Lab Report

CSE 2213: Data and Telecommunication Lab
Batch: 29/2nd Year 2nd Semester 2024

Report's Title: Implementation of Multiplexing and Demultiplexing using Statistical TDM.

Submitted By:

Suraya Jannat Mim (Roll: 17)
Anisha Tabassum (Roll: 19)

Course Instructors:

Dr. Md. Mustafizur Rahman (MMR)
Mr. Palash Roy (PR)

1. Introduction

In data communication systems, **Multiplexing (MUX)** and **Demultiplexing (DEMUX)** are essential techniques that allow multiple signals to be transmitted over a single communication channel and then separated at the receiver end. These processes increase the efficiency of the communication system by optimizing the use of bandwidth.

Types of Multiplexing:

1. **Time Division Multiplexing (TDM)** – Assigns fixed time slots to each signal. Each sender is allocated a fixed time slot in a round-robin fashion. If the sender has no data, the time slot may be wasted (in Synchronous TDM) or reassigned (in Statistical TDM).
2. **Frequency Division Multiplexing (FDM)** – Assigns different frequency bands. Different signals are transmitted over different frequency bands within the same channel. It is widely used in radio, television broadcasting, and cable TV.
3. **Statistical Multiplexing** – Dynamically allocates time slots based on demand.
4. **Wavelength Division Multiplexing (WDM)** – Used in fiber optics, with different wavelengths.

Types of Demultiplexing:

Each MUX method has a corresponding DEMUX process that reverses the multiplexing at the receiver.

Synchronous TDM (Sync TDM):

In synchronous TDM, time slots are pre-assigned to each input device whether they have data to send or not. This may cause wastage of bandwidth.

Statistical TDM (Stat TDM):

Statistical TDM dynamically allocates time slots to only those devices that have data to transmit. This leads to better bandwidth utilization and efficiency.

2. Objectives

- To understand the concepts of multiplexing and demultiplexing.
- To implement **Statistical Time Division Multiplexing (Stat TDM)** using a client-server model.
- To compare the efficiency and performance of Stat TDM with Synchronous TDM.

3. Algorithms/Pseudocode

Server-Side Steps:

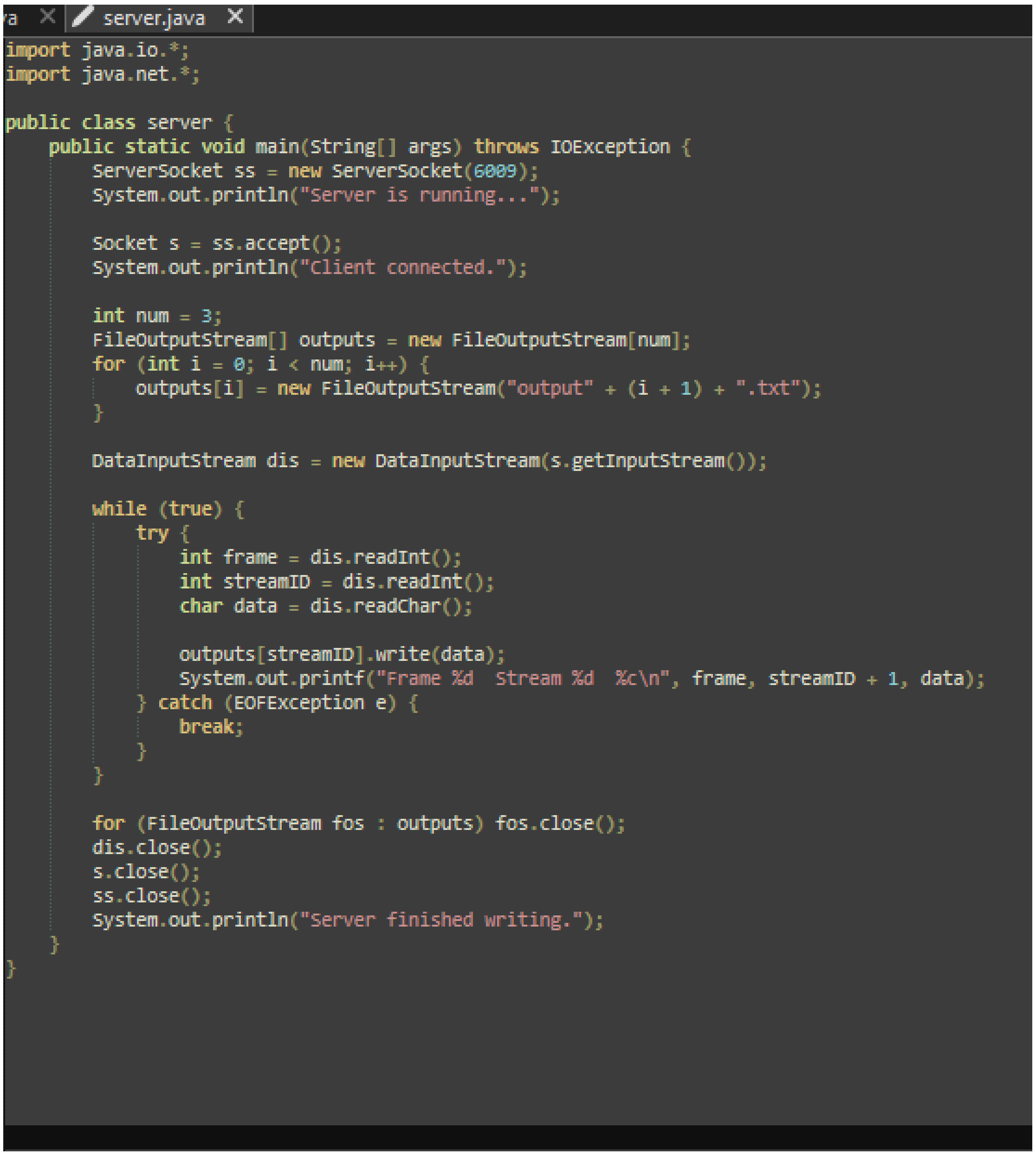
1. Start the server socket and wait for client connections.
2. Accept multiple client connections.
3. Continuously check which clients have data to send.
4. Create frames dynamically based on available data (Stat TDM).
5. Send multiplexed frames to a virtual receiver (simulate DEMUX).
6. Display demultiplexed data with client identifiers.

Client-Side Steps:

1. Connect to the server.
2. Input data to be sent.
3. Wait for allocation in the Stat TDM frame.
4. Send data when allocated.
5. Display acknowledgment or response from server.

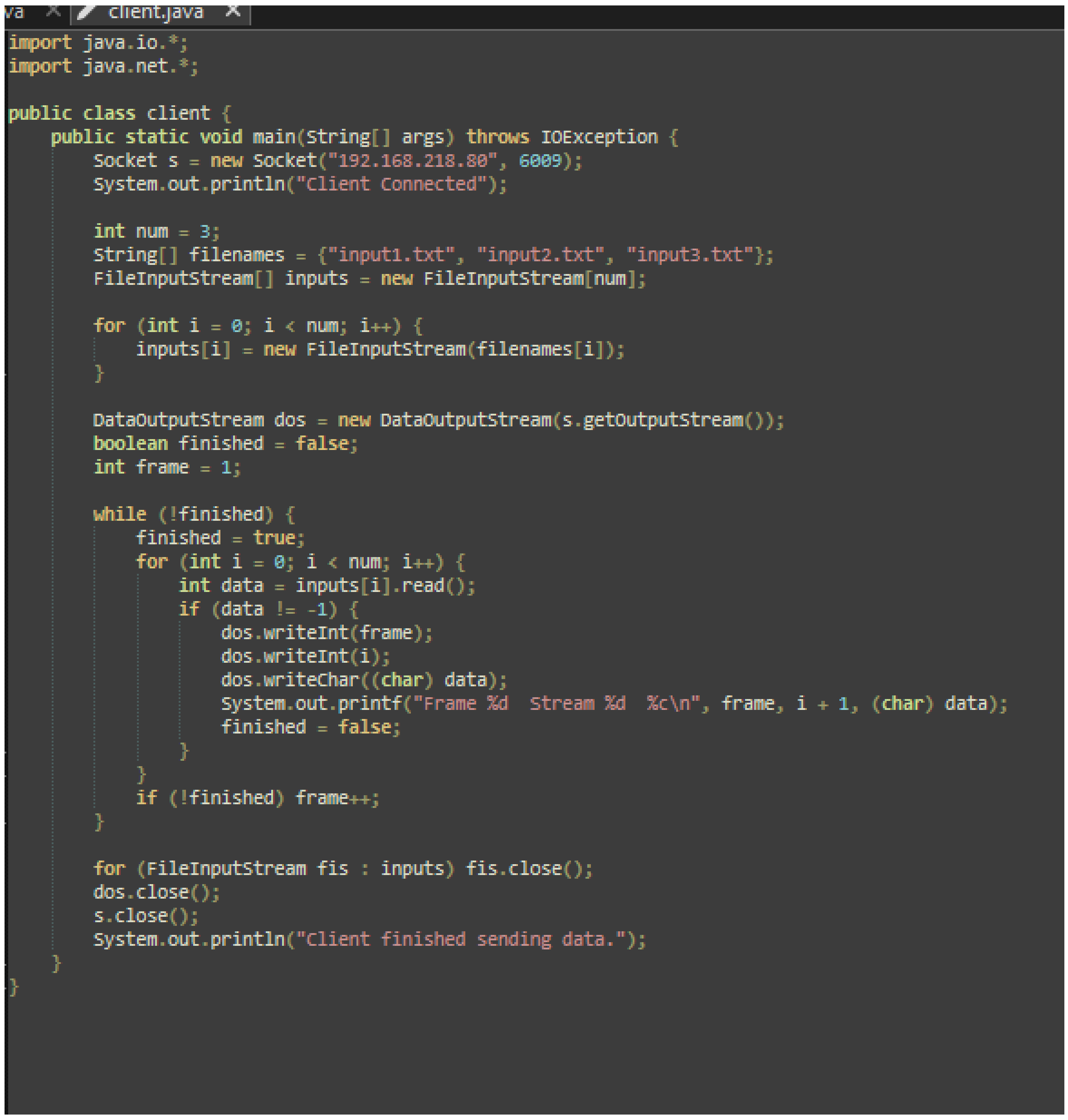
4. Implementation

Server Side:


a × server.java ×

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

public class server {
    public static void main(String[] args) throws IOException {
        ServerSocket ss = new ServerSocket(6009);
        System.out.println("Server is running...");

        Socket s = ss.accept();
        System.out.println("Client connected.");

        int num = 3;
        FileOutputStream[] outputs = new FileOutputStream[num];
        for (int i = 0; i < num; i++) {
            outputs[i] = new FileOutputStream("output" + (i + 1) + ".txt");
        }

        DataInputStream dis = new DataInputStream(s.getInputStream());

        while (true) {
            try {
                int frame = dis.readInt();
                int streamID = dis.readInt();
                char data = dis.readChar();

                outputs[streamID].write(data);
                System.out.printf("Frame %d  Stream %d  %c\n", frame, streamID + 1, data);
            } catch (EOFException e) {
                break;
            }
        }

        for (FileOutputStream fos : outputs) fos.close();
        dis.close();
        s.close();
        ss.close();
        System.out.println("Server finished writing.");
    }
}
```

Client Side:



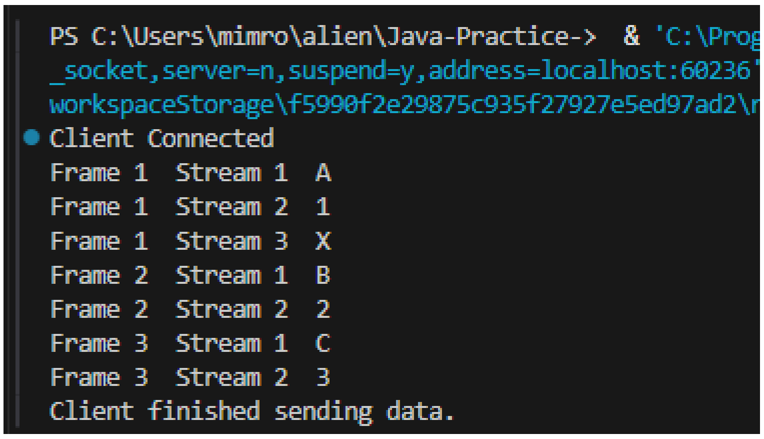
va ✕ client.java ✕

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

public class client {
    public static void main(String[] args) throws IOException {
        Socket s = new Socket("192.168.218.80", 6009);
        System.out.println("Client Connected");

        int num = 3;
        String[] filenames = {"input1.txt", "input2.txt", "input3.txt"};
        FileInputStream[] inputs = new FileInputStream[num];

        for (int i = 0; i < num; i++) {
            inputs[i] = new FileInputStream(filenames[i]);
        }

        DataOutputStream dos = new DataOutputStream(s.getOutputStream());
        boolean finished = false;
        int frame = 1;

        while (!finished) {
            finished = true;
            for (int i = 0; i < num; i++) {
                int data = inputs[i].read();
                if (data != -1) {
                    dos.writeInt(frame);
                    dos.writeInt(i);
                    dos.writeChar((char) data);
                    System.out.printf("Frame %d  Stream %d  %c\n", frame, i + 1, (char) data);
                    finished = false;
                }
            }
            if (!finished) frame++;
        }

        for (FileInputStream fis : inputs) fis.close();
        dos.close();
        s.close();
        System.out.println("Client finished sending data.");
    }
}
```

5. Result Analysis

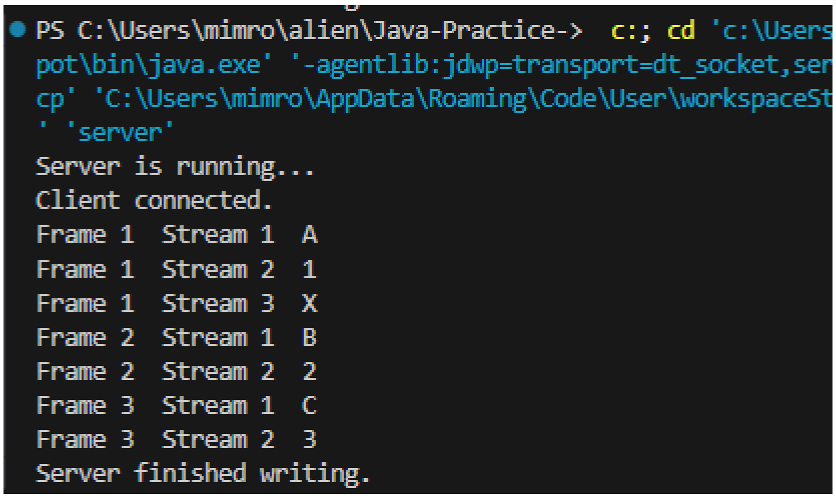
Outputs:

Client's Output:



```
PS C:\Users\mimro\alien\Java-Practice-> & 'C:\Prog
_socket,server=n,suspend=y,address=localhost:60236'
workspaceStorage\f5990f2e29875c935f27927e5ed97ad2\r
Client Connected
Frame 1  Stream 1  A
Frame 1  Stream 2  1
Frame 1  Stream 3  X
Frame 2  Stream 1  B
Frame 2  Stream 2  2
Frame 3  Stream 1  C
Frame 3  Stream 2  3
Client finished sending data.
```

Server's Output:



```
PS C:\Users\mimro\alien\Java-Practice->  c:; cd 'c:\Users
pot\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,ser
cp' 'C:\Users\mimro\AppData\Roaming\Code\User\workspaceSt
' 'server'
Server is running...
Client connected.
Frame 1  Stream 1  A
Frame 1  Stream 2  1
Frame 1  Stream 3  X
Frame 2  Stream 1  B
Frame 2  Stream 2  2
Frame 3  Stream 1  C
Frame 3  Stream 2  3
Server finished writing.
```

Explanation: The server only includes clients with actual data in each frame, which optimizes bandwidth usage—a characteristic of Stat TDM.

6. Discussion

Criteria	Synchronous TDM	Statistical TDM
Slot Allocation	Fixed and preassigned	Dynamic based on demand
Efficiency	Lower, bandwidth wastage	Higher, efficient usage
Complexity	Simpler to implement	More complex (needs queueing)
Idle Slot Handling	Present	Avoided
Implementation	Easier (fixed scheduling	Requires checking buffer state

Implementation-wise, Sync TDM is easier due to its fixed scheduling. However, Stat TDM, while slightly complex, is more efficient, particularly under bursty traffic conditions.

---

7. Learning and Difficulties

What We Learned:

- How to use socket programming for data transmission.
- Difference between static and dynamic multiplexing.
- Practical implementation of Stat TDM and DEMUX.
- Handling multiple client communications concurrently.

Difficulties Faced:

- Managing concurrency and synchronization in the server.
- Handling multiple client connections and partial data issues.
- Designing dynamic frames based on incoming client data.

---

8. Conclusion

In this lab, we successfully implemented a **Statistical Time Division Multiplexing** system using a client-server model. The server dynamically created frames only for clients with data, demonstrating improved bandwidth efficiency over Synchronous TDM. This experiment helped us better understand real-world data communication techniques and their practical challenges.