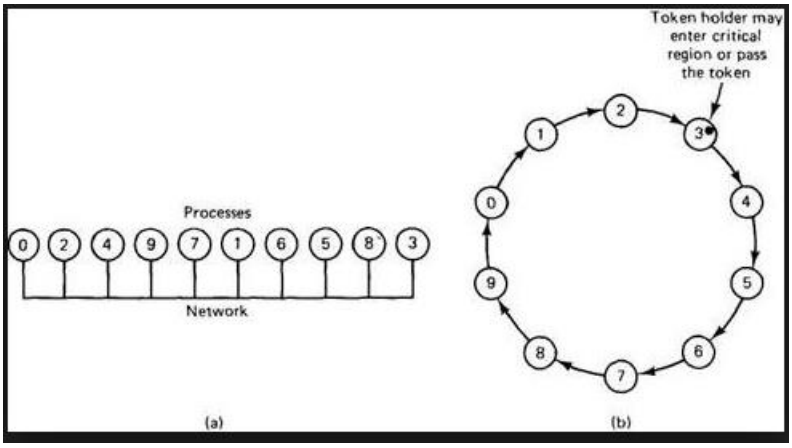
Program – 3

AIM: Implement Mutual Exclusion using Token Ring Algorithm

Introduction and 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. 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.



- 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 a 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

Program – 3

Code

Client

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <arpa/inet.h>
#include<unistd.h>
typedef struct resources
{
    int A;
    char B;
    int C;
    char D;
}resources;
void CriticalSection()
{
    resources R;
    FILE *f;
    f = fopen("shared_mem.txt", "r");
    fread(&R, sizeof(R), 1, f);
    fclose(f);
    printf("Read %d, %d, %d, %d, from memory\n", R.A, R.B, R.C,
R.D);
    printf("Working on data\n");
    R.A += 1;
    R.B += 1;
    R.C += 1;
    R.D += 1;
    f = fopen("shared_mem.txt", "w");
    fwrite(&R, sizeof(R), 1, f);
    fclose(f);
}
int Connect(int P)
{
    int sockid;
    int op_val;
    struct sockaddr_in serv_add;
    if ((sockid = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("Socket Failed\n");
        exit(EXIT_FAILURE);
    }
    setsockopt(sockid, SOL_SOCKET, SO_REUSEADDR, (const void
*)&op_val, sizeof(int));
    memset(&serv_add, 0, sizeof(serv_add));
    serv_add.sin_family = AF_INET;
    serv_add.sin_addr.s_addr = INADDR_ANY;
    serv_add.sin_port = htons(P);

```

Program – 3

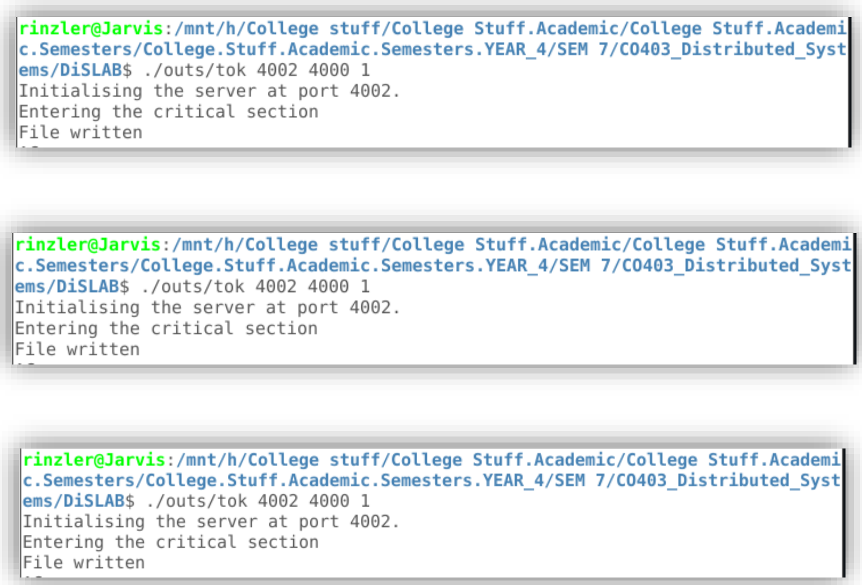
```
54      if (bind(sockid, (const struct sockaddr *)&serv_add,
55  sizeof(serv_add)) < 0)
56      {
57          perror("Bind Error");
58          exit(EXIT_FAILURE);
59      }
60      return sockid;
61  }
62  int main(int argc, char const *argv[])
63  {
64      int Add, Dest, Own;
65      Add = atoi(argv[1]);
66      Dest = atoi(argv[2]);
67      Own = atoi(argv[3]);
68      printf("My address : %d Next Node 2:  %d Permission 3:
69  %d\n",Add, Dest, Own );
70      printf("Making a node at my address = %d\n", Add);
71      int sock_id = Connect(Add);
72      struct sockaddr_in next_node, prev_node;
73      int len, n;
74      char resp[1024];
75      char buff[1024];
76      memset(&next_node, 0, sizeof(next_node));
77      next_node.sin_family = AF_INET;
78      next_node.sin_addr.s_addr = INADDR_ANY;
79      next_node.sin_port = htons(Dest);
80
81      if (Own)
82      {
83          printf("Entering Critical Section\n");
84          CriticalSection();
85          strcpy(resp, "ACK");
86          int c = sendto(sock_id, (const char *)resp, strlen(resp),
87  MSG_CONFIRM,
88                         (const struct sockaddr *) &next_node,
89  sizeof(next_node));
90          memset(&prev_node, 0, sizeof(prev_node));
91          int n = recvfrom(sock_id, (char *)buff, 1024, MSG_WAITALL,
92  (struct sockaddr *) &prev_node, &len);
93          buff[n] = '\0';
94
95          if (strcmp(buff, "ACK"))
96          {
97              strcpy(resp, "TERM");
98              int c = sendto(sock_id, (const char *)resp,
99  strlen(resp), MSG_CONFIRM,
100                             (const struct sockaddr *) &next_node,
101 sizeof(next_node));
102             printf("sent to %d DONE, process exit\n", c);
103         }
104         else
105         {
106             printf("Error message\n");
107         }
108         exit(0);
109     }
110     else
```

Program – 3

```
    {
        while(1)
        {
            memset(&prev_node, 0, sizeof(prev_node));
            int n = recvfrom(sock_id, (char *)buff, 1024,
MSG_WAITALL, ( struct sockaddr *) &prev_node, &len);
            buff[n] = '\0';
            if (!(strcmp(buff, "ACK")))
            {
                CriticalSection();
                sendto(sock_id, (const char *)buff, strlen(buff),
MSG_CONFIRM, (const struct sockaddr *) &next_node,
sizeof(next_node));
            }
            else if (!(strcmp(buff, "TERM")))
            {
                sendto(sock_id, (const char *)buff, strlen(buff),
MSG_CONFIRM, (const struct sockaddr *) &next_node,
sizeof(next_node));
                printf("Exit\n");
                exit(0);
            }
            else
            {
                printf("Invalid message\n");
            }
        }
    }


    return 0;
}
```

Results and Outputs:


rinzler@Jarvis:/mnt/h/College stuff/College Stuff.Academic/College Stuff.Academic.Semesters/College.Stuff.Academic.Semesters.YEAR_4/SEM 7/C0403_Distributed_Systems/DiSLAB$ ./outs/tok 4002 4000 1
Initialising the server at port 4002.
Entering the critical section
File written

rinzler@Jarvis:/mnt/h/College stuff/College Stuff.Academic/College Stuff.Academic.Semesters/College.Stuff.Academic.Semesters.YEAR_4/SEM 7/C0403_Distributed_Systems/DiSLAB$ ./outs/tok 4002 4000 1
Initialising the server at port 4002.
Entering the critical section
File written

rinzler@Jarvis:/mnt/h/College stuff/College Stuff.Academic/College Stuff.Academic.Semesters/College.Stuff.Academic.Semesters.YEAR_4/SEM 7/C0403_Distributed_Systems/DiSLAB$ ./outs/tok 4002 4000 1
Initialising the server at port 4002.
Entering the critical section
File written

Figure 1 Controller

Program – 3

Findings and Learnings:

1. We successfully implemented Token-Ring Mutual Exclusion.
2. This avoids Starvation
3. Lost Key is a major issue.