ASSIGNMENT NO. 5

Problem Statement:

To implement a token ring based mutual exclusion algorithm.

Tools / Environment:

Java Programming Environment, JDK 1.8 or C++ Programming Environment

Related Theory:

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 process is assigned a position in the ring as shown in Fig. The ring positions may be allocated in numerical order of network addresses or some other means. It does not matter what the ordering is. Each process knows who is next in line after itself.

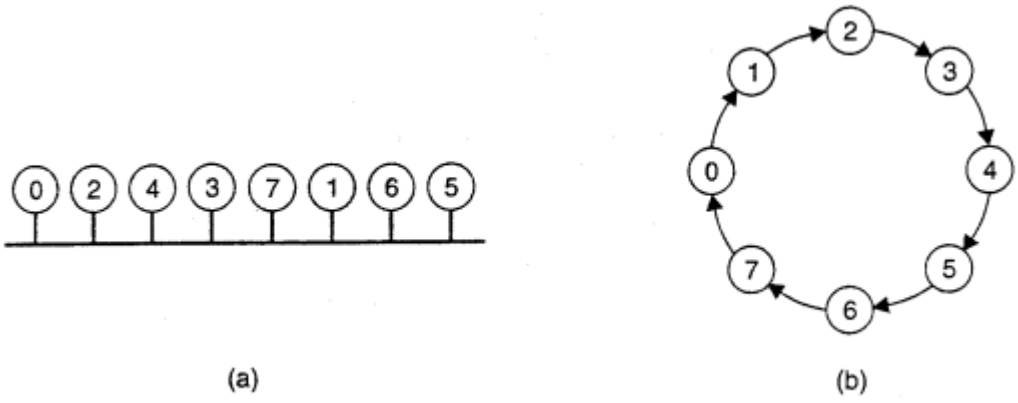


Fig. (a) An unordered group of processes on a network. (b) A logical ring constructed

When the ring is initialized, process 0 is given a token. The token circulates around the ring. It is passed from process k to process k +1 (modulo the ring size) in point-to-point messages. When a process acquires the token from its neighbor, it checks to see if it needs to access the shared resource. If so, the process goes ahead, does all the work it needs to, and releases the resources. After it has finished, it passes the token along the ring. It is not permitted to immediately enter the resource again using the same token.

If a process is handed the token by its neighbor and is not interested in the resource, it just passes the token along. As a consequence, when no processes need the resource, the token just circulates at high speed around the ring. The correctness of this algorithm is easy to see. Only one process has the token at any instant, so only one process can actually get to the resource. Since the token circulates among the processes in a well-defined order, starvation cannot occur.

Once a process decides it wants to have access to the resource, at worst it will have to wait for every other process to use the resource.

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 unbounded. The fact that the token has not been spotted for an hour does not mean that it has been lost; somebody 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 neighbor 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 throw the token over the head of the dead process to the next member down the line, or the one after that, if necessary. Of course, doing so requires that everyone maintain the current ring configuration.

- 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 neighbor 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

A brief comparison of the four mutual exclusion algorithms is given below.

Algorithm	**Messages per Entry/Exit**	**Delay before Entry**	**Problems**
Centralised	3	2	Coordinator Crash
Decentralised	3mk, k=1,2,....	2 m	Starvation, Low efficiency
Distributed	2 (n-1)	2 (n-1)	Crash of any process
Token Ring	1 to ∞	0 to n-1	Lost token, Process Crash

Table: A comparison of three mutual exclusion algorithms.

With the token ring algorithm, the number of messages is variable. If every process constantly wants to enter a critical region. then each token pass will result in one entry and exit, for an average of one message per critical region entered. At the other extreme, the token may sometimes circulate for hours without anyone being interested in it. In this case, the number of messages per entry into a critical region is unbounded.

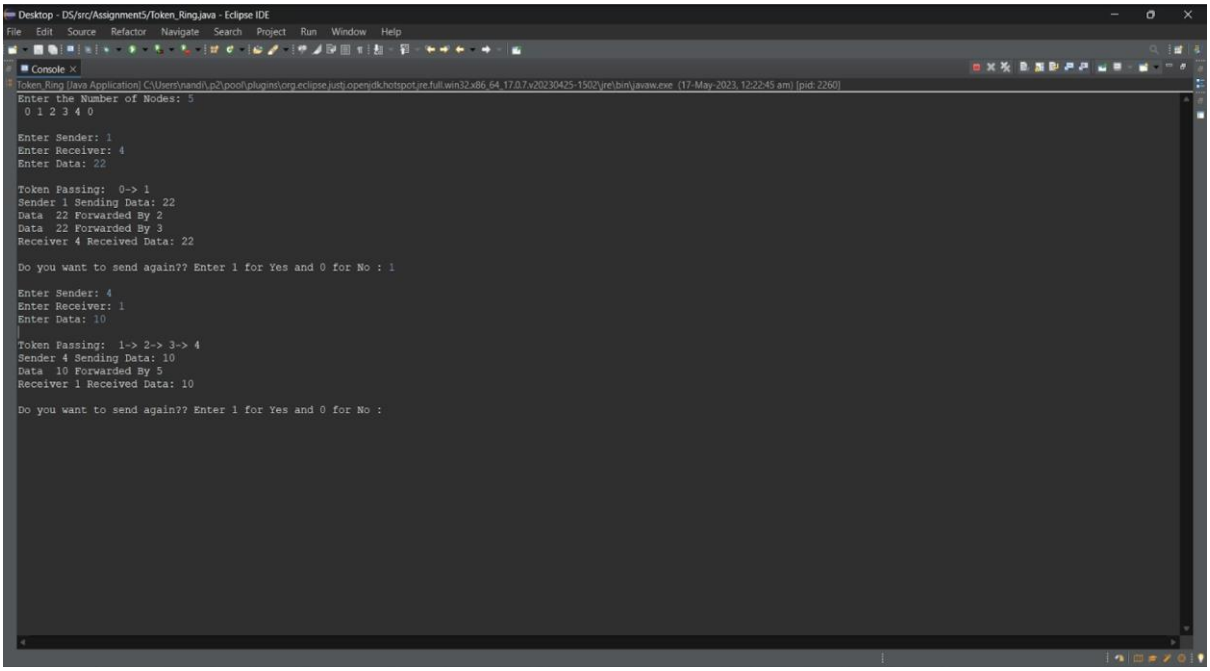
Finally, all algorithms except the decentralized one suffer badly in the event of crashes. Special measures and additional complexity must be introduced to avoid having a crash bring down the entire system.

Code:

```
package Assignment_5;

import java.io.*;
import java.util.*;

class Token_Ring {

    public static void main(String args[]) throws Throwable {
        Scanner scan = new Scanner(System.in);
        System.out.print("Enter the Number of Nodes: ");
        int n = scan.nextInt();
        int m = n - 1;
        // Decides the number of nodes forming the ring
        int token = 0;
        int ch = 0, flag = 0;
        for (int i = 0; i < n; i++) {
            System.out.print(" " + i);
        }
        System.out.println(" " + 0);
        System.out.println(" ");
        do{
            System.out.print("Enter Sender: ");
            int s = scan.nextInt();
            System.out.print("Enter Receiver: ");
            int r = scan.nextInt();
            System.out.print("Enter Data: ");
            int a = scan.nextInt();
            System.out.println(" ");
            System.out.print("Token Passing: ");
            for (int i = token, j = token; (i % n) != s; i++, j = (j + 1) % n) {
                System.out.print(" " + j + "->");
            }
            System.out.println(" " + s);
            System.out.println("Sender " + s + " Sending Data: " + a);
            for (int i = s + 1; i != r; i = (i + 1) % n) {
                System.out.println("Data  " + a + " Forwarded By " + i);
            }
```

```
            System.out.println("Receiver " + r + " Received Data: " + a +"\n");
            token = s;
            do{
               try {
                  if( flag == 1)
            System.out.print("Invalid Input!!...");
                  System.out.print("Do you want to send again?? Enter 1 for Yes and 0 for No : ");
                  ch = scan.nextInt();
                  System.out.println(" ");
                  if( ch != 1 && ch != 0 )
            flag = 1;
                  else
            flag = 0;
               } catch (InputMismatchException e){
                  System.out.println("Invalid Input");
               }
            }while( ch != 1 && ch != 0 );
         }while( ch == 1 );
      }
}
```

Output:



Conclusion:

Token Ring algorithm achieves mutual exclusion in a distributed system by creating a bus network of processes. We successfully implemented Token-Ring Mutual Exclusion.

Submitted By: Shreya Mahajan

Roll No.: 4345

Class: B.E.I.T.

Staff Name: Ms. M. A. Rane / Mr. K. V. Patil

Staff Signature:

Date: