# **DISTRIBUTED SYSTEMS (CSE 5306)**

# **PROJECT 1 REPORT**

**Team Members:**

## Asfiya Misba (1002028239)

## Divya Darshi (1002090905)

I have neither given nor received unauthorized assistance on this work.

Signed: AM, DD Date: Oct 4, 2022

# **ASSIGNMENT 1**

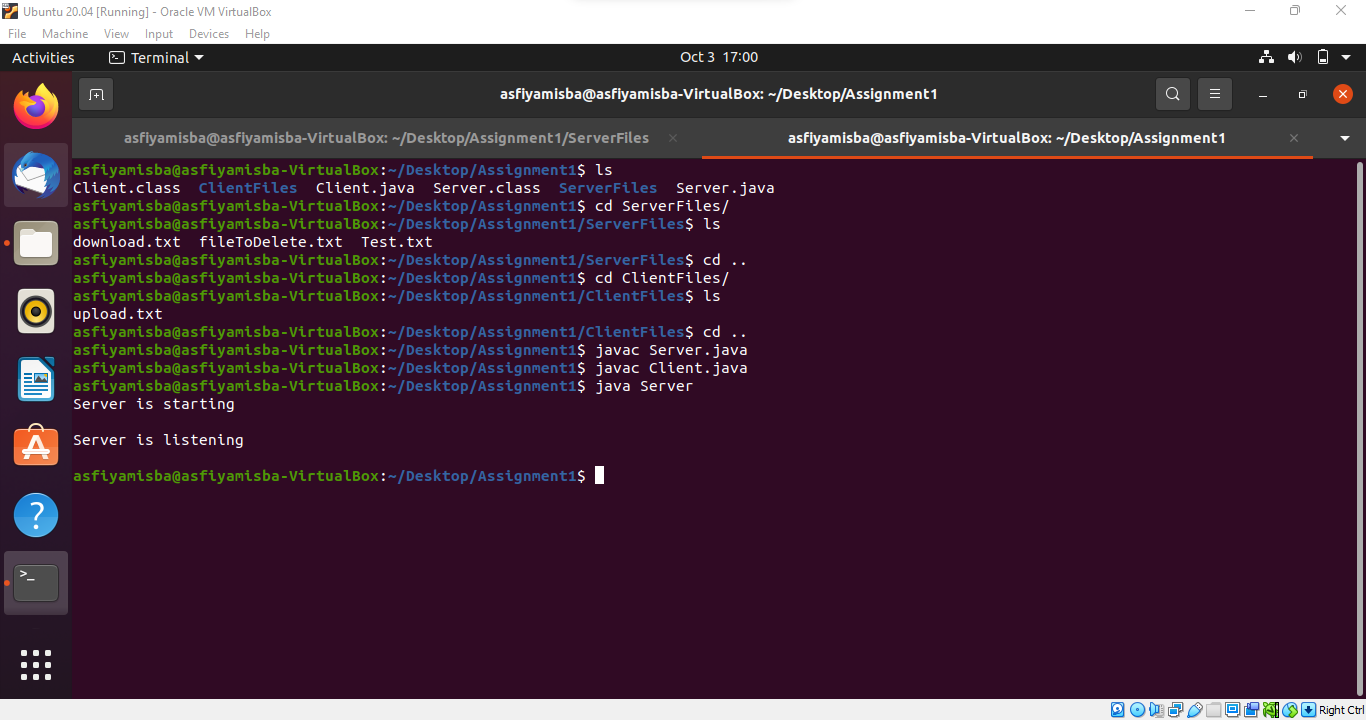
* Single-threaded file server to UPLOAD, DOWNLOAD, DELETE and RENAME is implemented.
* We have used different folders to hold the client and server files.
* A single-threaded server means that we initiate only one connection, whereas in a multi-threaded server we have multiple connections at the same time.
* In this assignment, we understood the working of the single-threaded client-server architecture.

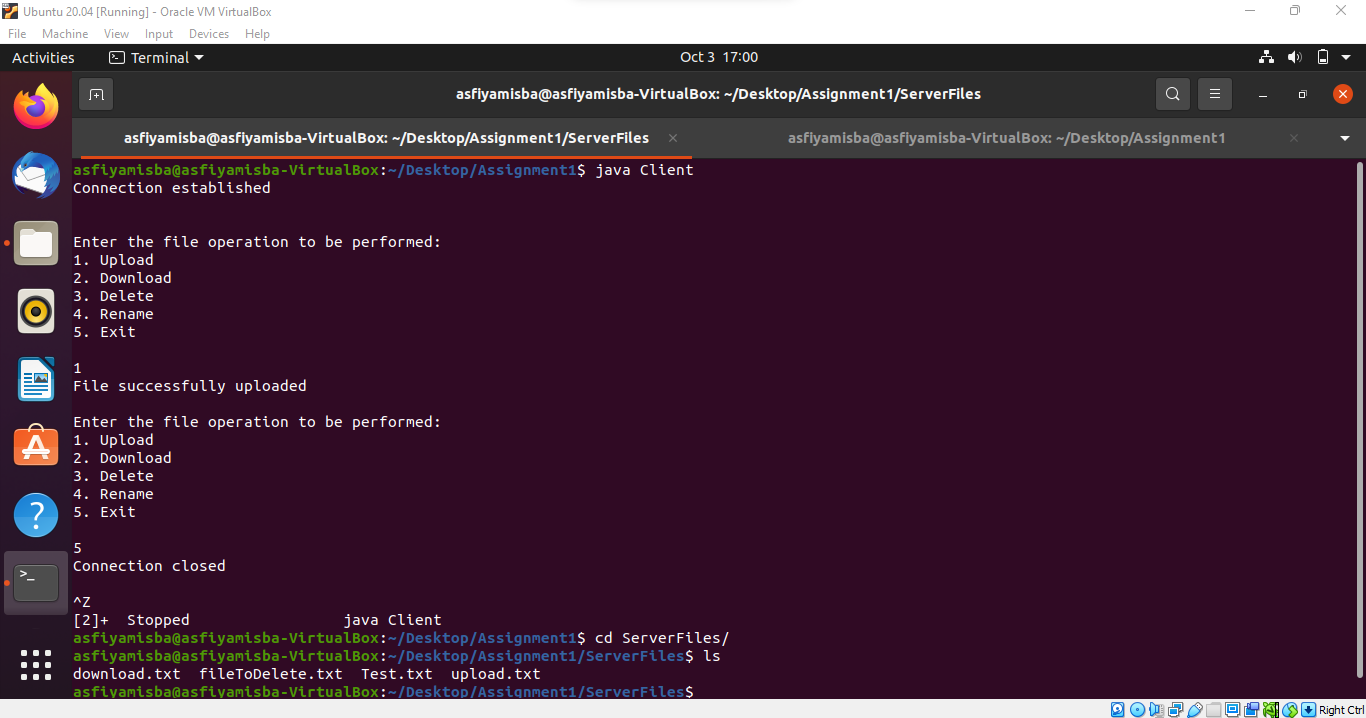
## ISSUES ENCOUNTERED

* We spent a considerable amount of time figuring out which file path to pass in the client file and which one to pass in the server file.
* While performing the rename operation, a renamed file was generated along with the original file. We then deleted the original file to make sure only one copy existed.
* If we would like to perform file operations for larger files, the byte array size must be increased.

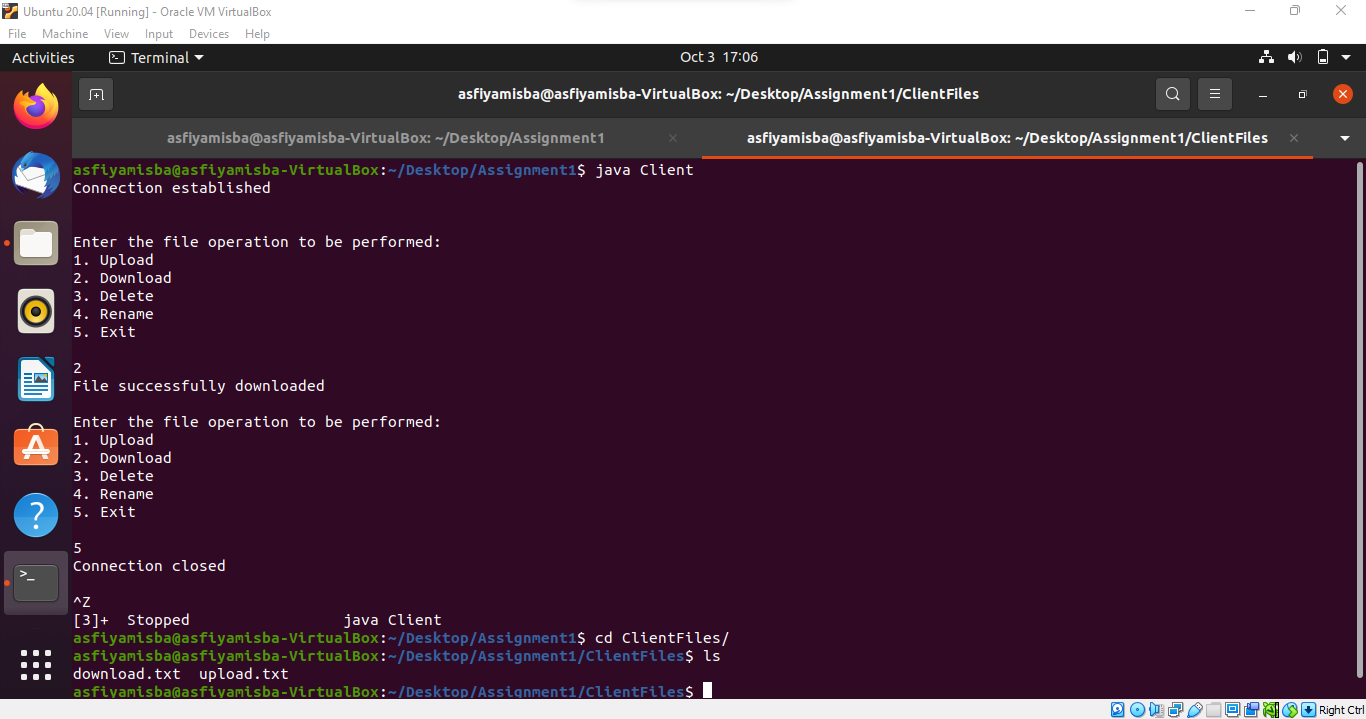
## OUTPUT SCREENSHOTS

1. **UPLOAD**

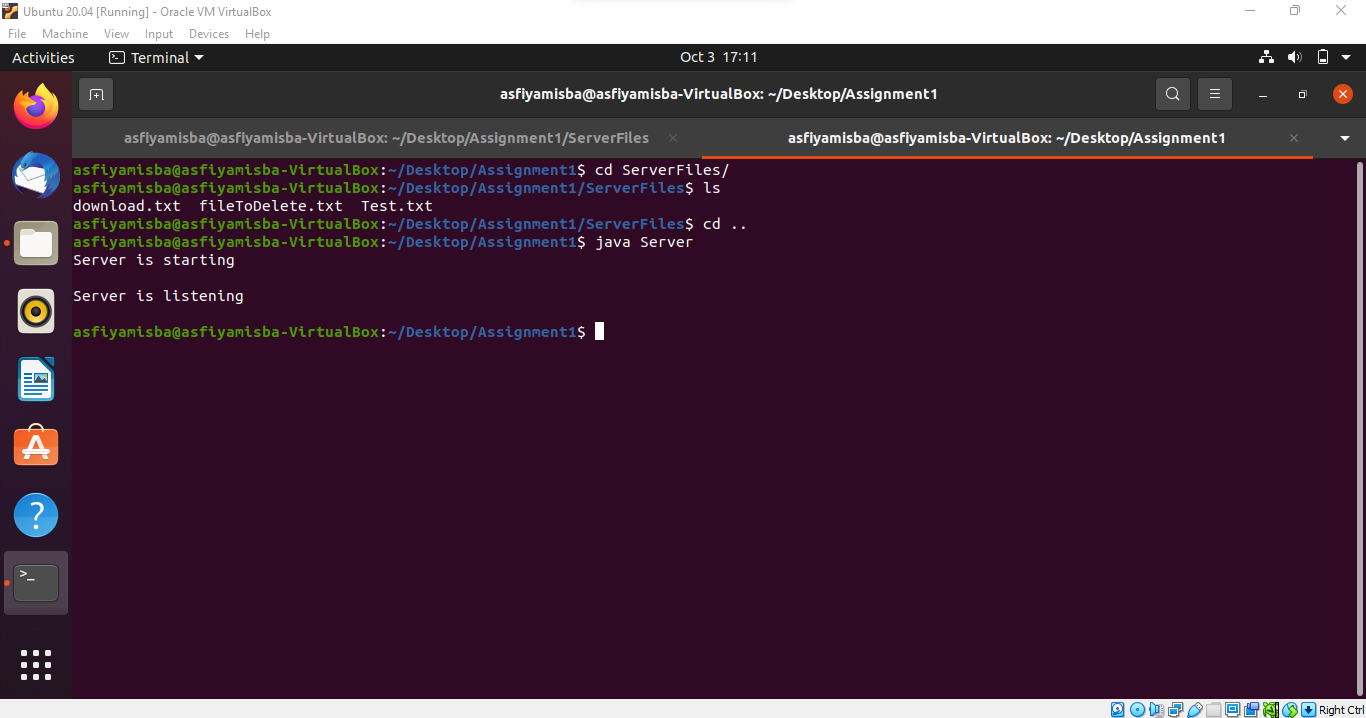


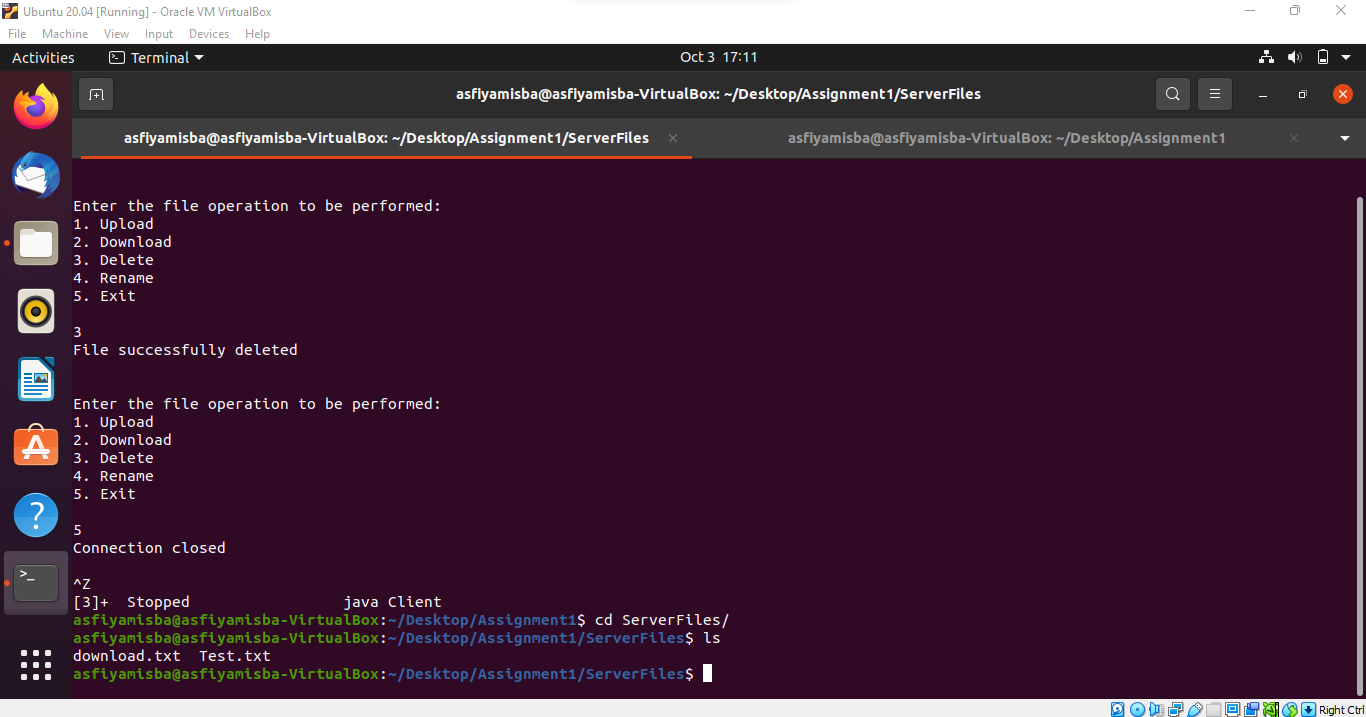


1. **DOWNLOAD**

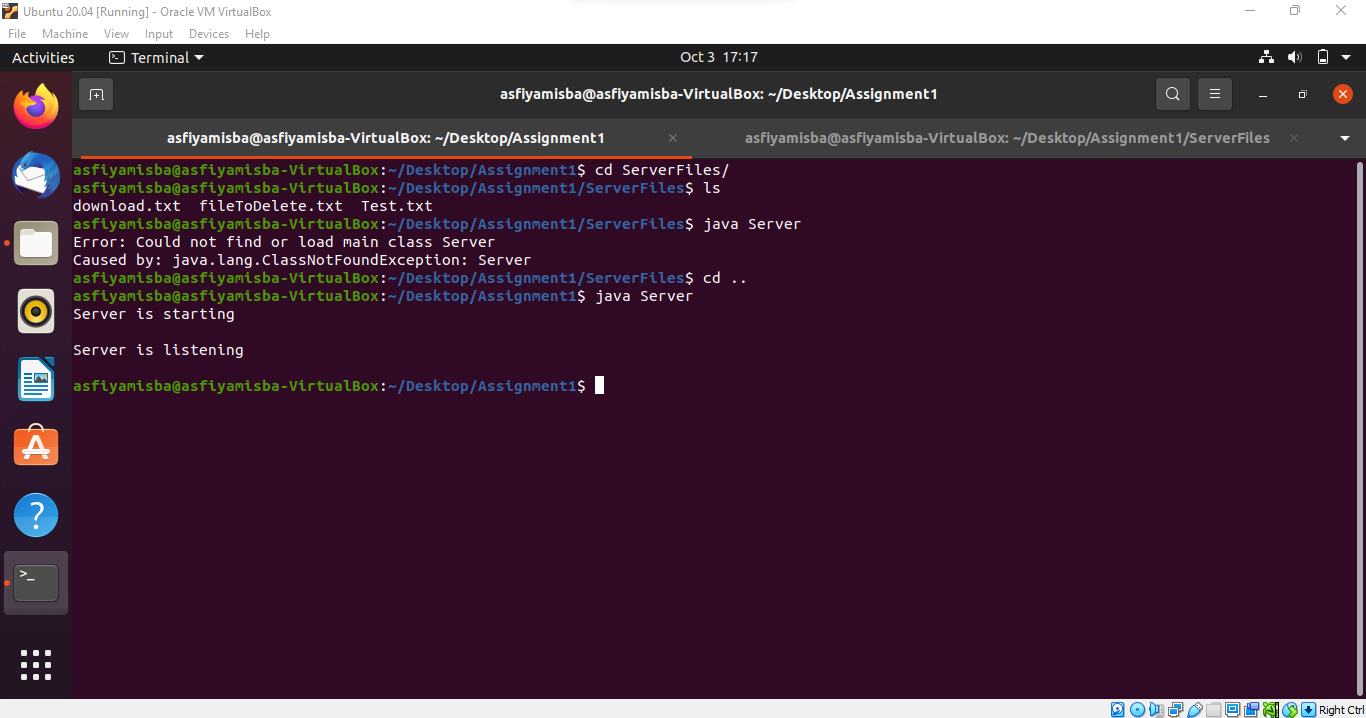
****

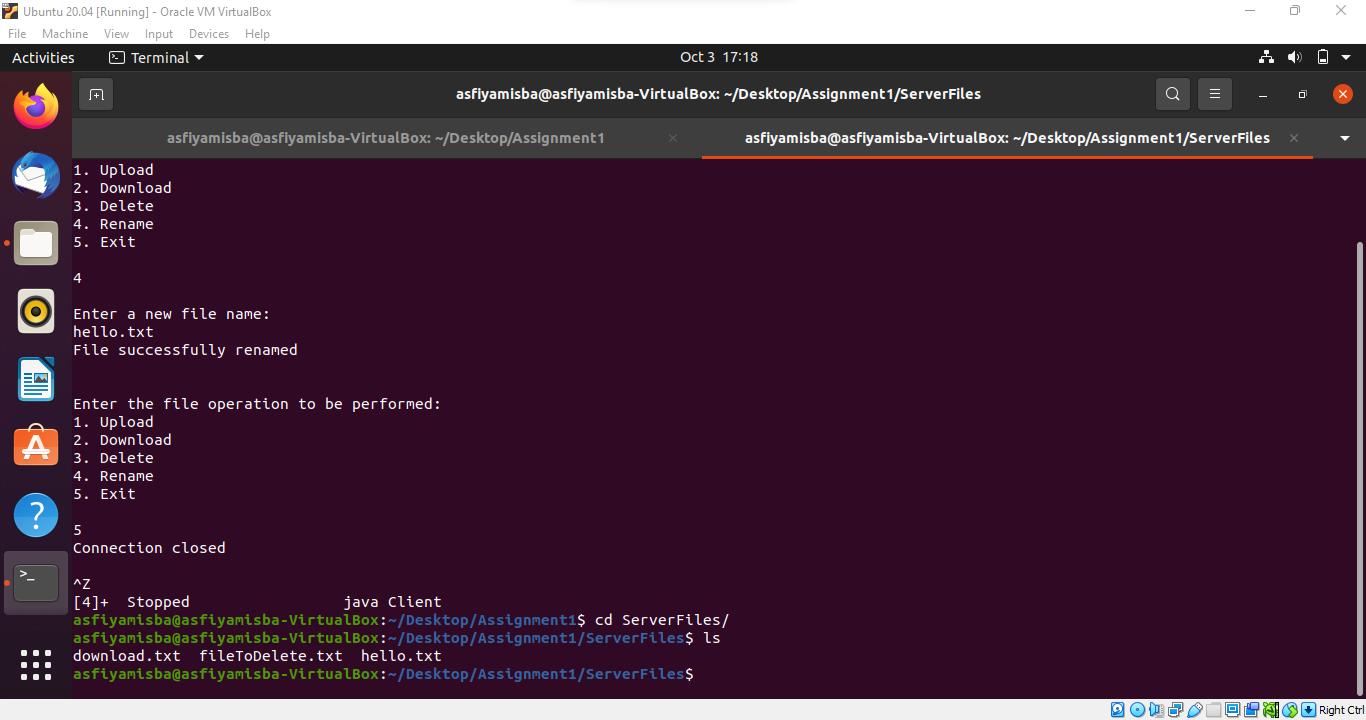
1. **DELETE**

****

****

1. **RENAME**

****

****

# **ASSIGNMENT 2**

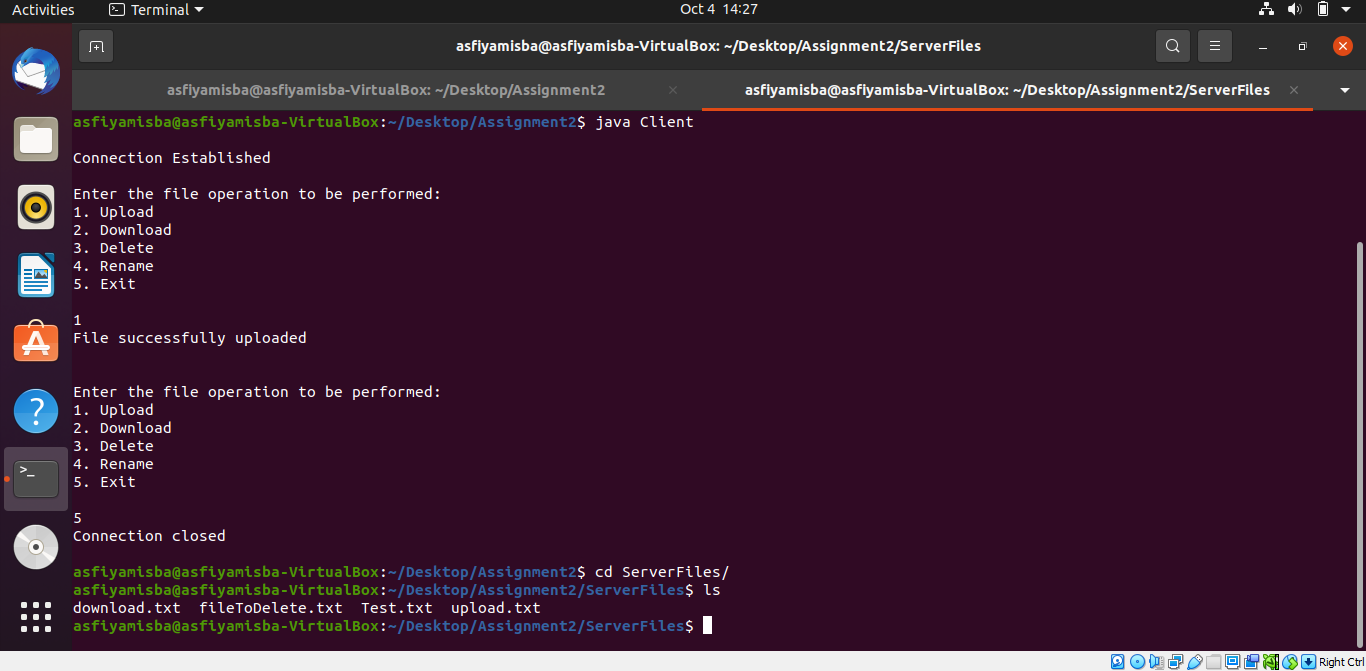
* Multi-threaded file server to UPLOAD, DOWNLOAD, DELETE and RENAME is implemented.
* A server with more than one thread is known as a multi-threaded server.
* Threads are generated whenever the client sends a request to communicate with the server.
* Multiple threads are generated to accept multiple requests from multiple clients at the same time.

## ISSUES ENCOUNTERED

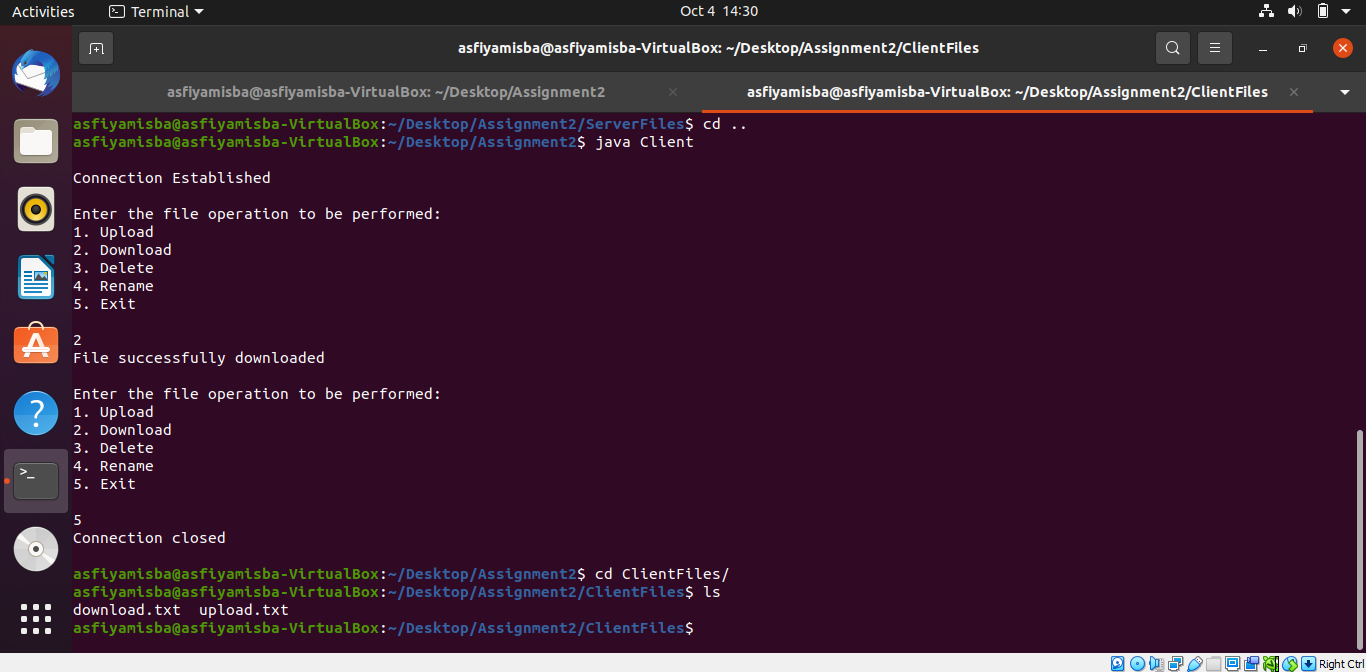
* While performing, the client repeatedly failed to connect to the server, and it was very difficult to upload and download files to the server.

## OUTPUT SCREENSHOTS

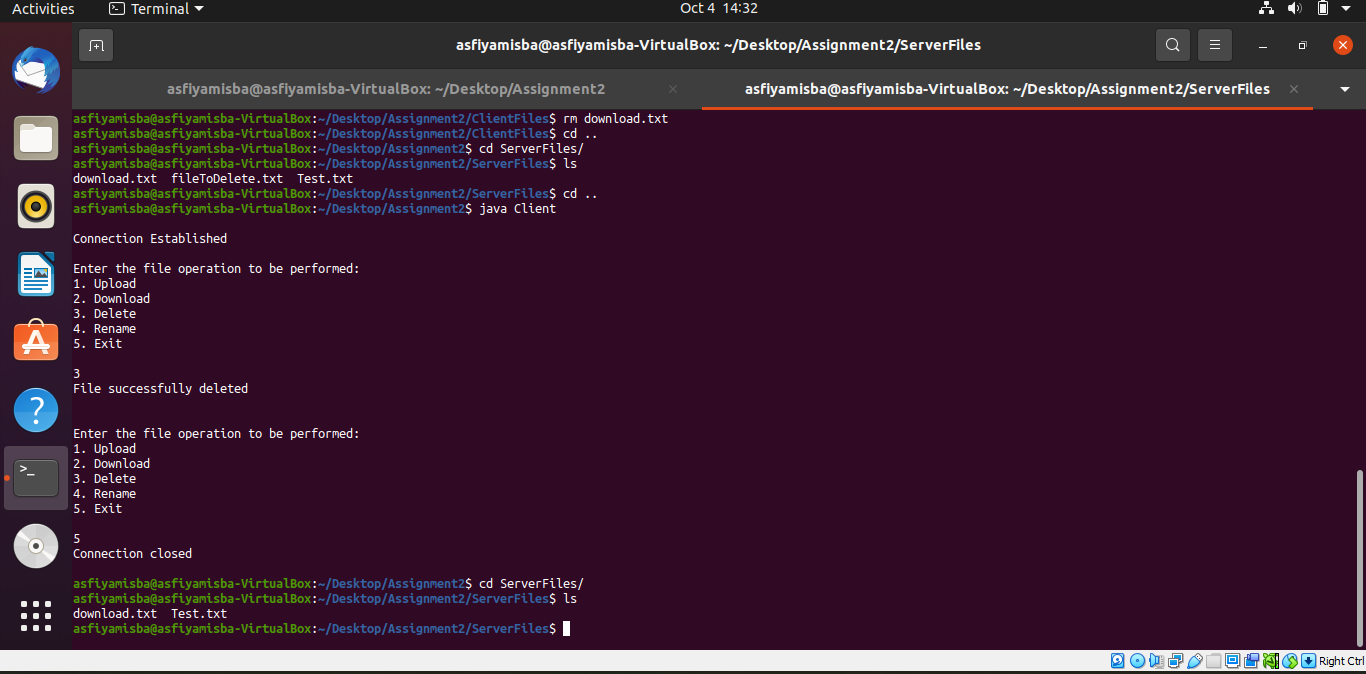
1. **UPLOAD**

****

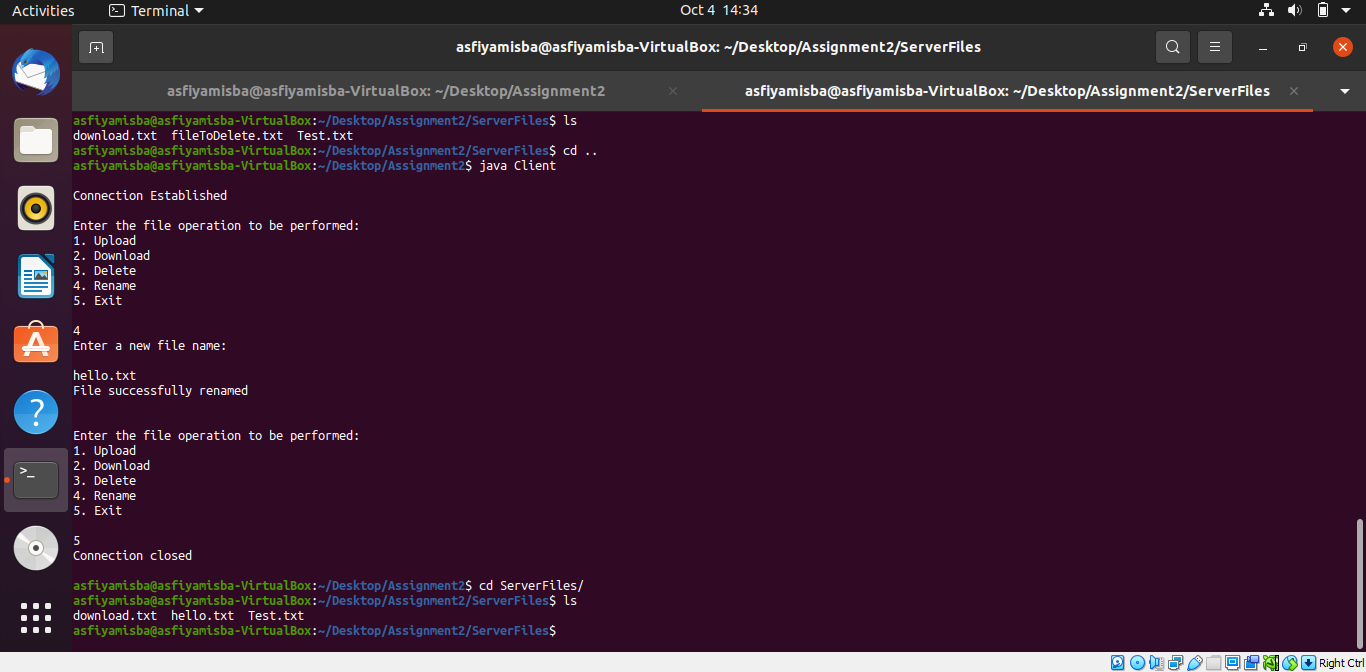
1. **DOWNLOAD**

****

1. **DELETE**

****

1. **RENAME**

****

# **ASSIGNMENT 3**

* Implement a computation server of Synchronous RPCs(calculate\_pi(), add(i, j), sort(array A), matrix\_multiply(matrix A, matrix B, matrix C)
* RPC is a powerful technique for creating distributed client/server-based applications.
* RPC is analogous to a function call. Like a function call, RPC passes the caller's

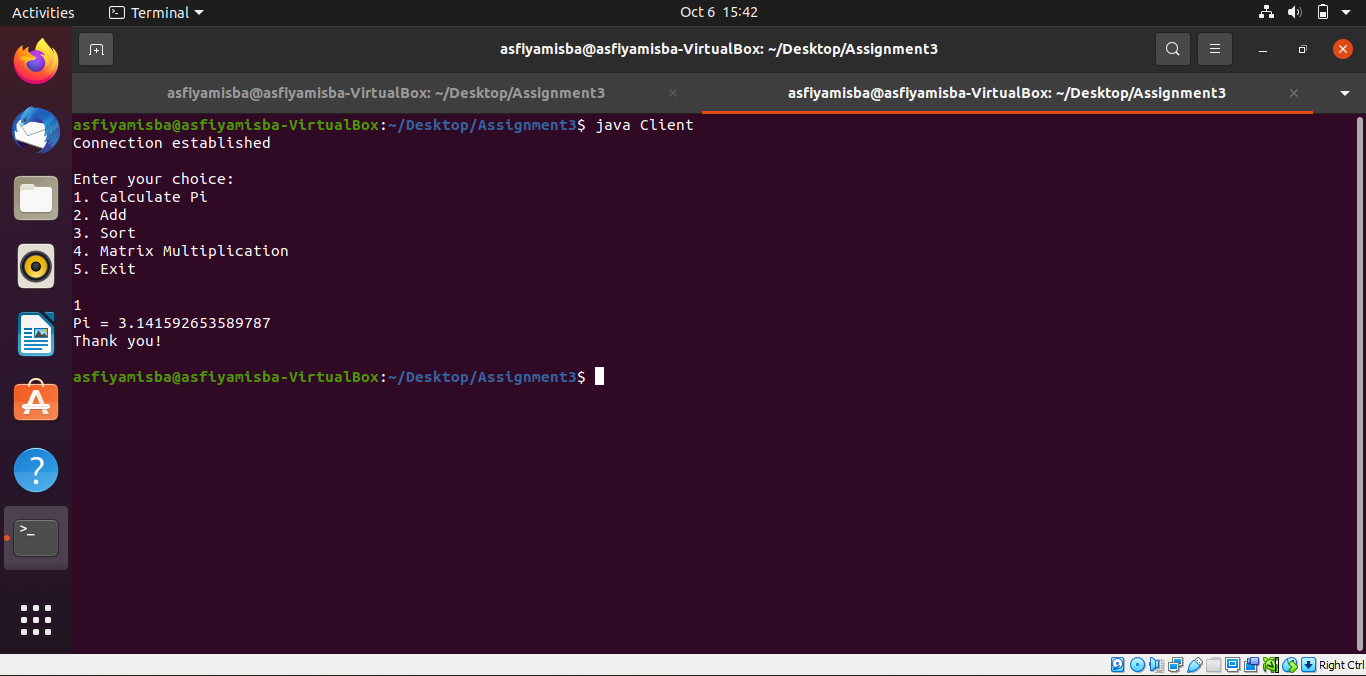
arguments to the remote procedure, and the caller waits for a response from the remote procedure.

## ISSUES ENCOUNTERED

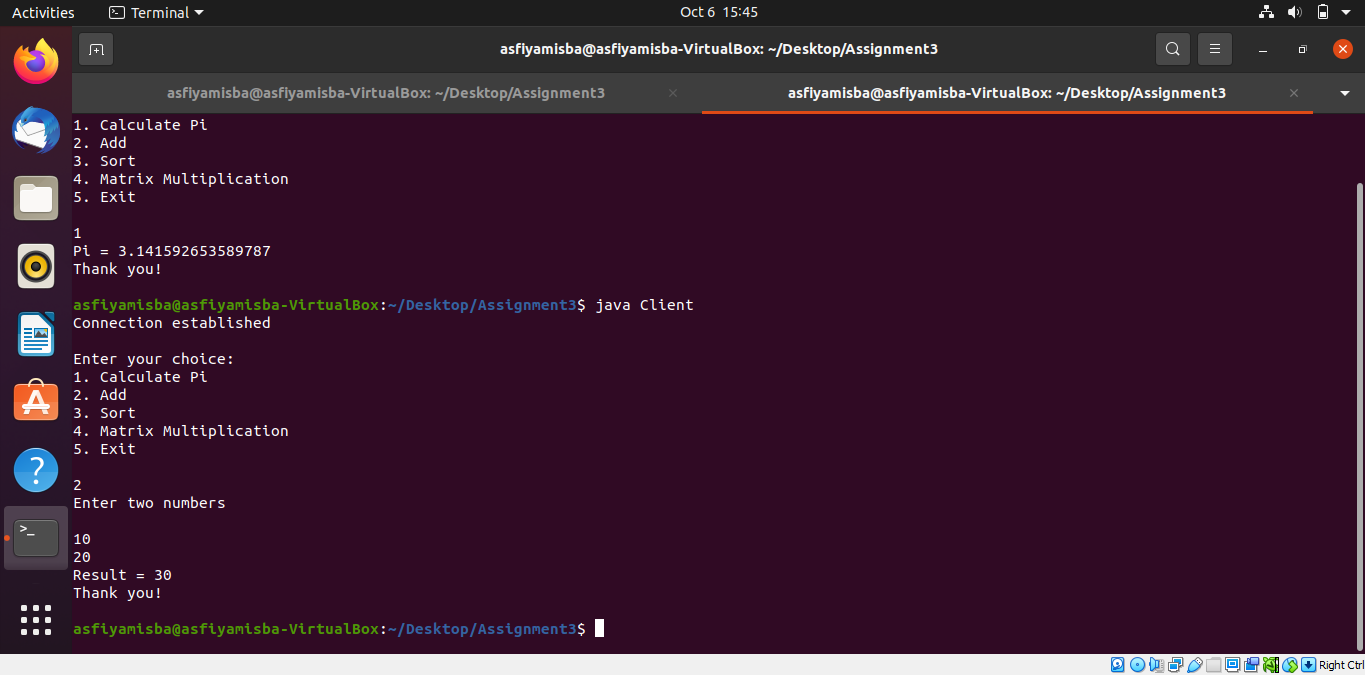
* While performing the operations, the client repeatedly failed to connect to the server.
* Some changes were made to successfully connect, and the problem was later resolved.
* We faced problems sending the two matrices from the client to the server side to calculate the results while receiving the product matrix from the server.
* Initially, the product matrix was printing zero values, we later figured out that the client was unable to read the product matrix from the server and that fixed it.

## OUTPUT SCREENSHOTS

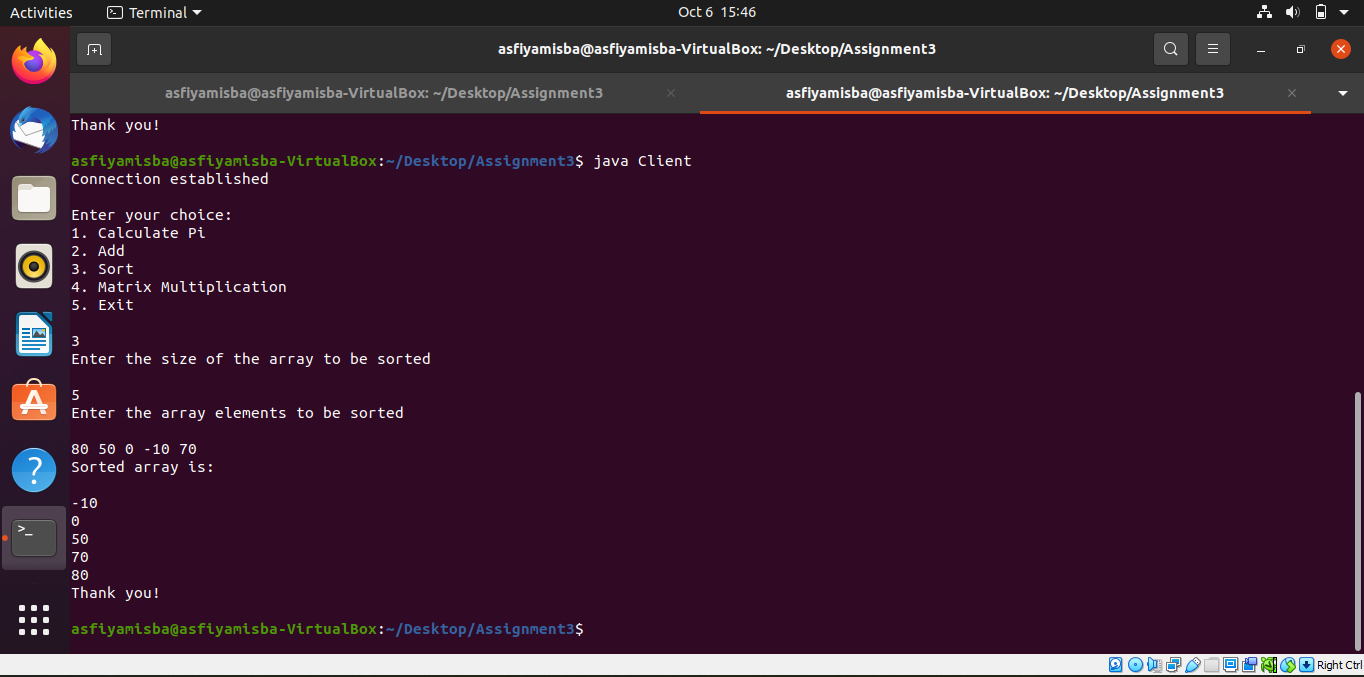
1. **CALCULATION OF PI**

****

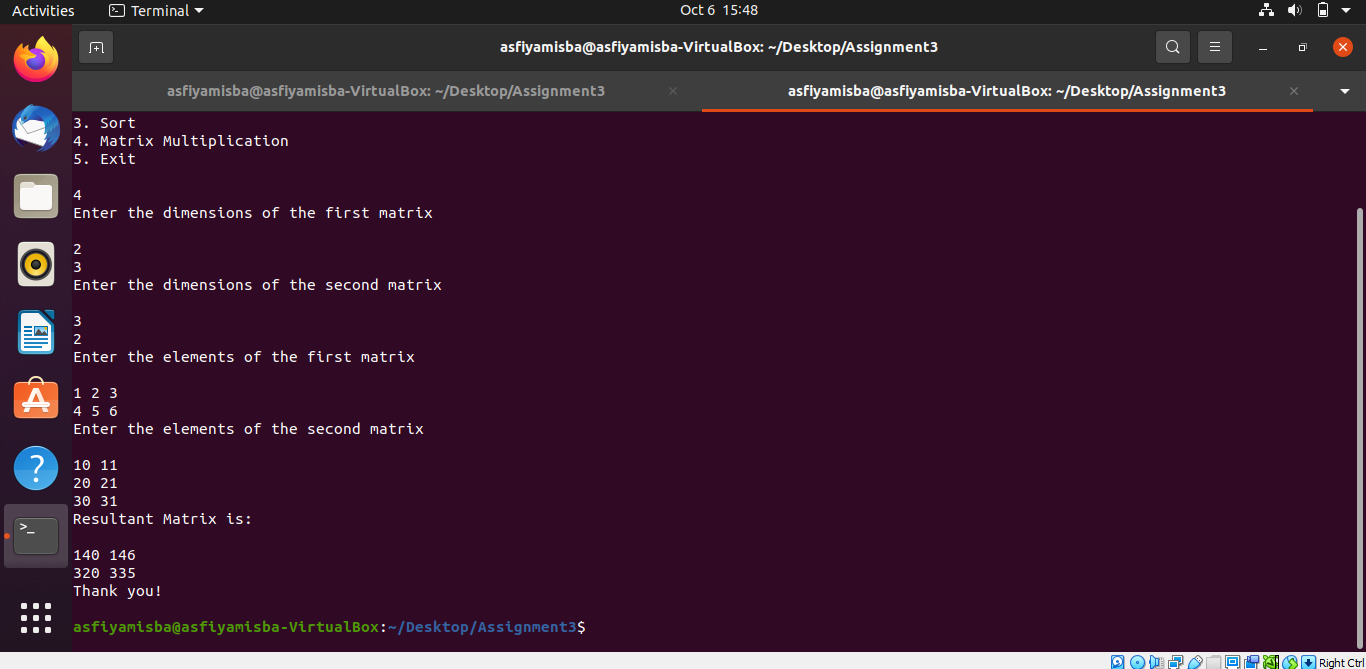
1. **ADDITION OF TWO NUMBERS**

****

1. **SORTING AN ARRAY**

****

1. **MATRIX MULTIPLICATION**

****

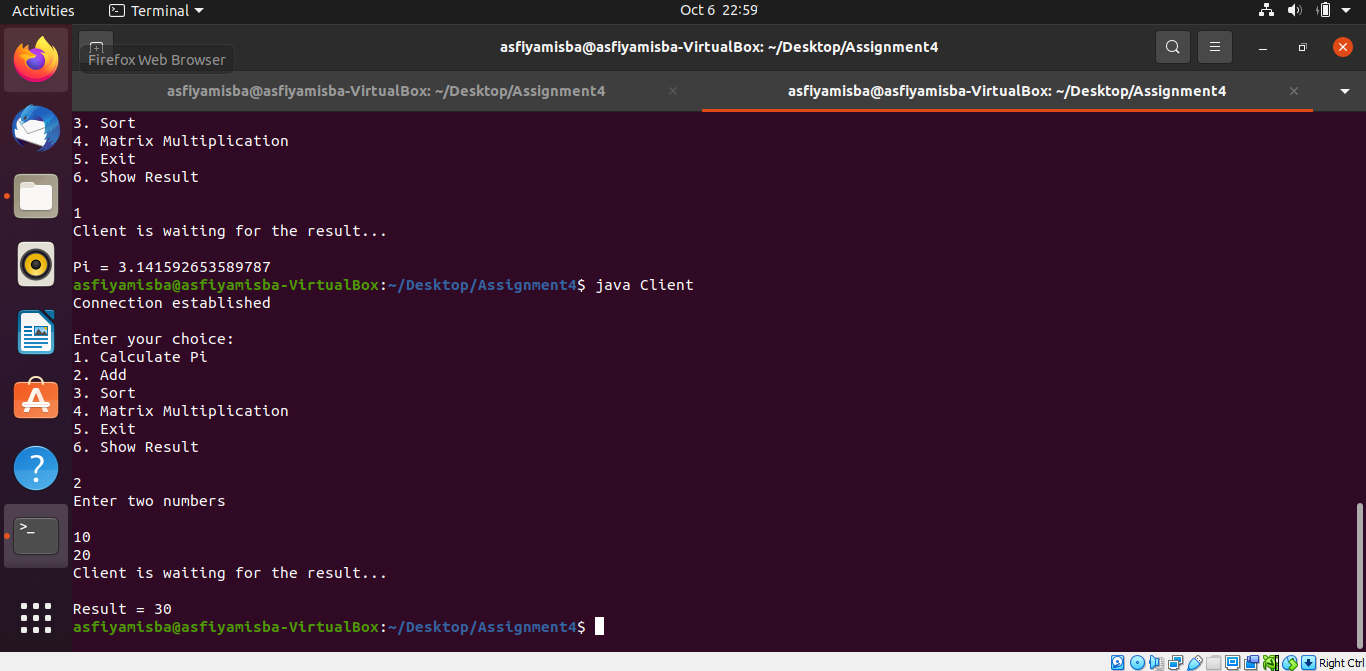
# **ASSIGNMENT 4**

* Implement the computation server using asynchronous and deferred synchronous RPCs.
* The server supports four RPCs: calculate\_pi(), add(i, j), sort(array A), matrix\_multiply(matrix A, matrix B, matrix C.
* The most important thing about asynchronous distributed systems is that they make no hard assumptions about the timing and order of events in distributed systems, so they are better suited to real-world scenarios.

## ISSUES ENCOUNTERED

* While performing the operations, the result returned in an unpredictable order.
* We spent a considerable amount figuring out how to send request to the Server.
* Lack of connection.

OUTPUT SCREENSHOTS



**LEARNING OUTCOMES**

* The project taught us how to solve Single and multithreaded client server problems.
* A socket is an endpoint in inter-network communication, and socket programming allows these endpoints to transfer data, thus supporting inter-network and program-to-program communication.
* And, how to work with message-oriented and RPC-based communication and how to apply it to real work projects.
* RPC hides all networking code in stub functions, allowing programmers to use procedure call semantics and simplifying writing distributed applications.

**WHY USE THREADS IN NETWORK PROGRAMMING?**

The reason is simple. Because we want not only one client to connect to the server at a time, but many clients at the same time. The architecture must support multiple clients simultaneously. For this reason, you should use threads on the server side so that each client request can allocate a separate thread to handle each request.

**The Challenges in converting the single-threaded server to a multi-threaded version.**

* Converting code from a single-threaded server to a multi-threaded server will allow the program to consume less memory by accepting many requests on a multi-threaded program.
* But it will waste the client's time if too many requests are made.
* Code length will also be increased significantly when the code is modified to allow multiple threads.
* And increased potential for deadlock occurrence.
* There is a balance between the two options and the option that suits the application requirements should be used.

**Why is Distributed locking necessary for multithreaded servers?**

* A multi-threaded server requires distributed locking because it is not just about multiple processes, it's about multiple clients running on different machines.
* These clients, running on separate machines, should not interfere with each other when accessing the resources.
* We want only one client to access a resource. If resources are shared, two clients interacting with the server at the same time can change the system state and produce erroneous results.
* Distributed locks provide mutually exclusive access to shared resources in a distributed environment. Distributed locks are used to improve the efficiency of services or implement the absolute mutual exclusion of accesses.

# 

# **REFERENCES**

[1] <https://www.oracle.com/java/technologies/jpl2-socket-communication.html>

[2]<https://www.techtarget.com/searchapparchitecture/definition/Remote-Procedure-Call-RPC?amp=1>

[3] <https://docs.sockettools.com/v10/activex/guide/control/asyncio.html>