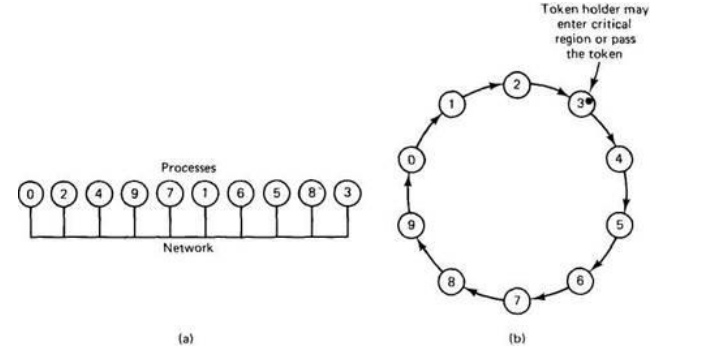
| **Name of Student:** Ajay Karthikesan | | | |
| --- | --- | --- | --- |
| **Roll Number:** 57 | | **Practical Number:** 8 | |
| **Aim of Practical:**  Implementation of mutual exclusion using the token ring algorithm | | | |
| **DOP:** 27.10.23 | | **DOS:** 30.10.23 | |
| **CO Mapped:** CO3 | **PO Mapped:** - | **Faculty Signature:** | **Marks:** |

**Aim: Implementation of mutual exclusion using the Token ring algorithm.**

**THEORY:**

**Token Ring Algorithm:**

Token Ring algorithm achieves mutual exclusion in a distributed system by creating a bus network of processes. A logical ring is constructed with these processes and each assigned a position in the ring. Each process knows who is next in line after itself. When the process ring is initialized, process 0 is given a token. The token circulates around the ring. When a process acquires the token from its neighbor, it checks to see if it is attempting to enter a critical region. If so, the process enters the region, does all the work it needs to, and leaves the region. After it has exited, it passes the token to the next process in the ring. It is not allowed to enter the critical region again using the same token. If a process is handed the token by its neighbor and is not interested in entering a critical region, it just passes the token along to the next process.



➢ **Advantages:**

• The correctness of this algorithm is evident. Only one process has the token at any instant, so only one process can be in a CS

• Since the token circulates among processes in a well-defined order, starvation cannot occur.

➢ **Disadvantages**

• Once a process decides it wants to enter a CS, at worst it will have to wait for every other process to enter and leave one critical region.

• If the token is ever lost, it must be regenerated. In fact, detecting that it is lost is difficult, since the amount of time between successive appearances of the token on the network is not constant. The fact that the token has not been spotted for an hour does not mean that it has been lost; some process may still be using it.

• The algorithm also runs into trouble if a process crashes, but recovery is easier than in the other cases. If we require a process receiving the token to acknowledge receipt, a dead process will be detected when its neighbour tries to give it the token and fails. At that point the dead process can be removed from the group, and the token holder can pass the token to the next member down the line.

**SOURCE CODE:**

**TokenServer.java:**

/\*\*

\* @author Ajay Karthikesan

\*/

import java.net.\*;

import java.io.\*;

class TokenServer {

public static DatagramSocket ds;

public static DatagramPacket dp;

public static void main(String[] args) throws Exception {

try {

ds = new DatagramSocket(1000);

} catch (Exception e) {

e.printStackTrace();

}

while (true) {

byte buff[] = new byte[1024];

ds.receive(dp = new DatagramPacket(buff, buff.length));

String str = new String(dp.getData(), 0, dp.getLength());

System.out.println("Message from " + str);

}

}

}

**TokenClient1.java:**

/\*\*

\* @author Ajay Karthikesan

\*/

import java.net.\*;

import java.io.\*;

class TokenClient1 {

public static DatagramSocket ds;

public static DatagramPacket dp;

public static BufferedReader br;

static int cp = 100;

public static void main(String[] args) throws Exception {

boolean hasToken;

try {

ds = new DatagramSocket(100);

} catch (Exception e) {

e.printStackTrace();

}

hasToken = true;

while (true) {

if (hasToken == true) {

System.out.println("Do you want to enter data...(yes/no):");

br = new BufferedReader(new InputStreamReader(System.in));

String ans = br.readLine();

if (ans.equalsIgnoreCase("yes")) {

System.out.println("ready to send");

System.out.println("sending");

System.out.println("Enter the data");

br = new BufferedReader(new InputStreamReader(System.in));

String str = "Client-1===> " + br.readLine();

byte buff[] = new byte[1024];

buff = str.getBytes();

ds.send(new DatagramPacket(buff, buff.length, InetAddress.getLocalHost(), 1000));

System.out.println("now sending");

} else if (ans.equalsIgnoreCase("no")) {

System.out.println("I am busy state");

//sending msg to client-2

String msg = "Token";

byte bf1[] = new byte[1024];

bf1 = msg.getBytes();

ds.send(new DatagramPacket(bf1, bf1.length, InetAddress.getLocalHost(), 200));

hasToken = false;

//recevingmsg from client-2

byte bf2[] = new byte[1024];

ds.receive(dp = new DatagramPacket(bf2, bf2.length));

String clientmsg = new String(dp.getData(), 0, dp.getLength());

System.out.println("The data is " + clientmsg);

if (clientmsg.equals("Token")) {

hasToken = true;

}

System.out.println("I am leaving busy state");

}

} else {

System.out.println("Entering in receive mode.");

byte bf[] = new byte[1024];

ds.receive(dp = new DatagramPacket(bf, bf.length));

String clientmsg1 = new String(dp.getData(), 0, dp.getLength());

System.out.println("The data is " + clientmsg1);

if (clientmsg1.equals("Token"));

{

hasToken = true;

}

}

}

}

}

**TokenClient2.java:**

/\*\*

\* @author Ajay Karthikesan

\*/

import java.net.\*;

import java.io.\*;

class TokenClient2 {

static DatagramSocket ds;

static DatagramPacket dp;

static BufferedReader br;

public static void main(String[] args) throws Exception {

try {

ds = new DatagramSocket(200);

} catch (Exception e) {

e.printStackTrace();

}

boolean hasToken = true;

while (true) {

//System.out.println("Entering if");

if (hasToken == true) {

System.out.println("Do you want to enter data(Yes/No):");

br = new BufferedReader(new InputStreamReader(System.in));

String str = br.readLine();

if (str.equalsIgnoreCase("yes")) {

System.out.println("Enter Data; ");

br = new BufferedReader(new InputStreamReader(System.in));

String msg = "Client-2===>" + br.readLine();

byte bf1[] = new byte[1024];

bf1 = msg.getBytes();

ds.send(new DatagramPacket(bf1, bf1.length, InetAddress.getLocalHost(), 1000));

System.out.println("Data sent");

} else {

//send to client 1.

String clientmsg = "Token";

byte bf2[] = new byte[1024];

bf2 = clientmsg.getBytes();

ds.send(new DatagramPacket(bf2, bf2.length, InetAddress.getLocalHost(), 100));

hasToken = false;

}

} else {

try {

byte buff[] = new byte[1024];

System.out.println("Entering in receiving mode.");

ds.receive(dp = new DatagramPacket(buff, buff.length));

String clientmsg1 = new String(dp.getData(), 0, dp.getLength());

System.out.println("The data is " + clientmsg1);

if (clientmsg1.equals("Token")) {

hasToken = true;

}

} catch (Exception e) {

e.printStackTrace();

}

}

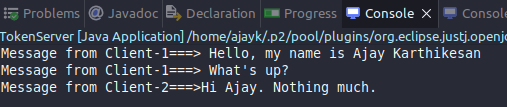
}

}

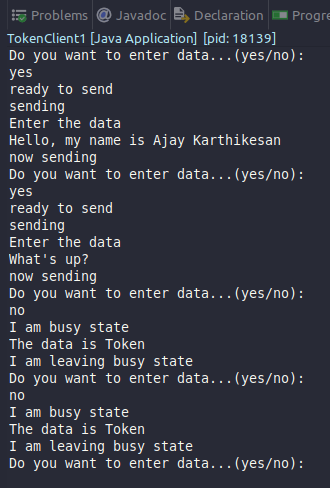
}

**OUTPUT:**

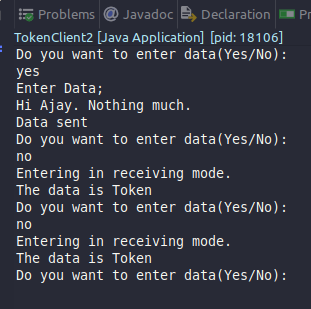
**Server:**

****

**Client1:**

****

**Client2:**

****

**CONCLUSION:**

Thus, we successfully implemented Token ring mutual exclusion.