

# Socket - Tri-Code Interoperability

## Enhancing Java for High-Performance Server-Client Communication

### Team

Utkarsh Satishkumar Shah  
Sagar Alpeshkumar Patel

### Introduction

The report is for the first mini-project for CMPE-275, and the goal of this report file is to discuss socket programming and to create client-server nodes using C++, Python, and Java languages. Along with that, we are also sharing findings and improvements that we have applied to the provided seed code.

### Interoperability Matrix

Client/Server	Java	Python3	C/C++
Java	Y	Y	Y
Python3	Y	Y	Y
C/C++	Y	Y	Y

# Snapshots for interoperability

## Java Server and Java Client

```
-----< first:mini.proj >-----
] Building mini.proj 1.0-SNAPSHOT
  from pom.xml
-----[ jar ]-----

] --- resources:3.3.1:resources (default-resources) @ mini.proj ---
- skip non existing resourceDirectory /home/utk/NetBeansProjects/mini.proj/src/main/resources

] --- compiler:3.11.0:compile (default-compile) @ mini.proj ---
- Nothing to compile - all classes are up to date

] --- exec:3.1.0:exec (default-cli) @ mini.proj ---
Feb 18, 2024 11:07:55 PM socket.BasicClient connect
INFO: Connected to 127.0.0.1

enter message ('exit' to quit): I am a client in Java
Received acknowledgement from server: Acknowledged

enter message ('exit' to quit): Hola
Received acknowledgement from server: Acknowledged

enter message ('exit' to quit): Good Bye
Received acknowledgement from server: Acknowledged

- enter message ('exit' to quit): exit
-----
BUILD SUCCESS
-----
```



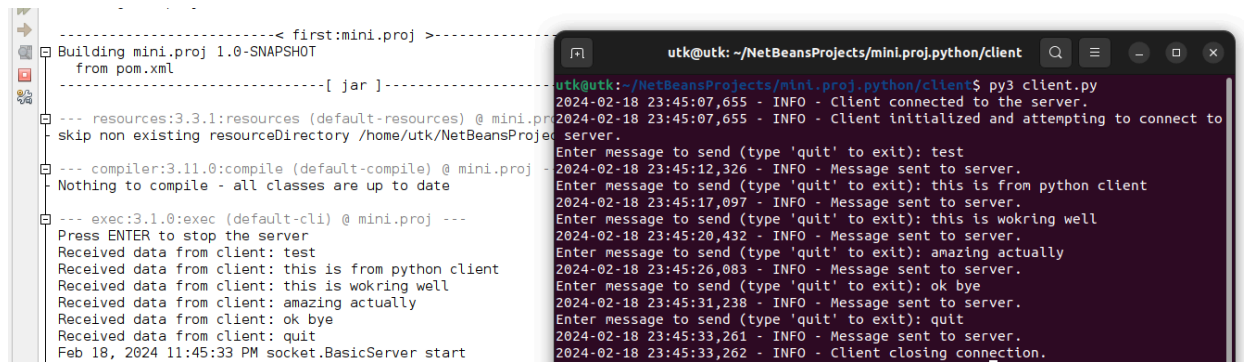
```
-----< first:mini.proj >-----
[ Building mini.proj 1.0-SNAPSHOT
  from pom.xml
-----[ jar ]-----

[ --- resources:3.3.1:resources (default-resources) @ mini.proj ---
- skip non existing resourceDirectory /home/utk/NetBeansProjects/mini.proj/src/main/resources

[ --- compiler:3.11.0:compile (default-compile) @ mini.proj ---
- Nothing to compile - all classes are up to date

[ --- exec:3.1.0:exec (default-cli) @ mini.proj ---
Press ENTER to stop the server
Received data from client: group_chat,app,I am a client in Java
Received data from client: group_chat,app,Hola
Received data from client: group_chat,app,Good Bye
```

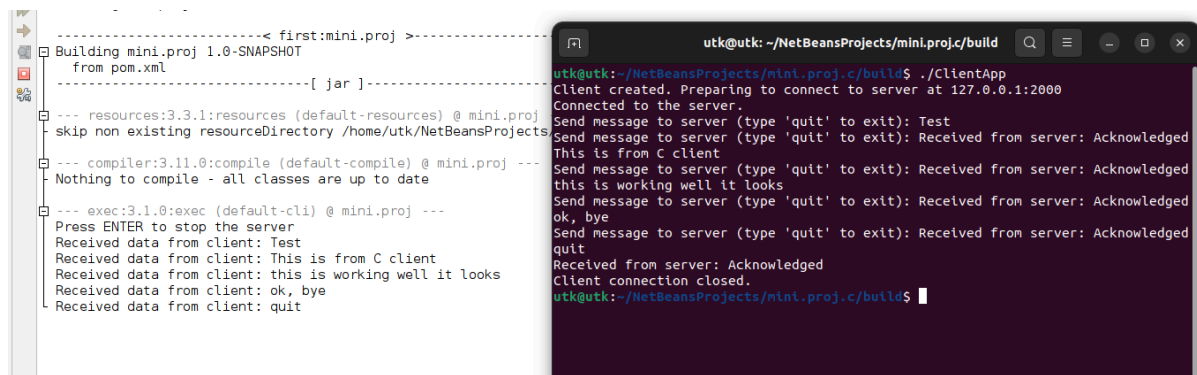
## Java Server and Python Client



```
-----< first:mini.proj >-----
Building mini.proj 1.0-SNAPSHOT
from pom.xml
-----[ jar ]-----
--- resources:3.3.1:resources (default-resources) @ mini.proj ---
skip non existing resourceDirectory /home/utk/NetBeansProjects/mini.proj
--- compiler:3.11.0:compile (default-compile) @ mini.proj ---
Nothing to compile - all classes are up to date
--- exec:3.1.0:exec (default-cli) @ mini.proj ---
Press ENTER to stop the server
Received data from client: test
Received data from client: this is from python client
Received data from client: this is wokring well
Received data from client: amazing actually
Received data from client: ok bye
Received data from client: quit
Feb 18, 2024 11:45:33 PM socket.BasicServer start

utk@utk: ~/NetBeansProjects/mini.proj.python/client
utk@utk:~/NetBeansProjects/mini.proj.python/client$ py3 client.py
2024-02-18 23:45:07,655 - INFO - Client connected to the server.
2024-02-18 23:45:07,655 - INFO - Client initialized and attempting to connect to
server.
Enter message to send (type 'quit' to exit): test
2024-02-18 23:45:12,326 - INFO - Message sent to server.
Enter message to send (type 'quit' to exit): this is from python client
2024-02-18 23:45:17,097 - INFO - Message sent to server.
Enter message to send (type 'quit' to exit): this is wokring well
2024-02-18 23:45:20,432 - INFO - Message sent to server.
Enter message to send (type 'quit' to exit): amazing actually
2024-02-18 23:45:26,083 - INFO - Message sent to server.
Enter message to send (type 'quit' to exit): ok bye
2024-02-18 23:45:31,238 - INFO - Message sent to server.
Enter message to send (type 'quit' to exit): quit
2024-02-18 23:45:33,261 - INFO - Message sent to server.
2024-02-18 23:45:33,262 - INFO - Client closing connection.
```

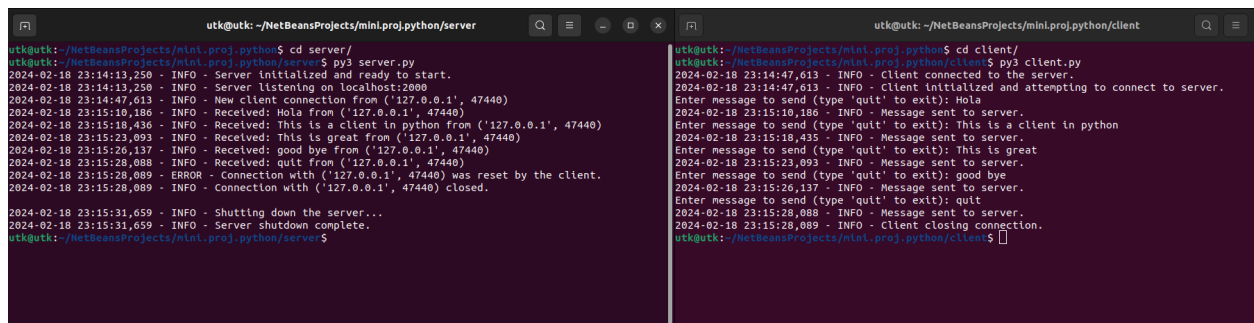
## Java Server and C/C++ Client



```
-----< first:mini.proj >-----
Building mini.proj 1.0-SNAPSHOT
from pom.xml
-----[ jar ]-----
--- resources:3.3.1:resources (default-resources) @ mini.proj ---
skip non existing resourceDirectory /home/utk/NetBeansProjects/mini.proj
--- compiler:3.11.0:compile (default-compile) @ mini.proj ---
Nothing to compile - all classes are up to date
--- exec:3.1.0:exec (default-cli) @ mini.proj ---
Press ENTER to stop the server
Received data from client: Test
Received data from client: This is from C client
Received data from client: this is working well it looks
Received data from client: ok, bye
Received data from client: quit

utk@utk: ~/NetBeansProjects/mini.proj.c/build
utk@utk:~/NetBeansProjects/mini.proj.c/build$ ./ClientApp
Client created. Preparing to connect to server at 127.0.0.1:2000
Connected to the server.
Send message to server (type 'quit' to exit): Test
Send message to server (type 'quit' to exit): Received from server: Acknowledged
This is from C client
Send message to server (type 'quit' to exit): Received from server: Acknowledged
this is working well it looks
Send message to server (type 'quit' to exit): Received from server: Acknowledged
ok, bye
Send message to server (type 'quit' to exit): Received from server: Acknowledged
quit
Received from server: Acknowledged
Client connection closed.
utk@utk:~/NetBeansProjects/mini.proj.c/build$
```

## Python Server and Python Client



```
utk@utk: ~/NetBeansProjects/mini.proj.python/server
utk@utk:~/NetBeansProjects/mini.proj.python$ cd server/
utk@utk:~/NetBeansProjects/mini.proj.python/server$ py3 server.py
2024-02-18 23:14:13,250 - INFO - Server Initialized and ready to start.
2024-02-18 23:14:13,250 - INFO - Server listening on localhost:2000
2024-02-18 23:14:47,613 - INFO - New client connection from ('127.0.0.1', 47440)
2024-02-18 23:15:10,186 - INFO - Received: Hola from ('127.0.0.1', 47440)
2024-02-18 23:15:18,436 - INFO - Received: This is a client in python from ('127.0.0.1', 47440)
2024-02-18 23:15:23,093 - INFO - Received: This is great from ('127.0.0.1', 47440)
2024-02-18 23:15:26,137 - INFO - Received: good bye from ('127.0.0.1', 47440)
2024-02-18 23:15:28,088 - INFO - Received: quit from ('127.0.0.1', 47440)
2024-02-18 23:15:28,089 - ERROR - Connection with ('127.0.0.1', 47440) was reset by the client.
2024-02-18 23:15:28,089 - INFO - Connection with ('127.0.0.1', 47440) closed.
2024-02-18 23:15:31,659 - INFO - Shutting down the server...
2024-02-18 23:15:31,659 - INFO - Server shutdown complete.
utk@utk:~/NetBeansProjects/mini.proj.python/server$

utk@utk:~/NetBeansProjects/mini.proj.python/client
utk@utk:~/NetBeansProjects/mini.proj.python$ cd client/
utk@utk:~/NetBeansProjects/mini.proj.python/client$ py3 client.py
2024-02-18 23:14:47,613 - INFO - Client connected to the server.
2024-02-18 23:14:47,613 - INFO - Client initialized and attempting to connect to server.
Enter message to send (type 'quit' to exit): Hola
2024-02-18 23:15:10,186 - INFO - Message sent to server.
Enter message to send (type 'quit' to exit): This is a client in python
2024-02-18 23:15:18,435 - INFO - Message sent to server.
Enter message to send (type 'quit' to exit): This is great
2024-02-18 23:15:23,093 - INFO - Message sent to server.
Enter message to send (type 'quit' to exit): good bye
2024-02-18 23:15:26,137 - INFO - Message sent to server.
Enter message to send (type 'quit' to exit): quit
2024-02-18 23:15:28,088 - INFO - Message sent to server.
2024-02-18 23:15:28,089 - INFO - Client closing connection.
utk@utk:~/NetBeansProjects/mini.proj.python/client$
```

# Python Server and Java Client

The screenshot shows two windows in the NetBeans IDE. The left window, titled 'utk@utk: ~/NetBeansProjects/mini.proj.python/server', displays the output of a Python server script. The right window, titled 'mini.proj - Apache NetBeans IDE 20', shows the output of a Java client application.

```
utk@utk: ~/NetBeansProjects/mini.proj.python/server$ py3 server.py
2024-02-18 23:46:45,489 - INFO - Server initialized and ready to start.
2024-02-18 23:46:45,489 - INFO - Server listening on localhost:2000
2024-02-18 23:46:59,331 - INFO - New client connection from ('127.0.0.1', 46062)
2024-02-18 23:47:09,514 - INFO - Received: group_chat,app,Test from ('127.0.0.1', 46062)
2024-02-18 23:47:16,228 - INFO - Received: group_chat,app,This is from Java client from ('127.0.0.1', 46062)
2024-02-18 23:47:21,343 - INFO - Received: group_chat,app,this is working well from ('127.0.0.1', 46062)
2024-02-18 23:47:29,865 - INFO - Received: group_chat,app,I am pleased. from ('127.0.0.1', 46062)
2024-02-18 23:47:31,971 - INFO - Received: group_chat,app,bye from ('127.0.0.1', 46062)
2024-02-18 23:47:35,083 - INFO - Connection with ('127.0.0.1', 46062) closed.
```

```
mini.proj - Apache NetBeans IDE 20
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help Search (Ctrl+F)
Output - Run (ClientApp)
< first:mini.proj >-----
Building mini.proj 1.0-SNAPSHOT
from pom.xml
[ jar ]-----
resources:3.3.1:resources (default-resources) @ mini.proj ---
skip non existing resourceDirectory /home/utk/NetBeansProjects/mini.proj/src/main/resources
compiler:3.11.0:compile (default-compile) @ mini.proj ---
Nothing to compile - all classes are up to date
exec:3.1.0:exec (default-cli) @ mini.proj ---
Feb 18, 2024 11:46:59 PM socket.BasicClient connect
INFO: Connected to 127.0.0.1

enter message ('exit' to quit): Test
Received acknowledgement from server: group_chat,app,Test

enter message ('exit' to quit): This is from Java client
Received acknowledgement from server: group_chat,app,This is from Java client

enter message ('exit' to quit): this is working well
Received acknowledgement from server: group_chat,app,this is working well

enter message ('exit' to quit): I am pleased.
Received acknowledgement from server: group_chat,app,I am pleased.

enter message ('exit' to quit): bye
Received acknowledgement from server: group_chat,app,bye

enter message ('exit' to quit): exit
BUILD SUCCESS
Total time: 36.223 s
Finished at: 2024-02-18T23:47:35:08:00
```

# Python Server and C/C++ Client

The screenshot shows two windows in the NetBeans IDE. The left window, titled 'utk@utk: ~/NetBeansProjects/mini.proj.python/server', displays the output of a Python server script. The right window, titled 'utk@utk: ~/NetBeansProjects/mini.proj.c/build', displays the output of a C/C++ client application.

```
utk@utk: ~/NetBeansProjects/mini.proj.python/server$ py3 server.py
2024-02-18 23:18:55,044 - INFO - Server initialized and ready to start.
2024-02-18 23:18:55,044 - INFO - Server listening on localhost:2000
2024-02-18 23:19:20,763 - INFO - New client connection from ('127.0.0.1', 57374)
2024-02-18 23:19:23,466 - INFO - Received: Test from ('127.0.0.1', 57374)
2024-02-18 23:19:29,675 - INFO - Received: This is working from ('127.0.0.1', 57374)
2024-02-18 23:19:35,056 - INFO - Received: It is Amazing from ('127.0.0.1', 57374)
2024-02-18 23:19:39,610 - INFO - Received: Lets quit from ('127.0.0.1', 57374)
2024-02-18 23:19:40,879 - INFO - Received: quit from ('127.0.0.1', 57374)
2024-02-18 23:19:40,880 - INFO - Connection with ('127.0.0.1', 57374) closed.
```

```
utk@utk: ~/NetBeansProjects/mini.proj.c/build$ ./ClientApp
Client created. Preparing to connect to server at 127.0.0.1:2000
Connected to the server.
Send message to server (type 'quit' to exit): Test
Send message to server (type 'quit' to exit): Received from server: Test

This is working
Send message to server (type 'quit' to exit): Received from server: This is working

It is Amazing
Send message to server (type 'quit' to exit): Received from server: It is Amazing

Lets quit
Send message to server (type 'quit' to exit): Received from server: Lets quit

quit
Received from server: quit

Client connection closed.
utk@utk: ~/NetBeansProjects/mini.proj.c/build$
```

# C/C++ Server and C/C++ Client

The screenshot shows two windows in the NetBeans IDE. The left window, titled 'utk@utk: ~/NetBeansProjects/mini.proj.c/build', displays the output of a C/C++ server script. The right window, titled 'utk@utk: ~/NetBeansProjects/mini.proj.c/build', displays the output of a C/C++ client application.

```
utk@utk: ~/NetBeansProjects/mini.proj.c/build$ ./ServerApp
Starting server...
Server is listening on port 2000
Accepted a client.
Received from client: C test

Received from client: Hola

Received from client: This is working

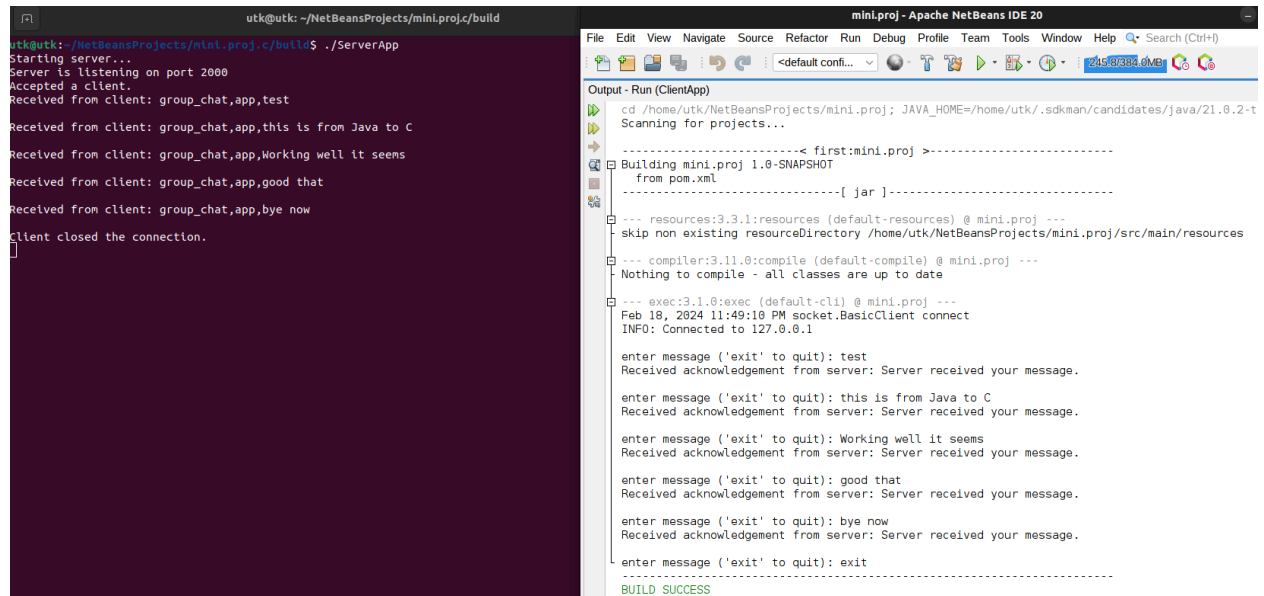
Received from client: good bye

Received from client: quit

Client closed the connection.
```

```
utk@utk: ~/NetBeansProjects/mini.proj.c/build$ ./ClientApp
Client created. Preparing to connect to server at 127.0.0.1:2000
Connected to the server.
Send message to server (type 'quit' to exit): C test
Send message to server (type 'quit' to exit): Received from server: Server received your mess
Hola
Send message to server (type 'quit' to exit): Received from server: Server received your mess
This is working
Send message to server (type 'quit' to exit): Received from server: Server received your mess
good bye
Send message to server (type 'quit' to exit): Received from server: Server received your mess
quit
Received from server: Server received your message.
Client connection closed.
utk@utk: ~/NetBeansProjects/mini.proj.c/build$
```

## C/C++ Server and Java Client



The screenshot shows the NetBeans IDE interface. On the left, a terminal window displays the output of the C/C++ server application. On the right, the IDE's output window shows the build and execution details of the Java client application.

```
utk@utk: ~/NetBeansProjects/mini.proj.c/build
utk@utk:~/NetBeansProjects/mini.proj.c/build$ ./ServerApp
Starting server...
Server is listening on port 2000
Accepted a client.
Received from client: group_chat,app,test
Received from client: group_chat,app,this is from Java to C
Received from client: group_chat,app,Working well it seems
Received from client: group_chat,app,good that
Received from client: group_chat,app,bye now
Client closed the connection.
```

```
mini.proj - Apache NetBeans IDE 20
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help Search (Ctrl+I)
Output - Run (ClientApp)
cd /home/utk/NetBeansProjects/mini.proj; JAVA_HOME=/home/utk/.sdkman/candidates/java/21.0.2-t
Scanning for projects...
-----< first:mini.proj >-----
Building mini.proj 1.0-SNAPSHOT
from pom.xml
-----[ jar ]-----
--- resources:3.3.1:resources (default-resources) @ mini.proj ---
skip non existing resourceDirectory /home/utk/NetBeansProjects/mini.proj/src/main/resources
--- compiler:3.11.0:compile (default-compile) @ mini.proj ---
Nothing to compile - all classes are up to date
--- exec:3.1.0:exec (default-cli) @ mini.proj ---
Feb 18, 2024 11:49:10 PM socket.BasicClient connect
INFO: Connected to 127.0.0.1

enter message ('exit' to quit): test
Received acknowledgement from server: Server received your message.

enter message ('exit' to quit): this is from Java to C
Received acknowledgement from server: Server received your message.

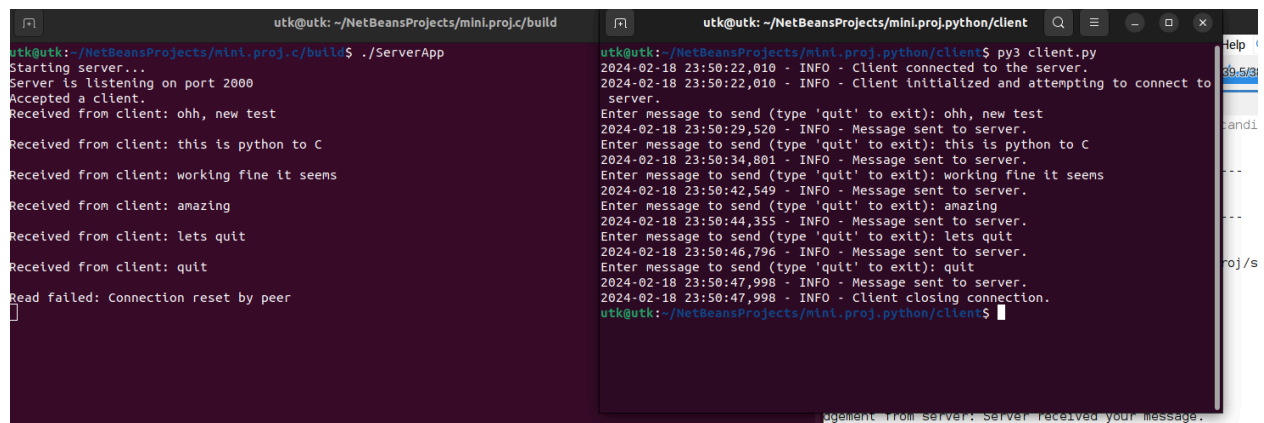
enter message ('exit' to quit): Working well it seems
Received acknowledgement from server: Server received your message.

enter message ('exit' to quit): good that
Received acknowledgement from server: Server received your message.

enter message ('exit' to quit): bye now
Received acknowledgement from server: Server received your message.

enter message ('exit' to quit): exit
BUILD SUCCESS
```

## C/C++ Server and Python Client



The screenshot shows the NetBeans IDE interface. On the left, a terminal window displays the output of the C/C++ server application. On the right, a terminal window displays the output of the Python client application.

```
utk@utk: ~/NetBeansProjects/mini.proj.c/build
utk@utk:~/NetBeansProjects/mini.proj.c/build$ ./ServerApp
Starting server...
Server is listening on port 2000
Accepted a client.
Received from client: ohh, new test
Received from client: this is python to C
Received from client: working fine it seems
Received from client: amazing
Received from client: lets quit
Received from client: quit
Read failed: Connection reset by peer
```

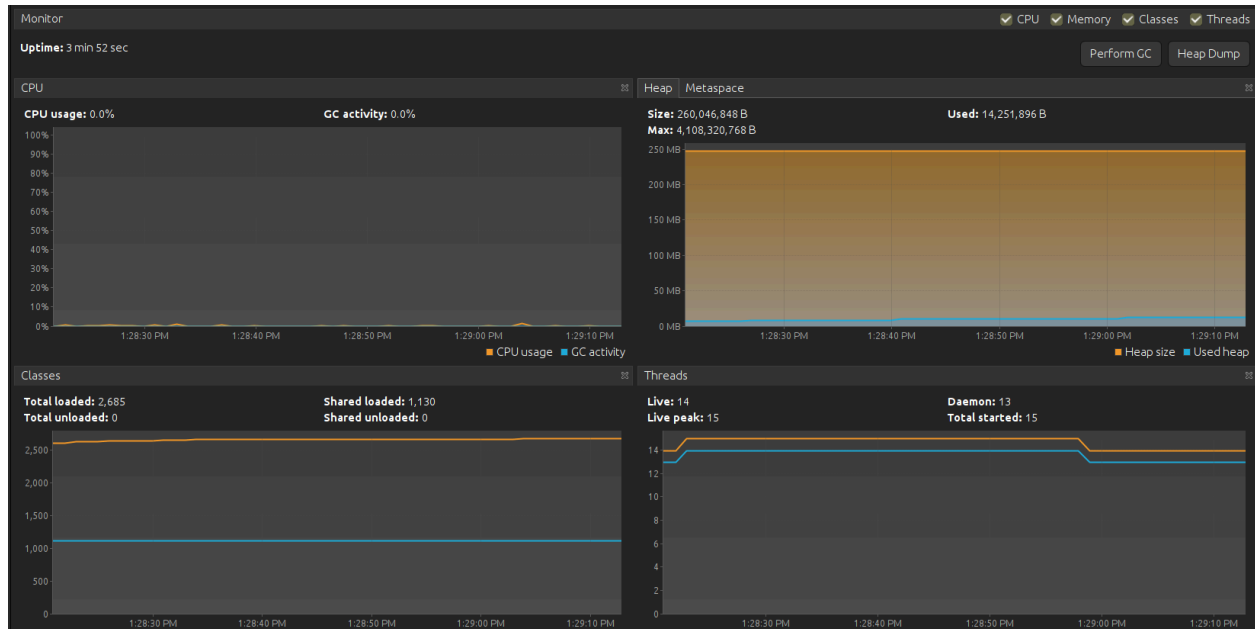
```
utk@utk: ~/NetBeansProjects/mini.proj.python/client
utk@utk:~/NetBeansProjects/mini.proj.python/client$ py3 client.py
2024-02-18 23:50:22,010 - INFO - Client connected to the server.
2024-02-18 23:50:22,010 - INFO - Client initialized and attempting to connect to
server.
Enter message to send (type 'quit' to exit): ohh, new test
2024-02-18 23:50:29,520 - INFO - Message sent to server.
Enter message to send (type 'quit' to exit): this is python to C
2024-02-18 23:50:34,801 - INFO - Message sent to server.
Enter message to send (type 'quit' to exit): working fine it seems
2024-02-18 23:50:42,549 - INFO - Message sent to server.
Enter message to send (type 'quit' to exit): amazing
2024-02-18 23:50:44,355 - INFO - Message sent to server.
Enter message to send (type 'quit' to exit): lets quit
2024-02-18 23:50:46,796 - INFO - Message sent to server.
Enter message to send (type 'quit' to exit): quit
2024-02-18 23:50:47,998 - INFO - Message sent to server.
2024-02-18 23:50:47,998 - INFO - Client closing connection.
utk@utk:~/NetBeansProjects/mini.proj.python/client$
```

# Java Findings and Optimization

## First Attempt

One Server and One Client Communication

VisualVM Profiling for CPU utilization, Heapspace, and GC activity



## Server output and timetaken

```
--- exec:3.1.0:exec (default-cli) @ mini.proj ---
Server Host: 0.0.0.0
Press ENTER to stop the server
--> server got a client connection
Session 21 started
from app, to group: pets/dogs, text: Hi there
from app, to group: pets/dogs, text: Hope it is all going well
from app, to group: pets/dogs, text: it is good to see you again
from app, to group: pets/dogs, text: Gotta go, see you soon.
from app, to group: pets/dogs, text: Good bye
Session 21 ending

java.net.SocketException: Socket closed
  at java.base/sun.nio.ch.NioSocketImpl.endAccept(NioSocketImpl.java:682)
  at java.base/sun.nio.ch.NioSocketImpl.accept(NioSocketImpl.java:755)
  at java.base/java.net.ServerSocket.implAccept(ServerSocket.java:698)
  at java.base/java.net.ServerSocket.platformImplAccept(ServerSocket.java:663)
  at java.base/java.net.ServerSocket.implAccept(ServerSocket.java:639)
  at java.base/java.net.ServerSocket.implAccept(ServerSocket.java:585)
  at java.base/java.net.ServerSocket.accept(ServerSocket.java:543)
  at socket.BasicServer.start(BasicServer.java:33)
  at java.base/java.lang.Thread.run(Thread.java:1583)

Total requests: 1
Average processing time: 28627725314 ns
-----
```

## Findings and Improvements

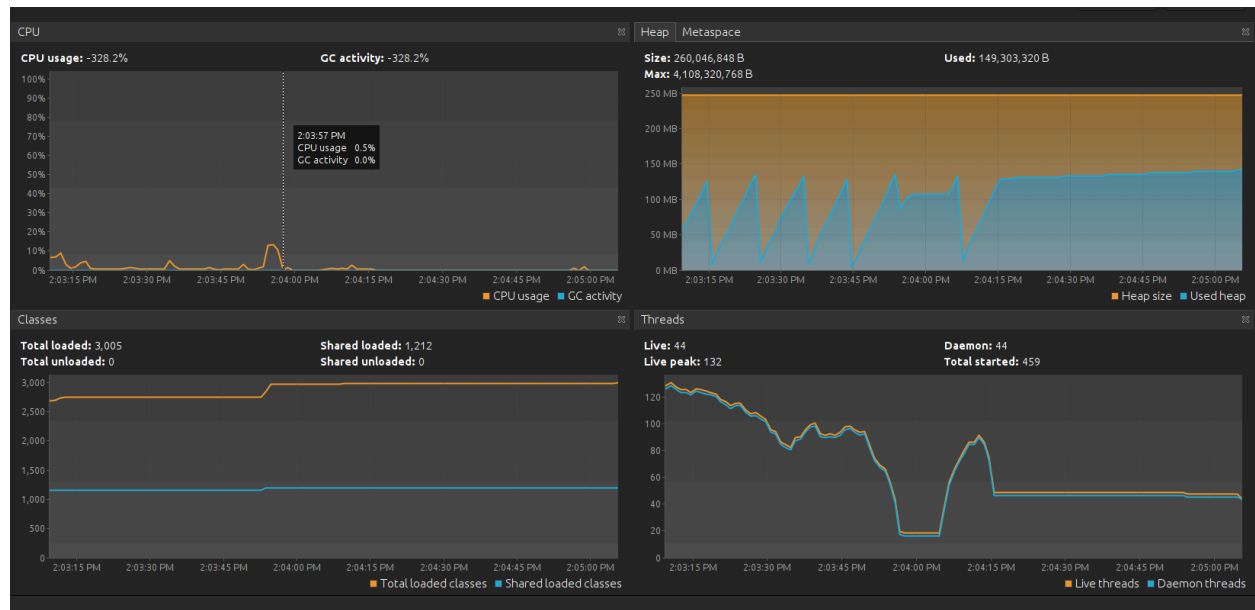
First effort is to add some kind of testing code so that we can measure the performance as we move on to improve the code quality and remove bottlenecks. It is applied on the provided raw seedcode which is suboptimal to handle sessions and socket communication.

Running into issues while executing the test

Client - 1000

Messages per client - 10000

```
java.net.SocketException: Broken pipe
    at java.base/sun.nio.ch.SocketDispatcher.write0(Native Method)
    at java.base/sun.nio.ch.SocketDispatcher.write(SocketDispatcher.java:62)
    at java.base/sun.nio.ch.NioSocketImpl.tryWrite(NioSocketImpl.java:394)
    at java.base/sun.nio.ch.NioSocketImpl.implWrite(NioSocketImpl.java:410)
    at java.base/sun.nio.ch.NioSocketImpl.write(NioSocketImpl.java:440)
    at java.base/sun.nio.ch.NioSocketImpl$2.write(NioSocketImpl.java:819)
    at java.base/java.net.Socket$SocketOutputStream.write(Socket.java:1195)
    at java.base/java.io.OutputStream.write(OutputStream.java:124)
    at socket.BasicClient.sendMessage(BasicClient.java:70)
    at app.TestClient.run(TestClient.java:19)
    at java.base/java.lang.Thread.run(Thread.java:1583)
java.net.SocketException: Broken pipe
    at java.base/sun.nio.ch.SocketDispatcher.write0(Native Method)
    at java.base/sun.nio.ch.SocketDispatcher.write(SocketDispatcher.java:62)
    at java.base/sun.nio.ch.NioSocketImpl.tryWrite(NioSocketImpl.java:394)
    at java.base/sun.nio.ch.NioSocketImpl.implWrite(NioSocketImpl.java:410)
    at java.base/sun.nio.ch.NioSocketImpl.write(NioSocketImpl.java:440)
    at java.base/sun.nio.ch.NioSocketImpl$2.write(NioSocketImpl.java:819)
    at java.base/java.net.Socket$SocketOutputStream.write(Socket.java:1195)
    at java.base/java.io.OutputStream.write(OutputStream.java:124)
    at socket.BasicClient.sendMessage(BasicClient.java:70)
    at app.TestClient.run(TestClient.java:19)
    at java.base/java.lang.Thread.run(Thread.java:1583)
```



Encountered error "Broken pipe"

java.net.SocketException: Broken pipe at java.base/sun.nio.ch.SocketDispatcher.write0(Native Method) at java.base/sun.nio.ch.SocketDispatcher.write(SocketDispatcher.java:62) at java.base/sun.nio.ch.NioSocketImpl.tryWrite(NioSocketImpl.java:394) at

```
java.base/sun.nio.ch.NioSocketImpl.implWrite(NioSocketImpl.java:410) at
java.base/sun.nio.ch.NioSocketImpl.write(NioSocketImpl.java:440) at
java.base/sun.nio.ch.NioSocketImpl$2.write(NioSocketImpl.java:819) at
java.base/java.net.Socket$SocketOutputStream.write(Socket.java:1195) at
java.base/java.io.OutputStream.write(OutputStream.java:124) at
socket.BasicClient.sendMessage(BasicClient.java:70) at app.TestClient.run(TestClient.java:19)
at java.base/java.lang.Thread.run(Thread.java:1583)
```

## **Better exception handling and logging**

We believe the error occurred on the client side since the server is getting overwhelmed with the number of connections or the rate of incoming data, and is closing connections to cope. If this is the case, we might need to optimize the code for session handling.

## **Utilize ThreadPool in Java**

In the current implementation, a new thread is created for each client connection. This can be inefficient and resource-intensive if there are many client connections, as creating and destroying threads can be expensive.

A thread pool can help mitigate this issue. With a thread pool, a fixed number of threads are created and kept alive. When a task is submitted to the thread pool, it is executed by one of the existing threads. This can be more efficient than creating a new thread for each task.

## **Sending 10M Messages**

Even after using ThreadPool, the program was again and again running into the BrokenPipe issue. We believe it is because the server is getting overwhelmed with the number of clients trying to communicate and the amount of messages received.

## **Switching to Non-blocking I/O (NIO)**

Also, with non-blocking I/O, a single thread can handle multiple client connections. This is because when the thread performs an I/O operation, it doesn't block if the operation can't be completed immediately. Instead, it can move on to another task and come back to the operation later when it can be completed. This means you can handle multiple client connections with a single thread, eliminating the need for a thread pool.

The selector in NIO is used to monitor both the server socket channel and all connected client socket channels. When a client connects, the server accepts the connection, sets the client socket to non-blocking mode, and registers it to the selector for read events. When data is ready to be read from a client socket, the server reads the data and processes it. This allows the server to handle multiple client connections using a single thread.



Even after switching to NIO, we still received a BrokenPipe error on the client side while sending 10M messages with 1000 client numbers.

While fixing this we found that we need to implement some form of flow control in our application. One simple way to do this is to have the client wait for an acknowledgment from the server after sending a certain amount of data. Once the acknowledgment is received, the client can continue sending more data. This ensures that the server has a chance to process the incoming data and prevents it from getting overwhelmed.

This solution finally worked and we were able to send 10M requests from multiple clients to the server.

## Bottleneck with 10000 threads

While increasing the number of threads, we again reached another bottleneck, and the communication was interrupted with an exception stating the socket is closed.

```
Feb 19, 2024 11:54:41 AM socket.BasicClient sendMessage
SEVERE: An error occurred while sending message
[ ] java.net.SocketException: Socket is closed
    at java.base/java.net.Socket.getOutputStream(Socket.java:1157)
    at socket.BasicClient.sendMessage(BasicClient.java:72)
    at app.TestClient.run(TestClient.java:24)
    at java.base/java.lang.Thread.run(Thread.java:1583)

Feb 19, 2024 11:54:41 AM socket.BasicClient sendMessage
SEVERE: An error occurred while sending message
[ ] java.net.SocketException: Socket is closed
    at java.base/java.net.Socket.getOutputStream(Socket.java:1157)
    at socket.BasicClient.sendMessage(BasicClient.java:72)
    at app.TestClient.run(TestClient.java:24)
    at java.base/java.lang.Thread.run(Thread.java:1583)
```

## Test observations for the Java application

100 Threads, 1,000 messages

INFO: Processed 100,000 requests

INFO: Total duration time: 1,779,891,827 ns

INFO: Average request processing time: 17,798 ns

100 Threads, 10,000 messages

INFO: Processed 1,000,000 requests

INFO: Total duration time: 14,257,202,868 ns

INFO: Average request processing time: 14,257 ns

100 Threads, 100,000 messages

INFO: Processed 10,000,000 requests

INFO: Total duration time: 165,155,258,498 ns  
INFO: Average request processing time: 16,515 ns

100 Threads, 100 messages  
INFO: Processed 10,000 requests  
INFO: Total duration time: 258,295,330 ns  
INFO: Average request processing time: 25,829 ns

1000 Threads, 100 messages  
INFO: Processed 100,000 requests  
INFO: Total duration time: 1,642,501,560 ns  
INFO: Average request processing time: 16,425 ns

10000 Threads, 100 messages  
INFO: Processed 923,000 requests  
INFO: Total duration time: 13,957,230,646 ns  
INFO: Average request processing time: 15,121 ns

## Citations

<https://www.baeldung.com/a-guide-to-java-sockets>  
<https://mailinator.blogspot.com/2008/02/kill-myth-please-nio-is-not-faster-than.html>  
<https://stackoverflow.com/questions/7611152/nio-performance-improvement-compared-to-traditional-io-in-java>  
<https://medium.com/@tesla8877/java-nio-vs-java-socket-exploring-advantages-and-disadvantages-in-network-communication-65565dd5d2c5>  
<https://www.baeldung.com/java-io-vs-nio>  
<https://s3-eu-central-1.amazonaws.com/ucu.edu.ua/wp-content/uploads/sites/8/2019/12/Petro-Karabyn.pdf>

# Python Findings and Optimization

## Summary of Server, Client

1. `Server.py`: This script sets up a server that listens for incoming client connections. It uses a thread pool to handle multiple clients concurrently. Each client is handled in a separate thread where the server receives messages from the client, logs them, and sends an acknowledgment back to the client. The server keeps track of all active sessions and their messages.

2. `Client.py`: This script sets up a client that connects to the server. It sends messages input by the user to the server and logs the server's responses. The client can send messages continuously until the user types 'quit'.

## Features, Tools, and Libraries

**Import Statements:** The `import` keyword is used to include Python's built-in libraries such as `logging`, `socket`, `concurrent.futures`, and `threading`.

**Logging:** The `logging` module is used to record events happening in the server and client. It's configured with `basicConfig` to set the level and format of the logs.

**Socket Programming:** The `socket` module is used to create a TCP/IP socket, bind it to an address, listen for incoming connections, accept a connection, send data to the client, and receive data from the client.

**Exception Handling:** `try/except/finally` blocks are used to catch and handle potential runtime errors, improving the robustness of the code.

**String Formatting:** The f-string syntax is used to insert variables into strings. This is a convenient way to create complex strings.

**Multithreading:** The `concurrent.futures.ThreadPoolExecutor` is used to create a pool of worker threads. The `threading.Lock` is used to ensure thread-safe access to shared resources.

**String Encoding/Decoding:** The `encode` and `decode` methods are used to convert strings to bytes and vice versa. These are built-in methods provided by Python's `str` and `bytes` classes.

## Essential Parts

**Socket Programming:** The creation of a socket, binding it to an address, listening for connections, accepting a connection, and sending and receiving data are all essential parts of a server and client.

Multithreading: The use of a thread pool to handle multiple clients concurrently is essential for a scalable server.

Exception Handling: Proper handling of exceptions is essential to prevent the server and client from crashing when errors occur.

Session Management: The server maintains a dictionary of sessions, which is essential for tracking client interactions.

Locks: The use of a lock to ensure thread-safe access to shared resources is essential in a multithreaded environment

## Citations

<https://realpython.com/python-sockets>

<https://realpython.com/intro-to-python-threading/#using-a-threadpoolexecutor/>

<https://docs.python.org/3/tutorial/errors.html>

<https://docs.python.org/3/library/threading.html#lock-objects>

<https://docs.python.org/3/howto/sockets.html>

<https://codefellows.github.io/sea-python-401d4/lectures/async.html>

<https://www.datacamp.com/tutorial/a-complete-guide-to-socket-programming-in-python>

# C++ Findings and Optimization

## Summary of the Server, Client, and Session Handler

### 1. Server:

The Server file is responsible for setting up the server, accepting incoming client connections, and managing these connections. It uses the POSIX socket API for networking, and it creates a new thread for each client to handle communication concurrently.

#### Key Features:

- **Socket Programming:** The server uses sockets for communication. It sets up a socket, binds it to a specific IP address and port, and listens for incoming connections.
- **Multithreading:** For each client that connects, the server creates a new thread to handle the client's requests. This allows the server to handle multiple clients concurrently.
- **Session Management:** The server uses the Session Handler to manage sessions for each client.

### 2. Client:

The Client file is responsible for setting up the client and communicating with the server. It also uses the POSIX socket API for networking.

#### Key Features:

- **Socket Programming:** The client sets up a socket and connects to the server using the server's IP address and port.
- **Communication:** The client sends requests to the server and receives responses. This communication is done using the send and read functions.

### 3. Session Handler:

The Session Handler file is responsible for managing sessions. A session starts when a client connects to the server and ends when the client disconnects. The Session Handler keeps track of all active sessions.

## Features

- **Session Management:** The Session Handler can create, end, and print all active sessions. It also generates a unique session ID for each session.
- **Thread-Safety:** The Session Handler uses a mutex to ensure that updates to the sessions are thread-safe. This is important because multiple threads (each handling a different client) may try to update the sessions concurrently.

## Libraries

1. `<iostream>`: Used for input/output operations.
2. `<string>`: Provides the `std::string` class.
3. `<vector>`: Provides the `std::vector` container.

4. `<thread>`: Used for working with threads.
5. `<mutex>`: Provides the `std::mutex` and `std::lock_guard` classes for synchronizing data access among threads.
6. `<chrono>`: Provides utilities to deal with time.
7. `<ctime>`: Provides functions to get and manipulate date and time information.
8. `<ma>`: Provides the `std::map` container.
9. `<netinet/in.h>`: Contains constants and structures needed for internet domain addresses.
10. `<unistd.h>`: Provides access to the POSIX operating system API.
11. `<cstring>`: Provides functions to manipulate C strings and arrays.
12. `<cerrno>`: Provides the macro `errno`, which is used to report error conditions.
13. `<atomic>`: Provides components for fine-grained atomic operations.

## Linker

The linker is a tool that combines multiple object files (produced by the compiler) into a single executable or library. It resolves references between object files, such as function calls and global variables, ensuring that all code dependencies are correctly mapped.

Once source files are compiled into object files, the linker takes these object files as input. It then performs symbol resolution (matching function calls with their definitions) and relocation (adjusting code and data references to fit the executable's memory layout). The output is an executable file or a library that can be run on a system.

Key Commands executed:

- `clang++ -c -Iinclude src/Server.cpp src/SessionHandler.cpp src/main_server.cpp` for compiling into object files.
- `clang++ Server.o SessionHandler.o main_server.o -o server_program` for linking object files into an executable.

Output: Executable files (`server_program`, `client_program`) that can be run to perform the tasks implemented in the source code.

## Static Analyzers

Static analyzers examine source code for potential errors, bugs, or vulnerabilities without executing the code. They use various analysis techniques to detect issues such as memory leaks, buffer overflows, and logical errors, helping developers to fix problems early in the development process.

The Clang static analyzer, for example, parses the source code, builds an abstract syntax tree (AST), and performs checks against a set of rules to identify potential issues. The analysis can be customized with different checkers to focus on specific types of problems.

Key Commands Used :

- `clang++ --analyze -Xanalyzer -analyzer-output=html -Iinclude src/Server.cpp src/SessionHandler.cpp` to analyze source files and generate reports.

Output: Reports in various formats (e.g., HTML, plist) detailing potential issues found in the code. These reports include information about the nature of the issues, their locations, and sometimes suggestions for fixes.

## Sanitizers

Sanitizers are dynamic testing tools that detect various types of errors such as memory corruption, undefined behavior, and data races during the execution of a program. They are instrumental in identifying issues that might not be caught by static analysis or are difficult to reproduce in a debugging environment.

When you compile and link your program with a sanitizer enabled (using specific compiler flags), the compiler inserts additional checks into the generated code. These checks are designed to detect and report runtime errors related to memory, behavior, or threading issues.

Key Commands:

- Compilation with sanitizers: `clang++ -fsanitize=address -include src/Server.cpp src/SessionHandler.cpp -o server_program` for AddressSanitizer.
- Execution of instrumented executables to detect runtime issues.

Output: Detailed error reports printed to the terminal or log files when the instrumented program encounters an error during its execution. These reports include information about the type of error, the location where it occurred, and sometimes a stack trace.

Incorporating linkers, static analyzers, and sanitizers into our project has significantly enhanced its quality. These tools helped us stitch together our code into working software, spot potential issues before they became problems, and catch errors that only show up when the software is running.

## Potential Improvements

1. Concurrency Management: Use a thread pool to handle clients and consider asynchronous I/O operations.
2. Logging: Record more detailed logs, use a logging library for better log management, and consider a centralized logging system for easier analysis.

## Citations

<https://www.geeksforgeeks.org/socket-programming-cc/>

<https://www.geeksforgeeks.org/socket-programming-in-cc-handling-multiple-clients-on-server-without-multi-threading/>

<https://rvarago.medium.com/introduction-to-cmake-for-cpp-4c464272a239>

Chat GPT - Bug fixing, Understanding and how to use (sanitizer, linker, analyzer), other key aspects of the socket programming

## Contribution

For our group project, Sagar and Utkarsh, who are roommates, worked together for the most part. We spent a lot of time working side-by-side project, using what each of us was good at to make the project better.

Utkarsh was good with Java. He focused a lot on learning all the necessary parts and used that knowledge to make sure those parts of our project worked well.

Sagar took charge of the Python socket programming part, having knowledge of networks. He used his knowledge and implemented a client-server connection to exchange data smoothly.

When it came to C++, we both worked together. We knew that combining our knowledge would help us deal with the tricky bits of C++. Working together on C++ helped us come up with better ideas and solutions.

For the project's presentation, Sagar made the poster, making sure it showed what our project was about and what we achieved. Utkarsh wrote the project report, carefully explaining everything we did, what we found out, and our final thoughts. We both checked to make sure that the parts of our project using Java, Python, and C++ worked well together.

Working together this way helped us learn a lot about different programming languages and made our working relationship stronger. It all helped make our project successful.