

Distributed Computing

Experiment – 3

Program to demonstrate Group Communication using Java	
Learning Objective:	To understand basic underlying concepts of forming distributed systems.
Learning Outcome:	Ability to demonstrate Group Communication.
Course Outcome:	CSL801.1
Program Outcome:	<p>(PO1) Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.</p> <p>(PO2) Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.</p> <p>(PO3) Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations</p>
Bloom's Taxonomy Level:	Analysis, Create
Theory:	<p>Group Communication in Distributed Computing -</p> <p>In a distributed system, communication between two processes is essential for exchanging various types of data, such as code or files. When a source process attempts to communicate with multiple processes simultaneously, this is termed Group Communication. A group consists of interconnected processes with an</p>

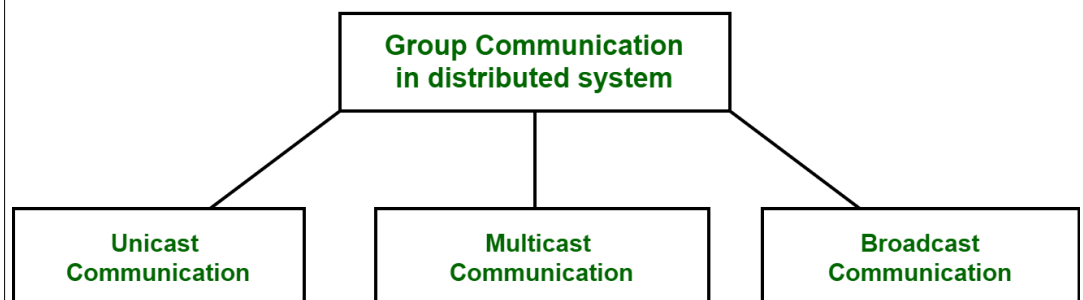
abstraction designed to conceal the intricacies of message passing, presenting communication as a standard procedure call. Group communication facilitates collaboration among processes from different hosts, enabling them to work together in a synchronized manner and ultimately enhancing the overall system performance.

Group Communication Properties:

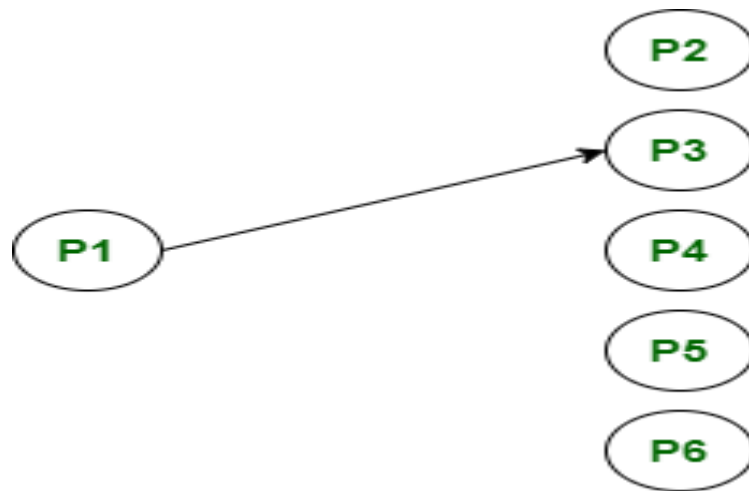
An integral aspect of group communication is atomicity, also referred to as the all-or-nothing property. This property ensures that if one or more members of a group encounter issues receiving a message, the process sending the message will receive an error notification. The ordering property manages the sequence of message delivery, with various types of message ordering including no order, FIFO ordering, casual ordering, and total ordering.

- **No order:** messages are sent to the group without concern for ordering.
- **FIFO ordering:** messages are delivered in the order in which they were sent.
- **Casual ordering:** messages are sent after receiving another message.
- **Total ordering:** all group members receive all messages in the same order.

Types of Group Communication in a Distributed System:

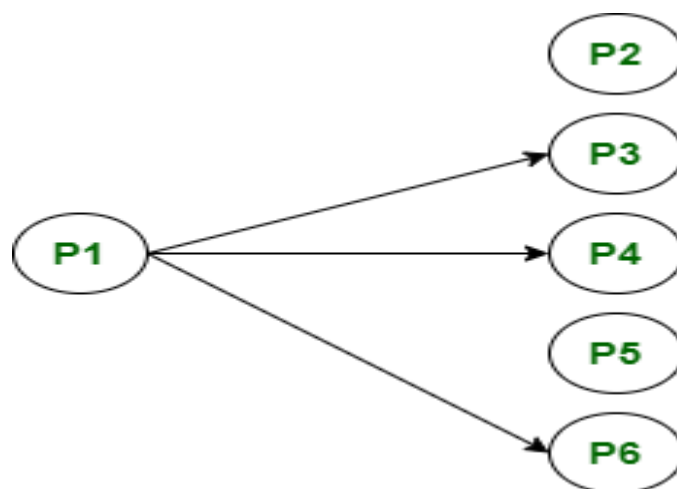


1. Unicast Communication:



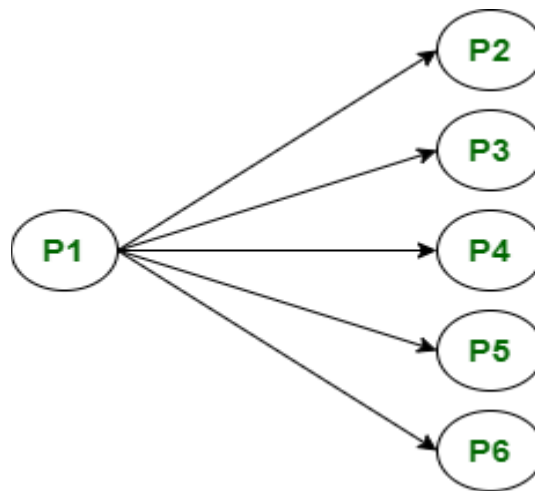
In unicast communication, the host process endeavors to communicate with a single process in a distributed system simultaneously, even though the same information may be conveyed to multiple processes. This mode is most efficient when two processes are involved in communication, as it exclusively addresses a specific process. However, the drawback lies in the overhead it incurs, requiring the identification of the exact process before exchanging information or data.

2. Multicast Communication:



Multicast communication occurs when the host process aims to communicate with a designated group of processes in a distributed system simultaneously. This technique is primarily employed to address the challenges of high workload on the host system and the redundancy of information from processes in the system. Multicast communication proves beneficial in reducing the time taken for message handling through efficient multitasking.

3. Broadcast Communication:



Broadcast communication occurs when the host process attempts to communicate with every process in a distributed system simultaneously. This mode is particularly useful when efficiently delivering a common stream of information to each process is essential. The communication is exceptionally fast compared to other modes, as it does not involve any processing. However, it has limitations, as it does not support a large number of processes and cannot address specific processes individually.

Program :

- **Server.java**

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
```



```
        writer.println(input);
    }
}
} catch (Exception e) {
    System.err.println(e);
}
}
}
```

- **Master.java**

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

public class master {
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        Socket socket = new Socket("localhost", 9001);
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        System.out.print("Enter your name: ");
        String name = sc.nextLine();

        out.println(name + " (Master)");
    }
}
```

```

Thread readerThread = new Thread() -> {
    try {
        while (true) {
            String line = in.readLine();
            if (line != null && !line.isEmpty()) {
                System.out.println(line);
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
});
readerThread.start();

while (true) {
    System.out.print("Enter a message: ");
    String message = sc.nextLine();
    out.println(name + ": " + message);
}
}
}

```

- **Slave1.java**

```

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

public class slave1 {
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
    }
}

```

```

Socket socket = new Socket("localhost", 9001);
BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
System.out.print("Enter your name: ");
String name = sc.nextLine();

out.println(name + " (Slave)");

Thread readerThread = new Thread(() -> {
    try {
        while (true) {
            String line = in.readLine();
            if (line != null && !line.isEmpty()) {
                System.out.println(line);
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
});
readerThread.start();

while (true) {
    System.out.print("Enter a message: ");
    String message = sc.nextLine();
    out.println(name + ": " + message);
}
}

```

- **Slave2.java**


```

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

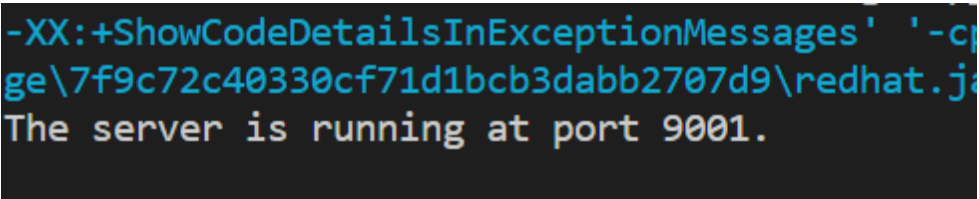
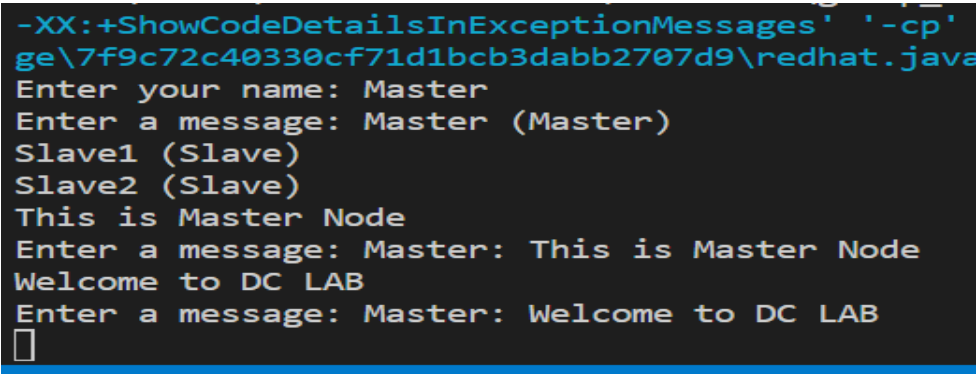
public class slave2 {
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        Socket socket = new Socket("localhost", 9001);
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        System.out.print("Enter your name: ");
        String name = sc.nextLine();

        out.println(name + " (Slave)");

        Thread readerThread = new Thread(() -> {
            try {
                while (true) {
                    String line = in.readLine();
                    if (line != null && !line.isEmpty()) {
                        System.out.println(line);
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        });
        readerThread.start();

        while (true) {

```

	<pre> System.out.print("Enter a message: "); String message = sc.nextLine(); out.println(name + ": " + message); } } } </pre>
Outcome :	<p>Server.java</p>  <pre> -XX:+ShowCodeDetailsInExceptionMessages' '-cp' ge\7f9c72c40330cf71d1bcb3dabb2707d9\redhat.java The server is running at port 9001. </pre> <p>Master.java</p>  <pre> -XX:+ShowCodeDetailsInExceptionMessages' '-cp' ge\7f9c72c40330cf71d1bcb3dabb2707d9\redhat.java Enter your name: Master Enter a message: Master (Master) Slave1 (Slave) Slave2 (Slave) This is Master Node Enter a message: Master: This is Master Node Welcome to DC LAB Enter a message: Master: Welcome to DC LAB </pre> <p>Slave1.java</p>

	 <pre> Enter your name: Slave1 Enter a message: Slave1 (Slave) Slave2 (Slave) Master: This is Master Node Master: Welcome to DC LAB □ </pre> <p>Slave2.java</p>  <pre> Enter your name: Slave2 Enter a message: Slave2 (Slave) Master: This is Master Node Master: Welcome to DC LAB □ </pre>
Conclusion:	<p>In this experiment, a successful group communication is performed so which is guided by Server and controlled by the master program which distributes messages to the 2 slave programs.</p>
References:	<p>[1] https://www.geeksforgeeks.org/group-communication-in-distributed-systems/</p> <p>[2] https://www.educative.io/answers/what-is-group-communication-in-distributed-systems</p> <p>[3] https://www.cambridge.org/core/books/abs/distributed-computing/message-ordering-and-group-communication/07A2B65ACB4D9B30919EC0324A50396B</p>

Rubrics for Assessment

Timely Submission	Submitted after 2 weeks 0	Submitted after deadline 1	On time Submission 2
Understanding	Student is confused about the concept 0	Students has justifiably understood the concept 2	Students is very clear about the concepts 3
Performance	Students has not performed the Experiment 0	Student has performed with help 2	Student has independently performed the experiment 3
Development	Student struggles to write code 0	Student can write code the requirement stated 1	Student can write exceptional code with his own ideas 2