ASSIGNMENT NO.: 05

Problem Statement:
Implement Token ring based mutual exclusion algorithm.

Tools / Environment:
Java Programming Environment, JDK 1.8.

Related Theory:

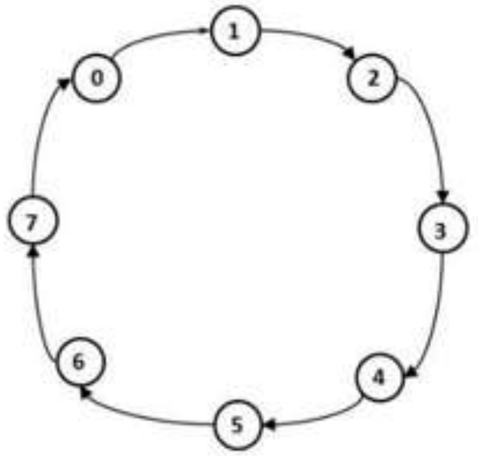


Figure 1: Token ring Algorithm

- In this algorithm it is assumed that all the processes in the system are organized in a logical ring. The figure blow describes the structure.
- The ring positions may be allocated in numerical order of network addresses and is unidirectional in the sense that all messages are passed only in clockwise or anti-clockwise direction.
- When a process sends a request message to current coordinator and does not receive a reply within a fixed timeout, it assumes the coordinator has crashed. It then initializes the ring and process Pi is given a token.
- The token circulates around the ring. It is passed from process k to k+1 in point to point messages. When a process acquires the token from its neighbour it checks to see if it is attempting to enter a critical region. If so the process enters the region does all the execution and leaves the region. After it has exited it passes the token along the ring. It is not permitted to enter a second critical region using the same token.

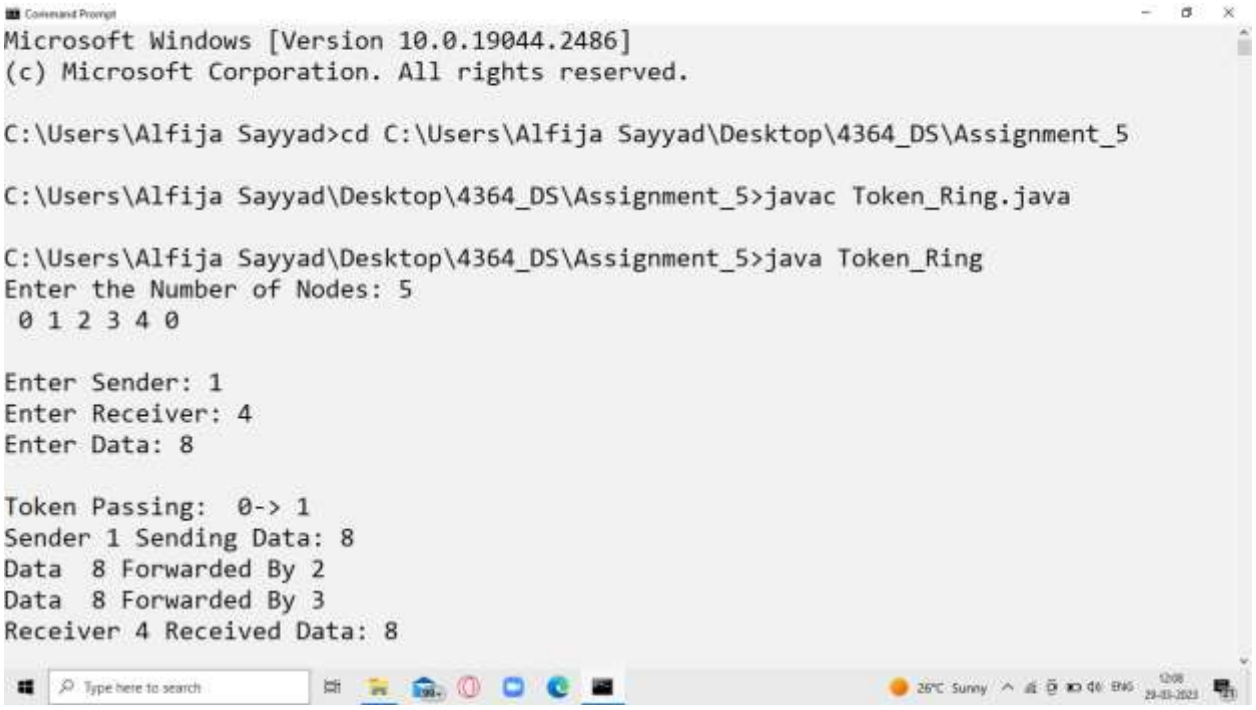
- If a process is handed the token by its neighbour and is not interested in entering a critical region it just passes along. When no processes want to enter any critical regions the token just circulates at high speed around the ring.
- Only one process has the token at any instant so only one process can actually be in a critical region. Since the token circulates among the process in a well-defined order, starvation cannot occur.
- Once a process decides it wants to enter a critical region, at worst it will have to wait for every other process to enter and leave one critical region.
- The disadvantage is that if the token is lost it must be regenerated. But the detection of lost token is difficult. If the token is not received for a long time it might not be lost but is in use.

Implementation:

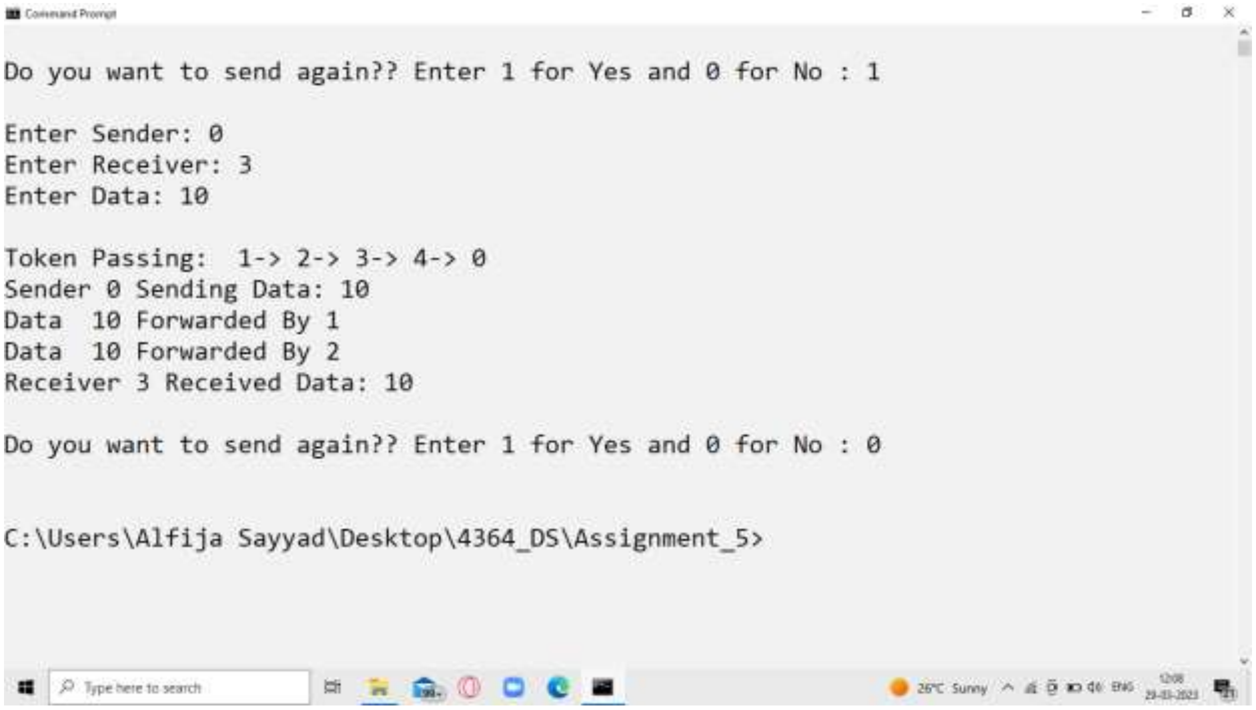
Token_Ring.java:

```
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:



```
Microsoft Windows [Version 10.0.19044.2486]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Alfija Sayyad>cd C:\Users\Alfija Sayyad\Desktop\4364_DS\Assignment_5

C:\Users\Alfija Sayyad\Desktop\4364_DS\Assignment_5>javac Token_Ring.java

C:\Users\Alfija Sayyad\Desktop\4364_DS\Assignment_5>java Token_Ring
Enter the Number of Nodes: 5
 0 1 2 3 4 0

Enter Sender: 1
Enter Receiver: 4
Enter Data: 8

Token Passing:  0-> 1
Sender 1 Sending Data: 8
Data  8 Forwarded By 2
Data  8 Forwarded By 3
Receiver 4 Received Data: 8
```



```

Do you want to send again?? Enter 1 for Yes and 0 for No : 1

Enter Sender: 0
Enter Receiver: 3
Enter Data: 10

Token Passing:  1-> 2-> 3-> 4-> 0
Sender 0 Sending Data: 10
Data  10 Forwarded By 1
Data  10 Forwarded By 2
Receiver 3 Received Data: 10

Do you want to send again?? Enter 1 for Yes and 0 for No : 0


C:\Users\Alfija Sayyad\Desktop\4364_DS\Assignment_5>
```

Conclusion:

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. Each process knows who is next in line after itself. When the 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.

Submitted By: Sayyad Alfija Faruk

Roll No.: 4364

Class: B.E.I.T.

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

Staff Signature:

Date: