Department of
Computer Science and Engineering

University of Dhaka

Title: Implemention of Multiplexing and Demultiplexing in Data Communication

CSE 2213: Data and Telecommunication Lab

Batch: 29/2nd Year 2nd Semester 2024

Course Instructors

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

1 Objective(s)

- To gather knowledge on the concept of Multiplexing (MUX) and Demultiplexing (DEMUX) in data communication.
- To simulate how multiple data streams can be sent over a shared communication medium.
- To demonstrate how demultiplexing helps identify and deliver data to the correct application.

2 Background Theory

Multiplexing is a technique that combines multiple signals into one signal over a shared medium. It optimizes the use of bandwidth by sharing a single physical channel among multiple logical streams. Real-world example of multiplexing can be thought of as multiple phone calls transmitted over a single fiber optic cable, or multiple video/audio streams combined over one internet connection. Different types of Multiplexing are as follows,

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.

Demultiplexing is the reverse process of multiplexing. It separates the combined signal and delivers it to the appropriate receiver or process based on identifiers like port numbers or channel IDs. A visual representation of the MUX and DEMUX processes in data communication is illustrated in Fig. 1.

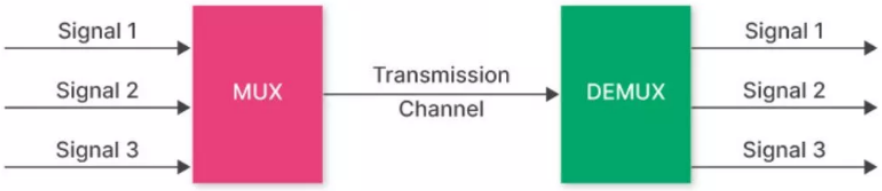

Figure 1: Illustration of MUX and DEMUX in data communication

3 Experiment Overview

In this lab, we will implement **Time Division Multiplexing (TDM)** by a client-side multiplexer to combine multiple data streams and a server-side demultiplexer to separate and reconstruct the streams. The key concepts of TDM can be summarized as follows:

3.1 Multiplexing (Client Side):

- Time Slot (T): The number of bytes read from each file before moving to the next. In each time slot, one byte of data will be read from one file from N input files.
- Aggregating T bytes from each source sequentially using Time Division Multiplexing (TDM). Then, the individual chunks are combined into a transmission unit (a single packet).
- Garbage Byte Handling: If a file reaches EOF before the others, fill its slot with a special byte (e.g., '#') to maintain synchronization.

3.2 Demultiplexing (Server Side)

- Receives the multiplexed byte stream.
- Splitting the incoming stream into logical chunks for each source and writing valid (non-garbage) bytes back to their respective output files to reconstruct files.

3.3 Example Scenario

Assume you have 3 input files: input1.txt, input2.txt, and input3.txt. Each file contains:
input1.txt: "ABC"
input2.txt: "1234"
input3.txt: "xy"
Now, if we assume $T = 2$, then, Multiplexed Packets (Client Side)
1st packet: "AB", "12", "xy" → ['A','B','1','2','x','y']
2nd packet: "C#", "34", "##" → ['C','#','3','4','#','#'] (Here, # represents garbage/padding bytes due to EOF.)
Reconstructed Output (Server Side):
output1.txt: "ABC"
output2.txt: "1234"
output3.txt: "xy"

3.4 Client Side Algorithm for Multiplexing

- Create a Socket object that takes the IP address and port number as input.
- Create an object of the DataOutputStream class, which is used to send data to the server side.
- Open N input files.
- For each round:
  - Read one byte from each file.
  - If EOF, insert special byte '#'.
  - Create a byte array from N bytes.
  - Send the byte array to the server.
- Stop when all files have reached EOF.

3.5 Server Side Algorithm for Demultiplexing

- Create a ServerSocket object, namely handshaking socket, which takes a port number as input.
- Create a plain Socket object that accepts client requests
- Create an object of the DataInputStream class, which is used to read data
- Create N output files.
- For each received packet:
  - Extract N bytes.
  - For each byte:
    - Ignore garbage byte '#'
    - Write a valid byte to the corresponding output file.
- Repeat until client closes connection.

4 Input/Output

Sample Input and Output of the above experiment can be as follows

Input Files

```
input1.txt: "ABCDEF"
input2.txt: "12345"
input3.txt: "xyz"
```

Transmission packets for T=2 (Client Side):

```
Client connected to the server on Handshaking port 5000
Client's Communication Port: 56190
Client is Connected
Round 1: AB 12 xy
Round 2: CD 34 z#
Round 3: EF 5# ##
```

The output of the server-side demultiplexing

```
Server is connected at port no: 5000
Server is connecting
Waiting for the client
Client request is accepted at port no: 56190
Server's Communication Port: 5000
output1.txt: ABCDEF
output2.txt: 12345
output3.txt: xyz
```

5 Home Task (Submit as a lab report)

Simulate the above MUX and DEMUX concept in data communication for Statistical TDM (Stat-TDM).

5.1 Problem Analysis

in Stat-TDM, time slots are dynamically assigned based on which sources have data to send and dynamically allocated only to sources that have data to send. Here, frame size is smaller or variable, and only active sources are included. A header or address field is required to identify which source file each byte belongs to.

In brief, in Stat-TDM, variable or compact frame sizes are used. Each data unit includes a header or ID (adds overhead).

5.1.1 Client Side Algorithm for Multiplexing

- Open N input files.
- Initialize an empty frame.
- For each source:
  - If it has data available:
    - Read 1 byte.
    - Create a data unit: Stream ID, Data.
    - Add to frame.
- Send/store the frame.

5.1.2 Server Side Algorithm for Demultiplexing

- For each frame:
  - Parse Stream ID, Data pairs.
  - Route data to corresponding output files using stream IDs.

5.2 Example Scenario

Assume you have 3 input files: input1.txt, input2.txt, and input3.txt. Each file contains:
input1.txt: "ABC"
input2.txt: "123"
input3.txt: "X".
Now, an example scenario of Stat-TDM is illustrated in Table 1.

Table 1: Statistical TDM Frame Structure with Stream IDs and Data

Frame #	**Stream ID**	**Data**
1	1	A
	2	1
	3	X
2	1	B
	2	2
3	1	C
	2	3

6 Discussion & Conclusion

Based on the focused objective(s), this task will help us to demonstrate how single-level time division multiplexing can combine multiple data streams and reconstruct them accurately, even with uneven data lengths.

7 Policy

Copying from the Internet, classmates, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.