

Northeastern University
CS 6650 Scalable Dist Systems
Homework Set #1 [100 points]
Name: Zheng Yin
email: yin.zheng@northeastern.edu

GOAL: An introductory understanding of Dist Sys and Java client server programming.

INSTRUCTIONS: Please provide clear explanations *I your own sentences, directly answering the question, demonstrating your understanding of the question and its solution, in depth, with sufficient detail. Submit your solutions [PDF preferred]. Include your full name. Do not email the solutions.*

I. Study **Chapter 2 Systems Models** Coulouris Book [10 points]

Answer the following questions using explanation and diagrams as needed:

2.11

a) In a distributed system it is hard to set limits on the time that can be taken for process execution, message delivery or clock drift. It is possible to suggest likely upper and lower bounds for process execution time, message delay and clock drift rates in a distributed system, but it is difficult to arrive at realistic values and to provide guarantees of the chosen values.

b) By limiting the number of clients to the capacity of the server, you can ensure that the system can handle all requests within a bounded time.

Another way to deal with more clients is to use a server with more processors, which will increase the system's processing power and allow it to handle more requests simultaneously.

Another option is to replicate the service, this means creating multiple instances of the service and distributing the clients among them. This can be done using load balancers, which distribute incoming requests among the available service instances. This approach can increase the system's availability and scalability, as it allows the system to handle more requests and clients, and also provides fault-tolerance, as if one instance of the service goes down, the others can still handle the requests.

c) replicating the service is not always the best solution, it depends on the nature of the service, the cost of replication, and the network latency.

2.14

Service A may experience arbitrary failures due to the lack of checksums for message bodies, which can lead to corrupted or duplicated messages. Additionally, it may also experience omission failures where messages are lost during communication.

Service B, on the other hand, may only experience omission failures such as lost or dropped messages. However, it will not experience timing failures as it operates within an asynchronous distributed system. Despite passing integrity tests, it cannot be considered reliable as it does not pass validity tests.

II. Please refer to the 2 articles (PDF) on Middleware for Distributed Systems. [10 points]
See Fig 1. layers of Middleware

What is middleware for Dist Systmes/ What are its uses (list and explain).

Consider a multiplayer game as a Dist Sys. It is played on 4 players PCs and a Game Server.

What are the heterogeneous elements among these 5 systems at each layer? Give examples to ilustarte this (e. g. MacOS on PC1; Linux on PC 2 etc.) and then explain how middleware helps here.

Definition of Middlewares

Middleware is a class of software technologies designed to help manage the complexity and heterogeneity inherent in distributed systems. It is defined as a layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system.

The uses of middlewares

Distributed Tuples: Distributed tuples, also known as distributed data structures, are data structures that are spread across multiple nodes in a distributed system. The goal of distributed tuples is to provide a way for systems to share and process data in a distributed environment.

Remote Procedure Call : Remote Procedure Call (RPC) is a type of middleware that allows for transparent communication between systems, enabling them to call procedures or functions on remote systems as if they were local.

Message-Oriented Middleware : Message-oriented middleware (MOM) is a type of middleware that provides messaging infrastructure to support the asynchronous exchange of messages between systems in a distributed environment. It allows systems to send and receive messages asynchronously, improving the scalability and reliability of the distributed system.

Distributed Object Middleware : Distributed Object Middleware (DOM) is a type of middleware that allows for the implementation of distributed objects, which are objects that can be accessed and manipulated by multiple systems in a distributed environment. DOM provides the infrastructure for remote method invocation (RMI) and remote object lookup (ROL), which allow systems to access and manipulate objects on remote systems as if they were local.

Give examples to ilustarte this (e. g. MacOS on PC1; Linux on PC 2 etc.) and then explain how middleware helps here.

In a multiplayer game as a distributed system, there are several heterogeneous elements among the 5 systems (4 player PCs and a game server) at each layer.

At the hardware layer, the systems may have different architectures, such as PC1 having an AMD 64-bit architecture while PC2 has an ARM 64-bit architecture. This can cause compatibility issues when trying to run the same game on different systems.

At the operating system layer, the systems may have different operating systems, such as PC1 running Windows while PC2 runs Linux. This can also cause compatibility issues when trying to run the same game on different systems.

At the application layer, the systems may have different versions of the game, or different levels of performance, such as PC1 having a high-end graphics card while PC2 has a lower-end graphics card. Middleware can help to address these heterogeneous elements by providing a common interface and set of services that can be used to communicate and interact with different systems.

Middleware can provide a communication protocol that can be used to send and receive messages between systems, regardless of their architecture or operating system.

Middleware can provide a resource sharing mechanism that allows for the sharing of resources such as databases, memory, and processing power among different systems.

Middleware can provide a security mechanism that can be used to authenticate and encrypt communication between systems, ensuring that only authorized systems can access the game server.

Middleware can provide a load balancing mechanism that can be used to distribute the workload across multiple systems, ensuring that the game runs smoothly on all systems.

Middleware can provide a scalability mechanism that allows for the easy scaling of the distributed system by adding or removing resources as needed.

Overall, middleware can help to address the heterogeneous elements among the systems in a multiplayer game as a distributed system by providing a set of services that make it easier to develop, deploy, and manage the game across different systems.

III. From Chapter 3 Coulouris Book **Networking**

[20 points]

Answer the following questions using explanation and diagrams as needed:

3.1

The send and receive latencies include (operating system) software overheads as well as network delays. Assuming that the former dominate, then the estimates are as below. If network overheads dominate, then the times may be reduced because the multiple response packets can be transmitted and received right after each other.

i) UDP: $5 + 2000/10000 + 2 + 5(5 + 10000/10000) = 37.2$ milliseconds

ii) TCP: $5 + 5 + 2000/10000 + 2 + 5(5 + 10000/10000) = 42.2$ milliseconds

iii) same machine: the messages can be sent by a single in memory copy; estimate interprocess data transfer rate at 40 megabits/second. Latency/message ~ 5 milliseconds. Time for server call: $5 + 2000/40000 + 5 + 50000/40000 = 11.3$ milliseconds

3.7 part (ii) ONLY (i.e., FTP)

File transfer (such as FTP) can use either a connection-oriented or connectionless protocol, depending on the specific implementation.

FTP over TCP (FTP-Control) uses a connection-oriented protocol for control commands and negotiations, such as establishing a connection, logging in, and navigating the file system. This ensures that data sent between the client and server is delivered in the proper order and retransmitted if necessary.

FTP over UDP (FTP-Data) uses a connectionless protocol for transferring the actual file data. This allows for faster transfer speeds as there is no overhead for establishing and maintaining a connection. However, since UDP is connectionless, it does not guarantee the delivery of packets, or guarantee the order of packets, thus it may require error checking and retransmission mechanisms at application level.

In summary, using FTP over TCP provide a reliable data transfer, but it may be slower due to the overhead of maintaining a connection, while using FTP over UDP provides faster data transfer but with less reliability.

IV. Study **Chapter 13 Java Socket Programming** from this book

Object-Oriented Programming with Java: Essentials and Applications Authors Buyya, Selvi and Chu [2009] Tata McGraw Hill [PDF posted in Lecture 1 Folder]

13.22 In your own words, explain what the Tannenbaum text book's Layer cake cut diagram is about (pg 78 Figure 2.16: Client-server organizations in a two-tiered architecture), providing 2 examples each for the 5 architectural models shown (e. g. a Web app, A client-server app, Youtube etc.) [10 points]

ANS

Multitiered architectures involve physically distributing a client-server application across several machines. This can be done in a variety of ways, such as having two types of machines: a client machine with the user-interface level programs, and a server machine with the processing and data level programs. Another approach is to distribute the layers of the application across different machines, leading to a two-tiered architecture. This can include placing the entire user-interface software on the client side, moving part of the application to the front end, or having most of the application running on the client machine but with operations on files or database entries going to the server. Two popular organizations in client-server environments are a PC or workstation connected to a distributed file system or database, and the client's local disk containing part of the data.

- a) Some public API, Chatgpt
- b) Google Doc, Colab
- c) Job application website like amazonuniversity.jobs, grammarly
- d) Bofa(Bank of America APP), Zelle
- e) YouTube, Microsoft Office

13.23 What is a port? List some well-known ports and explain the applications associated with them. [5 points]

ANS

In the context of a network, a port is a numbered endpoint of communication on a host. When data is sent to a host on a network, it is sent to a specific port on that host, allowing the host to differentiate the data and direct it to the appropriate application or process.

In an operating system, a port is a software construct that acts as a channel for input and output (I/O) operations. It is a mechanism that allows different processes to communicate with one another. When a process wants to send data to another process, it sends it to the other process's port. The operating system's kernel then routes the data to the appropriate process.

Examples

Port 3306: MySQL - used for connecting to a MySQL database.

Port 80: HTTP (Hypertext Transfer Protocol) - used for transmitting data over the web.

Port 443: HTTPS (HTTP Secure) - used for transmitting secure data over the web.

Port 22: SSH (Secure Shell) - used for remotely accessing and managing servers.

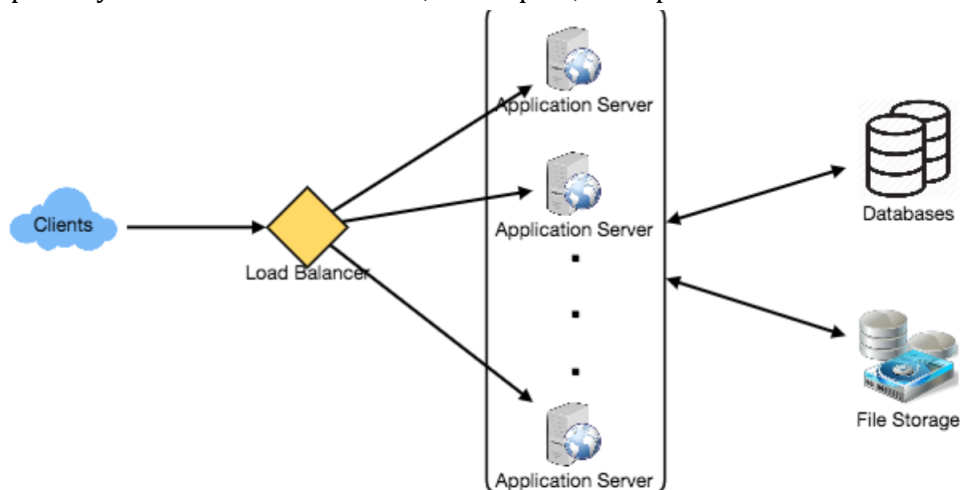
- V. Study Dist Sys Design Goals.pdf. Consider the architecture of Twitter. What do the Goals and Transparencies described in this paper mean in the context of Twitter? Why are they important? Explain with a diagram. [10 points]

ANS

In the context of Twitter's architecture, transparency, scalability, dependability, performance, and flexibility are key design principles that are aimed to be achieved through various design choices and techniques.

Transparency refers to the degree to which the inner workings of the architecture are visible to users, developers, and operators. Twitter aims to be transparent in its operations and make it easy for developers to understand and work with the platform. This is achieved by providing clear and detailed documentation, as well as open-sourced libraries and tools. Additionally, Twitter provides transparency in its service by providing a detailed API documentation, so that developers can easily access and interact with the data on the platform.

access transparency, location transparency, migration transparency, replication transparency, failure transparency, and concurrency transparency are key design principles that aim to provide transparency and ease of use for users, developers, and operators.



Access Transparency refers to the ability of the users to access the service regardless of the location or physical structure of the system. Twitter's architecture uses a combination of load balancers, reverse proxies, and Content Delivery Networks (CDN) to ensure that users can access the service from anywhere with minimal latency.

Location Transparency refers to the ability of the system to hide the physical location of data and services from the users. Twitter's architecture uses data sharding and replication to distribute data and services across multiple servers and nodes. This allows the system to hide the physical location of data and services and provide users with a consistent experience regardless of their location.

Migration Transparency refers to the ability of the system to move data and services between servers and nodes without disrupting the users. Twitter's architecture uses live data migration and data replication techniques to ensure that data and services can be moved seamlessly without disrupting the users.

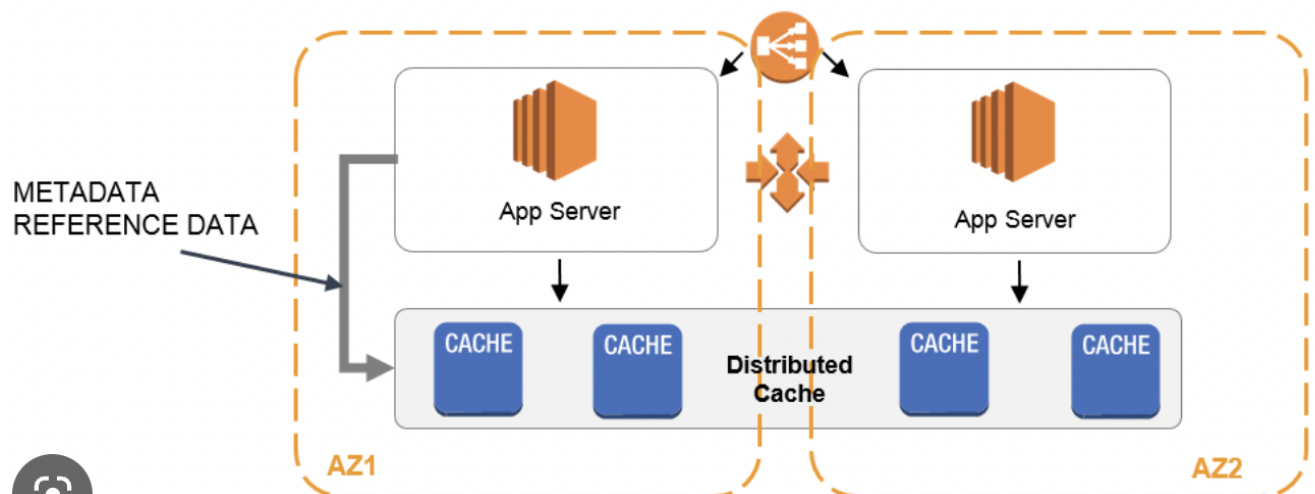
Replication Transparency refers to the ability of the system to replicate data and services across multiple servers and nodes without disrupting the users. Twitter's architecture uses data replication and sharding to ensure that data is available and accessible even if a server or node fails. **Failure Transparency** refers to the ability of the system to hide failures and errors from the users. Twitter's architecture uses redundancy and failover mechanisms to ensure that the service remains available even in the event of failures or errors.

Concurrency Transparency refers to the ability of the system to handle multiple concurrent users and requests without disrupting the users. Twitter's architecture uses load balancing and caching techniques to ensure that the service can handle a large number of concurrent users and requests without impacting performance.

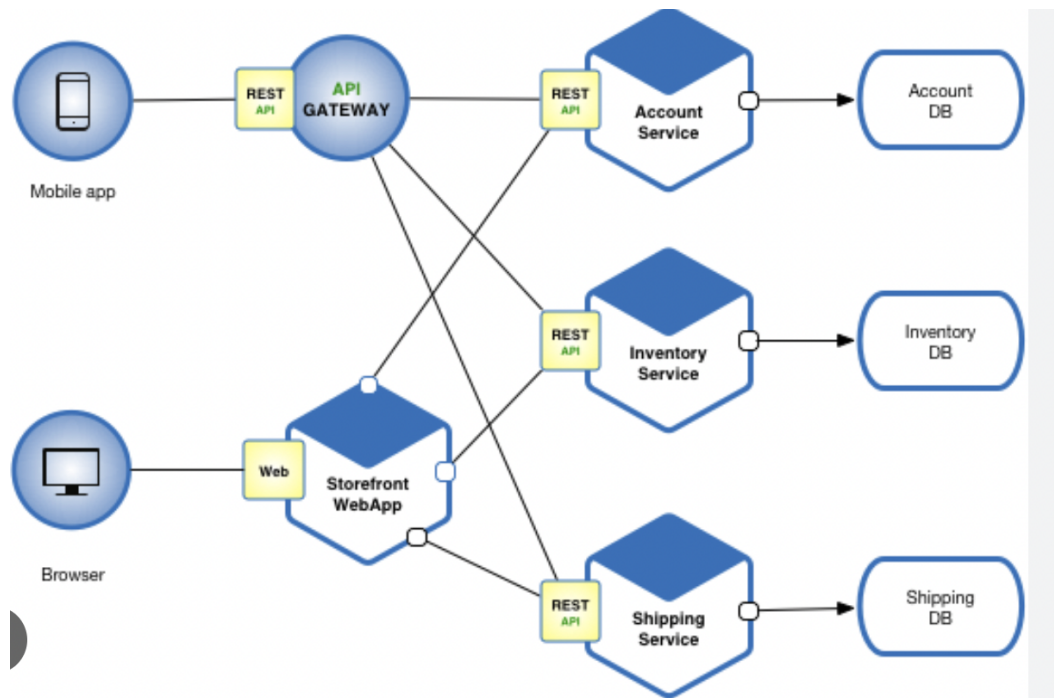
Scalability refers to the ability of the architecture to handle an increasing amount of load and user traffic. Twitter's architecture is designed to be highly scalable, with the use of distributed systems, data sharding, and load balancing techniques. This allows the platform to handle a large number of concurrent users and requests, while still providing a low-latency service.

Dependability refers to the ability of the architecture to continue to function even in the event of failures or errors. Twitter's architecture is designed to be highly available and fault-tolerant, with the use of redundancy and failover mechanisms. This ensures that the platform can continue to operate even if individual servers or nodes experience issues.

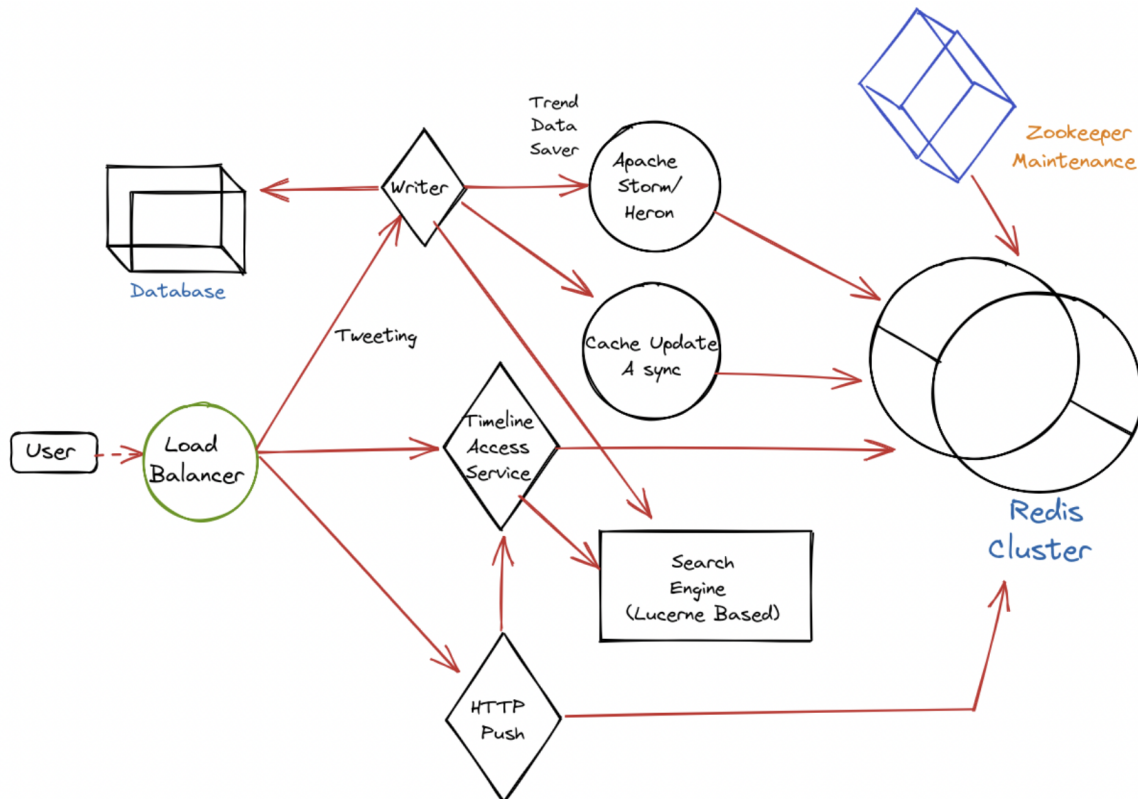
Performance refers to the speed and efficiency of the architecture. Twitter's architecture is designed to provide a low-latency service, with the use of caching and other performance optimization techniques. Additionally, the platform is designed to handle a large number of requests and users concurrently, which helps to ensure that the service remains responsive and fast.



Flexibility refers to the ability of the architecture to adapt and change as the needs of the platform and its users evolve. Twitter's architecture is designed to be modular and extensible, which allows for easy modification and additions of new features or services. Additionally, the platform makes use of a microservices architecture, which allows for individual services to be updated and scaled independently, which helps to ensure the platform remains flexible and adaptable.



In summary, Twitter's architecture is designed with transparency, scalability, dependability, performance, and flexibility in mind. These design principles are achieved through the use of various techniques and design choices, such as distributed systems, data sharding, load balancing, redundancy, and failover mechanisms, caching, and microservices architecture.



From Chapter 4 Coulouris Book

- VI. Answer the following questions using explanation and diagrams as needed. No implementation needed.
1. 4.2 [5 points] A server creates a port that it uses to receive requests from clients. Discuss the design issues concerning the relationship between the name of this port and the names used by clients.

ANS

The main design issues for locating server ports include:

- (a) How clients can determine the appropriate port and IP address to access a service, which can be achieved by using a name server/binder to map the service name to its corresponding port or by assigning well-known location-independent port IDs.
- (b) How to manage service availability when different servers offer the service at different times. This can be done by using location-independent port IDs that allow the service to have the same port at different locations, but if a binder is used, the client needs to consult it again to find the new location.
- (c) Efficiency of access to ports and local identifiers. Operating systems often allow processes to use efficient local names to refer to ports, but this can become an issue when a server creates a non-public port for a particular client, as the local name is not understandable to the client and must be translated to a global identifier for the client to use.

2. 4.15 [10 points] Outline the design of a scheme that uses message retransmissions with IP multicast to overcome the problem of dropped messages. Your scheme should take the following points into account:

- i) There may be multiple senders.
- ii) Generally only a small proportion of messages are dropped.
- iii) Recipients may not necessarily send a message within any particular time limit.

ANS

To allow for point (i) senders must attach a sequence number to each message. Recipients record last sequence number from each sender and check sequence numbers on each message received.

For point (ii) a negative acknowledgement scheme is preferred (recipient requests missing messages, rather than acknowledging all messages). When they notice a missing message, they send a message to the sender to ask for it. To make this work, the sender must store all recently sent messages for retransmission. The sender re-transmits the messages as a unicast datagram.

Point (iii) - refers to the fact that we can't rely on a reply as an acknowledgement. Without acknowledgements, the sender will be left holding all sent messages in its store indefinitely. Possible solutions: a) senders discards stored messages after a time limit b) occasional acknowledgements from recipients which may be piggy backed on messages that are sent.

VII. Java Socket Programming implementation

[25 points]

The goal of this assignment is to implement a TCP client and server. You can use Java. Your TCP or UDP client/server will communicate over the network and exchange data.

The server will start in passive mode listening for a transmission from the client. The client will then start and contact the server (on a given IP address and port number). The client will pass the server a string (eg: "network") up to 80 characters in length.

On receiving a string from a client, the server should: 1) reverse all the characters, and 2) reverse the capitalization of the strings ("network" would now become "KROWTEN").

The server should then send the string back to the client. The client will display the received string and exit.

ANS

This is the TcpServer class, it creates a ServerSocket and binds it to the specified port number. The server enters an infinite loop that listens for new connections using the accept() method. Once a connection is established, a new TcpServerThread is created and started to handle communication with the client.

```
import java.io.*;
import java.net.*;

public class TcpServer {
    public static void main(String[] args) throws IOException {
        if (args.length < 1) {
            System.err.println("Usage: java TcpServer <port number>");
            System.exit(1);
        }
        int portNumber = Integer.parseInt(args[0]);
        boolean listening = true;

        try (ServerSocket serverSocket = new ServerSocket(portNumber)) {
            while (listening) {
                new TcpServerThread(serverSocket.accept()).start();
            }
        } catch (IOException e) {
            System.err.println("Could not listen on port " + portNumber);
            System.exit(-1);
        }
    }
}
```

This is the TcpServerThread class, it handles communication with the client using a PrintWriter to send data and a BufferedReader to receive data. The thread enters an infinite loop that reads data from the client using the readLine()

method and echoes it back using the `println()` method. The thread continues to run and communicate with the client until the client disconnects or sends an end-of-stream signal.

```
packa  
ge  
tcpLo  
ng;
```

```
import java.io.*;  
import java.net.*;  
  
class TcpServerThread extends Thread {  
    private Socket socket = null;  
  
    public TcpServerThread(Socket socket) {  
        super("TcpServerThread");  
        this.socket = socket;  
    }  
  
    public void run() {  
        try {  
            PrintWriter out = new  
PrintWriter(socket.getOutputStream(), true);  
            BufferedReader in = new BufferedReader(  
                new InputStreamReader(  
                    socket.getInputStream()));  
            String inputLine, outputLine;  
            while ((inputLine = in.readLine()) != null) {  
                outputLine = new  
StringBuffer(inputLine).reverse().toString().toUpperCase();  
                ;  
            }  
        }  
    }  
}
```

```

        System.out.println(outputLine);

        out.println(outputLine);

    }

    socket.close();

} catch (IOException e) {

    e.printStackTrace();

}

}

}

```

On the client side, you can use the same TcpClient code that I provided earlier, but keep in mind that the client will need to run indefinitely as well, in order to keep the connection open and continue sending and receiving data.

```

import java.io.*;
import java.net.*;
import java.util.Scanner;

public class TcpClient {

    public static void main(String[] args) throws IOException {

        if (args.length < 2) {

            System.err.println("Usage: java TcpClient <host name> <port number>");

            System.exit(1);

        }

        String hostName = args[0];
        int portNumber = Integer.parseInt(args[1]);

        try (

            Socket socket = new Socket(hostName, portNumber);

            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

            BufferedReader in = new BufferedReader(

                new InputStreamReader(socket.getInputStream()));

        ) {

            Scanner scanner = new Scanner(System.in);

            String inputLine, outputLine;

            while (true) {

                System.out.print("Enter message: ");

                inputLine = scanner.nextLine();

```

```

        out.println(inputLine);
        outputLine = in.readLine();
        System.out.println("Server: " + outputLine);
    }
} catch (UnknownHostException e) {
    System.err.println("Unknown host " + hostName);
    System.exit(1);
} catch (IOException e) {
    System.err.println("Couldn't get I/O for the connection to " + hostName);
    System.exit(1);
}
}
}

```

Running the server

The screenshot shows an IDE with the following components:

- Editor:** Displays `TcpServer.java` with the following code:

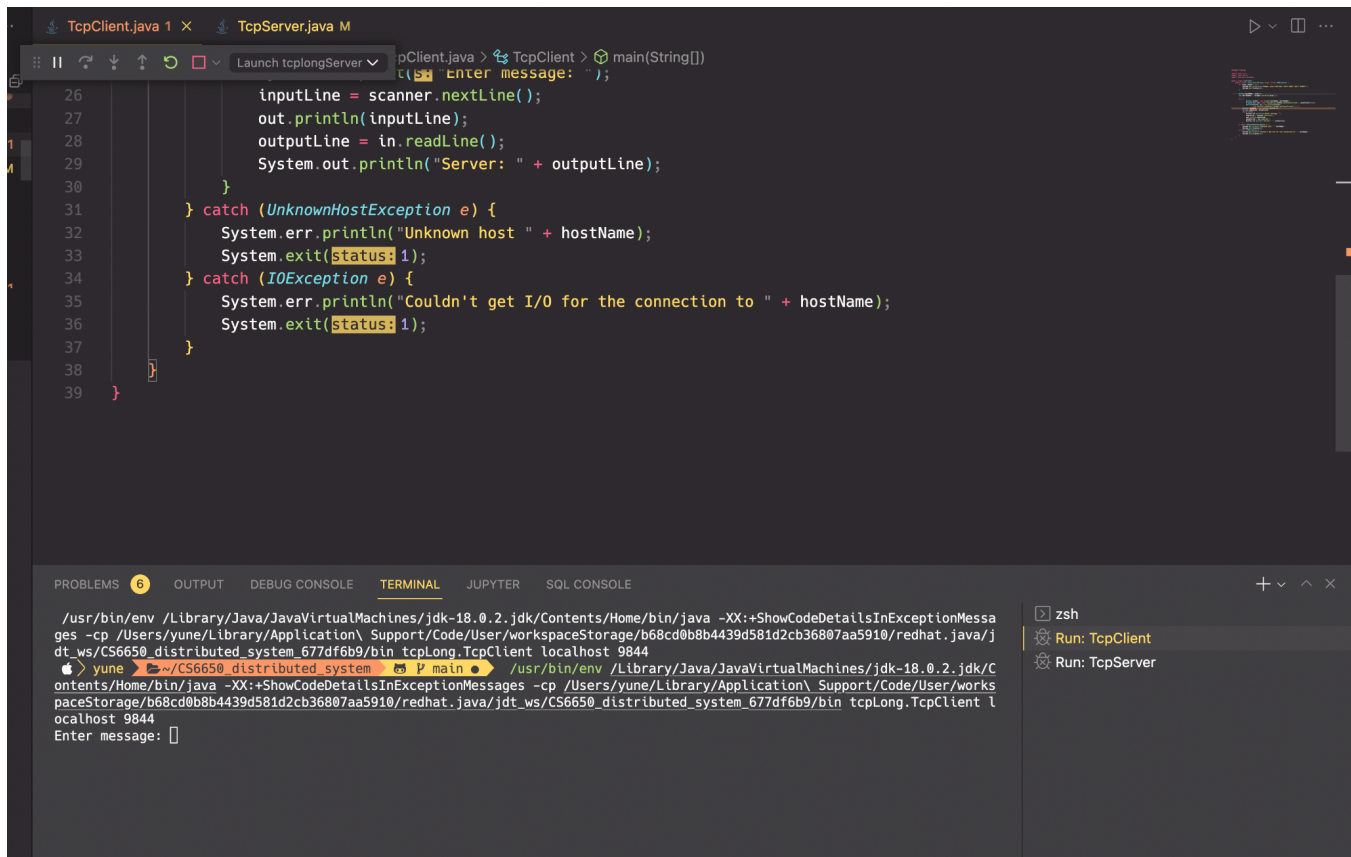

```

1  import java.net.*;
2
3  public class TcpServer {
4
5      public static void main(String[] args) throws IOException {
6          if (args.length < 1) {
7              System.err.println("Usage: java TcpServer <port number>");
8              System.exit(1);
9          }
10         int portNumber = Integer.parseInt(args[0]);
11         boolean listening = true;
12         System.out.println("listening on port " + portNumber);
13         try (ServerSocket serverSocket = new ServerSocket(portNumber)) {
14             while (listening) {
15                 new TcpServerThread(serverSocket.accept()).start();
16             }
17         } catch (IOException e) {
18             System.err.println("Could not listen on port " + portNumber);
19             System.exit(-1);
20         }
21     }
22 }
      
```
- Terminal:** Shows the command execution and output:


```


/usr/bin/env /Library/Java/JavaVirtualMachines/jdk-18.0.2.jdk/Contents/Home/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/yune/Library/Application\ Support/Code/User/workspaceStorage/b68cd0b8b4439d581d2cb36807aa5910/redhat.java/jdt_ws/CS6650_distributed_system_677df6b9/bin tcpLong.TcpServer 9844
yune ~/CS6650_distributed_system main • /usr/bin/env /Library/Java/JavaVirtualMachines/jdk-18.0.2.jdk/Contents/Home/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/yune/Library/Application\ Support/Code/User/workspaceStorage/b68cd0b8b4439d581d2cb36807aa5910/redhat.java/jdt_ws/CS6650_distributed_system_677df6b9/bin tcpLong.TcpServer 9844
listening on port 9844
      
```
- Run Configuration:** On the right, it shows two configurations: `Run: TcpClient` and `Run: TcpServer`, with `Run: TcpServer` currently selected.

Running the client



The screenshot shows the VS Code editor with the `TcpClient.java` file open. The code is a Java client that connects to a server at `localhost 9844`. It uses a `Scanner` to read input from the user and sends it to the server. The server's response is printed to the console. The terminal window at the bottom shows the command to run the client: `java -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/yune/Library/Application\ Support/Code/User/workspaceStorage/b68cd0b8b4439d581d2cb36807aa5910/redhat.java/jdt_ws/CS6650_distributed_system_677df6b9/bin tcpLong.TcpClient localhost 9844`. The prompt "Enter message:" is visible.

Output



The screenshot shows the terminal output of the `TcpClient` program. The user enters the message "evres EHT YB DEGNAHC EB OT TXET YM SI SIHT". The server responds with "Server: This is my text to be changed by the SERVER". The user enters the message "revres EHT YB DEGNAHC EB OT TXET YM SI SIHT". The server responds with "Server: revres EHT YB DEGNAHC EB OT TXET YM SI SIHT".

I attached my code to github. Here's the link. You can also see the readme in the page.

https://github.com/zyune/CS6650_distributed_system/tree/main/stage1/Socketprograming/src/tcpLong.

I also put launch.json to the github. if you are using vscode as your editor feel free to use it.