Dinesh Pamu

B2- 30

ASSIGNMENT NO: 6

Title:

Bully and Ring algorithm for leader election

Problem Statement:

Implement Bully and Ring algorithm for leader election.

Software and tools:

1. Programming language
2. Communication protocol
3. Distributed system libraries
4. Operating system
5. Code editor
6. Algorithm Implementation

Theory:

Algorithm Implementation: To implement the Bully and Ring algorithm for leader election, the following steps can be followed:

Bully Algorithm:

1. Each process is assigned a unique ID and the highest ID process is selected as the initial leader.
2. When a process detects that the current leader has failed, it starts an election by sending an election message to all the processes with a higher ID than itself.
3. If no response is received within a specified timeout period, the process declares itself the new leader and sends a victory message to all the processes.
4. If a response is received from a higher ID process, the process waits for a victory message from the new leader.
5. When a victory message is received, the process updates its leader and continues normal operations.

Ring Algorithm:

1. Each process is assigned a unique ID and the processes are arranged in a ring topology.
2. When a process wants to initiate an election, it sends an election message to its right-hand neighbor.
3. The receiving process compares the sender's ID with its own and forwards the message to its right-hand neighbor if the sender's ID is higher.
4. If a process receives its own message back, it declares itself the new leader and sends a victory message to all the processes.
5. If a process receives a message from a higher ID process, it forwards the message to its right-hand neighbor.
6. When a victory message is received, the process updates its leader and continues normal operations.

Demonstration:

To demonstrate the effectiveness of the implemented Bully and Ring algorithm for leader election, we will perform the following steps:

1. Create multiple processes with unique IDs.
2. Simulate the failure of the current leader.
3. Observe the election process and verify that the process with the highest ID becomes the new leader.
4. Verify that the new leader is able to coordinate the activities of other processes.

Conclusion:

Implementing Bully and Ring algorithm for leader election is an important concept in distributed systems. By implementing these algorithms, students will have a clear understanding of leader election in distributed systems and how it can be achieved using the Bully and Ring algorithm. This lab practical will help students develop their skills in distributed systems and learn about the challenges and solutions associated with leader election.

Code:

Server Side code:

```
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import java.rmi.server.ServerNotActiveException;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Server extends UnicastRemoteObject implements ServerInterface {
  private static final long serialVersionUID = 1L;
  List<ServerInterface> servers;
  boolean isCoordinator;

  protected Server() throws RemoteException {
    super();
    servers = new ArrayList<>();
  }

  public void addServer(ServerInterface server) throws RemoteException {
    servers.add(server);
  }

  public void removeServer(ServerInterface server) throws RemoteException {
    servers.remove(server);
  }

  public void election() throws RemoteException {
    System.out.println("Starting Election");
```

```
    int myId = this.getId();
    int maxId = myId;
    for (ServerInterface server : servers) {
        if (server.getId() > maxId) {
            maxId = server.getId();
        }
    }
    if (maxId == myId) {
        System.out.println("I am the coordinator");
        isCoordinator = true;
        for (ServerInterface server : servers) {
            if (server.getId() != myId) {
                server.coordinator(myId);
            }
        }
    } else {
        System.out.println("Sending Election Message");
        for (ServerInterface server : servers) {
            if (server.getId() > myId) {
                server.election();
            }
        }
    }
}

public void coordinator(int id) throws RemoteException {
    System.out.println("Coordinator is Server " + id);
    isCoordinator = false;
}

public int getId() throws RemoteException {
    try {
                return Integer.parseInt(getClientHost().substring(12));
        } catch (NumberFormatException | ServerNotActiveException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
        return 0;
}

public static void main(String[] args) throws RemoteException {
    Scanner scanner = new Scanner(System.in);
    Server server = new Server();
    System.out.print("Enter Server Id: ");
    int id = scanner.nextInt();
    System.out.println("Server " + id + " started");
    java.rmi.registry.LocateRegistry.createRegistry(1099);
    try {
                java.rmi.Naming.rebind("rmi://localhost/server" + id, server);
        } catch (MalformedURLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
```

```
        }
      while (true) {
        if (server.isCoordinator) {
          System.out.println("Press enter to send a message to all servers");
          scanner.nextLine();
          for (ServerInterface s : server.servers) {
            s.receiveMessage("Hello from Server " + id);
          }
        } else {
          System.out.println("Waiting for coordinator to send message");
          scanner.nextLine();
        }
      }
  }

  public void receiveMessage(String message) throws RemoteException {
      System.out.println("Message Received: " + message);
  }

@Override
public boolean isCoordinator() throws RemoteException {
        // TODO Auto-generated method stub
        return false;
}
}
```

Interface file:

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface BankAccount extends Remote {

        void deposit (int amount) throws RemoteException;
        void withdraw (int amount) throws RemoteException;
        double getBalance() throws RemoteException;

}
```

Client-side code:

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.util.Scanner;

public class Client {
  public static void main(String[] args) throws RemoteException {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter Server Id: ");
      int id = scanner.nextInt();
      try {
```

```
        ServerInterface server = (ServerInterface) Naming.lookup("rmi://localhost/server" + id);
        server.addServer(server);
        server.election();
        while (true) {
            if (server.isCoordinator()) {
                System.out.println("Press enter to send a message to all servers");
                scanner.nextLine();
                server.receiveMessage("Hello from Server " + id);
            } else {
                System.out.println("Waiting for coordinator to send message");
                scanner.nextLine();
            }
        }
    }
    catch (Exception e) {
        System.out.println("Exception occurred: ");
    }
  }
}
```

Output: