailed results are attached. |

## 4.2 Test Case Descriptions

### 4.2.1 Test 1

The epoll mode server successfully receives connection request from clients, and establish connections

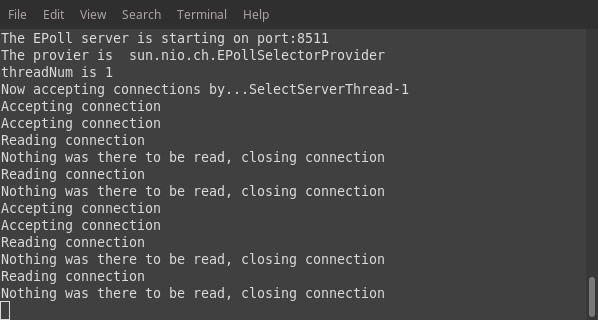
# **Run the Java Version:**

The client side:

### java-epoll-conn-clientside

I use python version client try to connect port 8511, the java version epoll server

The server side:



the server received the requests and established the connection finally disconnect due to no data incoming.

Test passes.

### 4.2.2 Test 2

This Epoll mode server successfully responds to the client.

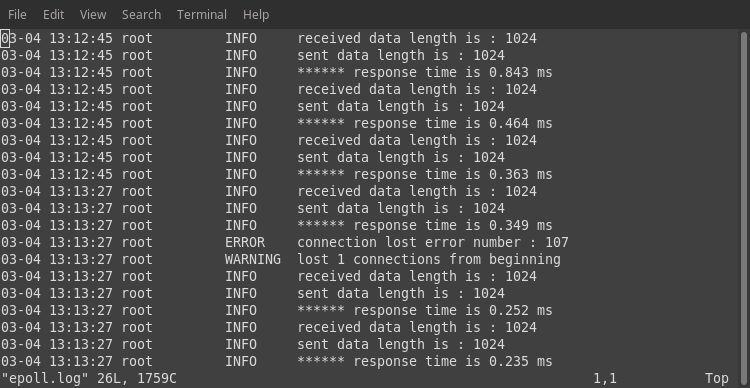
# **Run the Python Version:**

The client side:

### python-epoll-respond-clientside

We can find that this is a client log, it shows that the client sent 4 packets to server

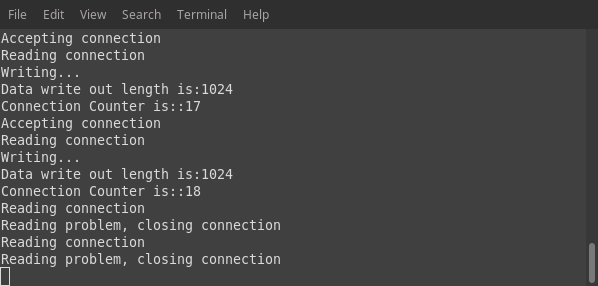
The server side:



From the epoll log file, we can see the responds of the server.

# **Run the Java Version:**

The server side:



This information is printed in the terminal by java version epoll server, it contains data length it echo back and current connections of the server.

Test passes.

### 4.2.3 Test 3

Program successfully listens on the port user specified test

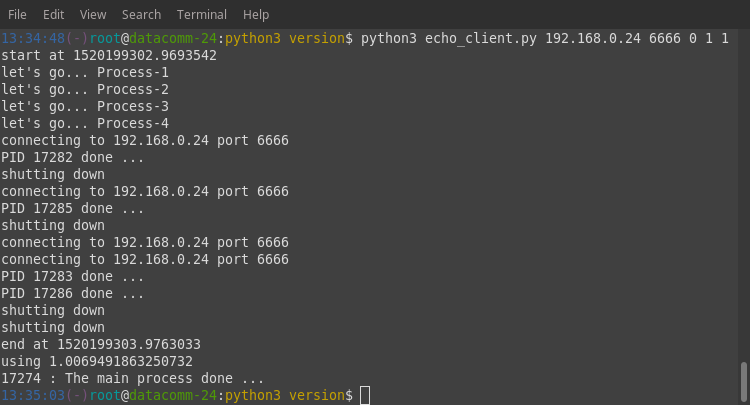
# **Run the Python Version:**

The server side:

### python-epoll-port-serverside

It shows that I set the server port to 6666

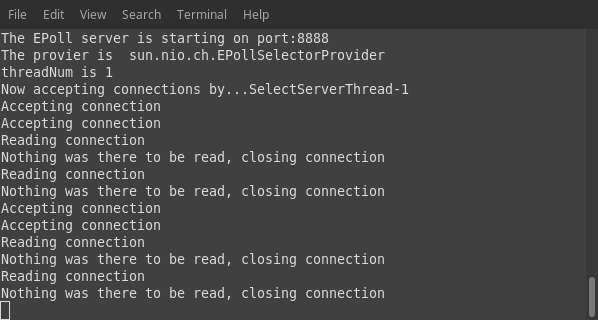
The client side:



The client successfully connect to port 6666.

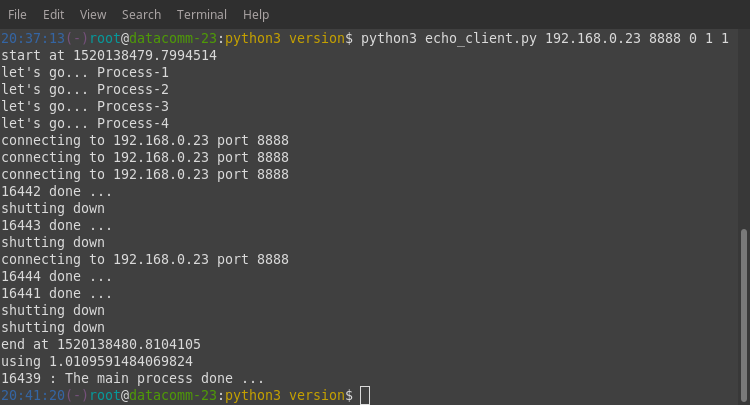
# **Run the Java Version:**

The server side:



Here the server port is set to 8888;

The client side:



The clients successfully connect to server.

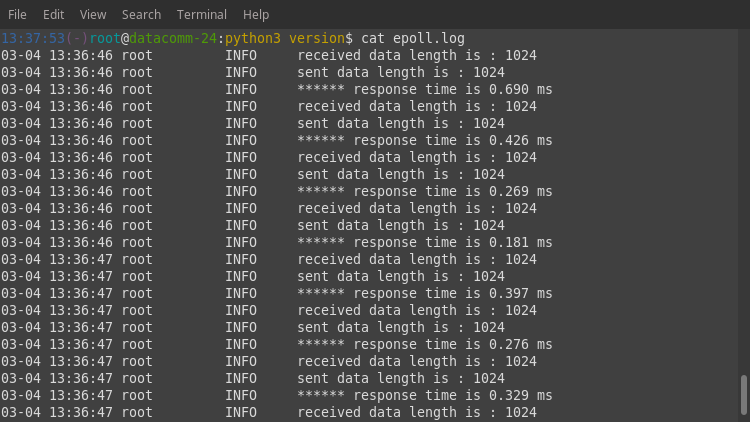
Test passes.

### 4.2.4 Test 4

Server logs print expected information.

# **Run the Python Version:**

The server side:



We can see that the log have the information of the data length each request send and the the length server echo back and the respond time for each request.

Test passes.