Experiment 4–Group Communication

Aim: Student should be able to develop a program for Group communication

Tools : Java

Theory:

Group Communication

The most elementary form of message-based interaction is one-to-one communication (also known as point-to-point, or unicast, communication) in which a single-sender process sends a message to a single-receiver process. However, for performance and ease of programming, several highly parallel distributed applications require that a message passing system should also provide group communication facility.

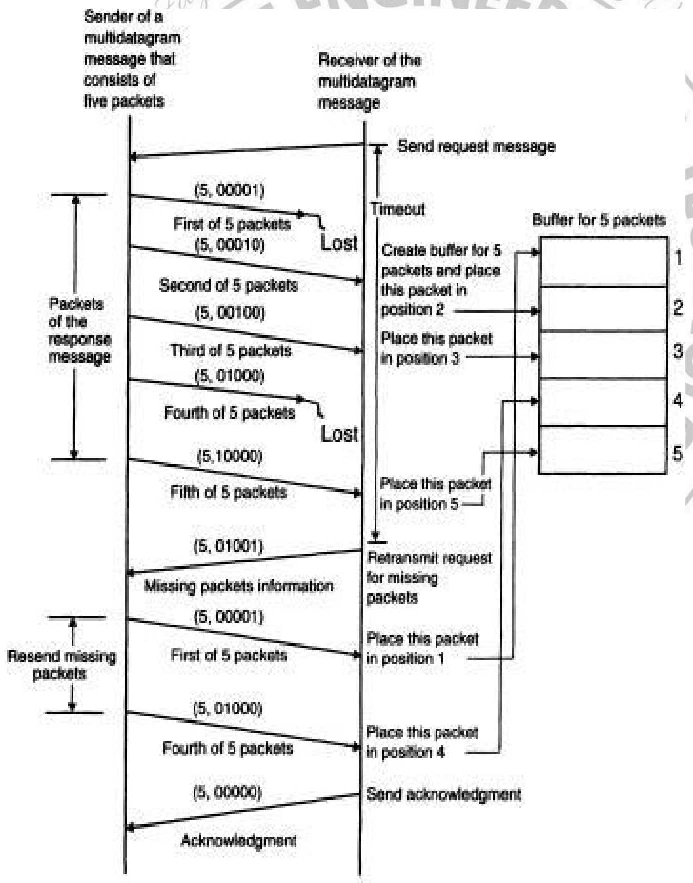

Sender of a multidatagram message that consists of five packets
Receiver of the multidatagram message
Send request message
Timeout
Buffer for 5 packets
(5, 00001)
First of 5 packets
Lost
(5, 00010)
Second of 5 packets
Create buffer for 5 packets and place this packet in position 2
Packets of the response message
(5, 00100)
Third of 5 packets
Place this packet in position 3
(5, 01000)
Fourth of 5 packets
Lost
(5,10000)
Fifth of 5 packets
Place this packet in position 5
1
2
3
4
5
(5, 01001)
Missing packets information
Retransmit request for missing packets
Resend missing packets
(5, 00001)
First of 5 packets
Place this packet in position 1
(5, 01000)
Fourth of 5 packets
Place this packet in position 4
(5, 00000)
Send acknowledgment
Acknowledgment

Fig. 3.13 An example of the use of a bitmap to keep track of lost and out of sequence packets in a multidatagram message transmission.

Depending on single or multiple senders and receivers, the following three types of group communication are possible:

1. One to many (single sender and multiple receivers)

2. Many to one (multiple senders and single receiver)

3. Many to many (multiple senders and multiple receivers)

One-to-Many Communication

In this scheme, there are multiple receivers for a message sent by a single sender. One-to-many scheme is also known as multicast communication. A special case of multicast communication is broadcast communication, in which the message is sent to all processors connected to a network. Multicast/broadcast communication is very useful for several practical applications.

For example, consider a server manager managing a group of server processes all providing the same type of service. The server manager can multicast a message to all the server processes, requesting that a free server volunteer to serve the current request. It then selects the first server that responds. The server manager does not have to keep track of the free servers. Similarly, to locate a processor providing a specific service, an inquiry message may be broadcast. In this case, it is not necessary to receive an answer from every processor; just finding one instance of the desired service is sufficient.

Many-to-One Communication

In this scheme, multiple senders send messages to a single receiver. The single receiver may be selective or nonselective. A selective receiver specifies a unique sender; a message exchange takes place only if that sender sends a message. On the other hand, a nonselective receiver specifies a set of senders, and if anyone sender in the set sends a message to this receiver, a message exchange takes place.

Thus we see that an important issue related to the many-to-one communication scheme is nondeterminism. The receiver may want to wait for information from any of a group of senders, rather than from one specific sender. As it is not known in advance which member (or members) of the group will have its information available first, such behavior is nondeterministic. In some cases it is useful to dynamicalJy control the group of senders from whom to accept message. For example, a buffer process may accept a request from a producer process to store an item in the buffer whenever the buffer is not full; it may accept a request from a consumer process to get an item from the buffer whenever the buffer is not empty. To program such behavior, a notation is needed to express and control nondeterminism. One such construct is the "guarded command" statement introduced by Dijkstra

Many-to-Many Communication

In this scheme, multiple senders send messages to multiple receivers. The one-to-many and many-to-one schemes are implicit in this scheme. Hence the issues related to one-to-many and many-to-one schemes, which have already been described above, also apply to the many-to- many communication scheme. In addition, an important issue related to many-to-many communication scheme is that of ordered message delivery.

Ordered message delivery ensures that all messages are delivered to all receivers in an order acceptable to the application. This property is needed by many applications for their correct functioning. For example, suppose two senders send messages to update the same record of a database to two server processes having a replica of the database. If the messages of the two senders are received by the two servers in different orders, then the final values of the updated record of the database may be different in its two replicas. Therefore, this application requires that all messages be delivered in the same order to all receivers.

Ordered message delivery requires message sequencing. In a system with a single sender and multiple receivers (one-to-many communication), sequencing messages to all the receivers is trivial. If the sender initiates the next multicast transmission only after confirming that the previous multicast message has been received by all the members, the messages win be delivered in the same order. On the other hand. in a system with multiple senders and a single receiver (many-to-one communication), the messages will be delivered to the receiver in the order in which they arrive at the receiver's machine.

Ordering in this case is simply handled by the receiver. Thus we see that it is not difficult to ensure ordered delivery of messages in many-to-one or one-to-many communication schemes.

However, in many-to-many communication, a message sent from a sender may arrive at a receiver's destination before the arrival of a message from another sender; but this order may be reversed at another receiver's destination (see Fig. 3.14). The reason why messages of different senders may arrive at the machines of different receivers in different orders is that when two processes are contending for access to a LAN, the order in which messages of the two processes are sent over the LAN is nondeterministic. Moreover, in a WAN environment, the messages of different senders may be routed to the same destination using different routes that take different amounts of time (which cannot be correctly predicted) to the destination. Therefore, ensuring ordered message delivery requires a special message-handling mechanism in many-to-many communication scheme.

The commonly used semantics for ordered delivery of multicast messages are absolute ordering, consistent ordering, and causal ordering.

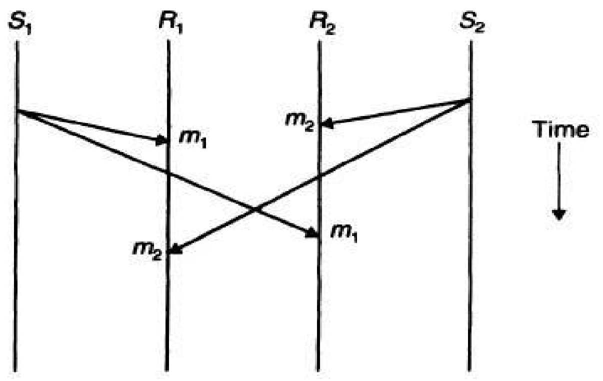


Fig. 3.14 No ordering constraint for message delivery.

Result and Discussion:

Server.java:

```
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.io.PrintWriter;

public class Server {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(12345);
            System.out.println("Server started. Waiting for clients...");

            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("Client connected from: " +
clientSocket.getInetAddress().getHostName());
                PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
                out.println("Hi this comp c 41");
                clientSocket.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Client.java:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;
```

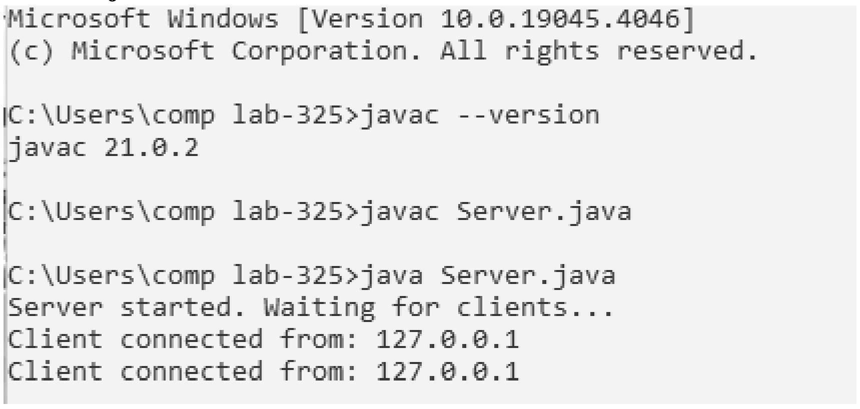
```
public class Client {
    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 12345);
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            String message = in.readLine();
            System.out.println("Message from server: " + message);
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Client1.java:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;

public class Client1 {
    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 12345);
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            String message = in.readLine();
            System.out.println("Message from server: " + message);
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output:
Server.java



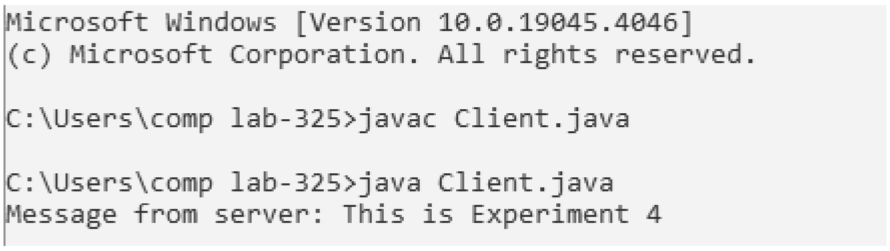
```
Microsoft Windows [Version 10.0.19045.4046]
(c) Microsoft Corporation. All rights reserved.

C:\Users\comp lab-325>javac --version
javac 21.0.2

C:\Users\comp lab-325>javac Server.java

C:\Users\comp lab-325>java Server.java
Server started. Waiting for clients...
Client connected from: 127.0.0.1
Client connected from: 127.0.0.1
```

Client.java

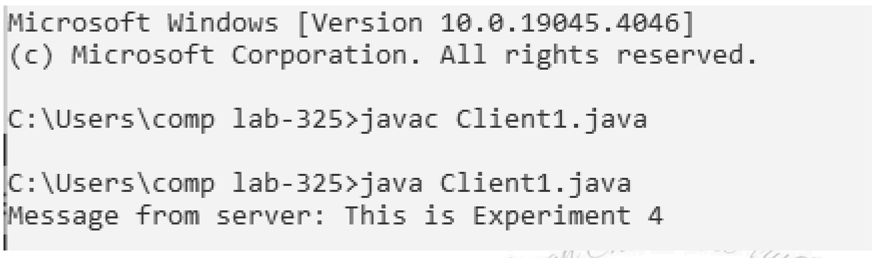


```
Microsoft Windows [Version 10.0.19045.4046]
(c) Microsoft Corporation. All rights reserved.

C:\Users\comp lab-325>javac Client.java

C:\Users\comp lab-325>java Client.java
Message from server: This is Experiment 4
```

Client.java



```
Microsoft Windows [Version 10.0.19045.4046]
(c) Microsoft Corporation. All rights reserved.

C:\Users\comp lab-325>javac Client1.java

C:\Users\comp lab-325>java Client1.java
Message from server: This is Experiment 4
```

Learning Outcomes: The student should have the ability to

LO1: Describe the protocol for Group communication.

LO2: Compare the different protocol techniques used in group communication.

Course Outcomes: Upon completion of the course students will be able to group communication.

Conclusion: After performing the experiment I was able to describe the protocol for group communication and also compare the different protocol techniques used in group communication.

For Faculty Use

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance / Learning Attitude [20%]**	
Marks Obtained				