Experiment 6

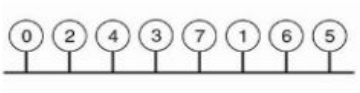
AIM: Write a program to implement Mutual Exclusion to share files using token based algorithm.

THEORY:

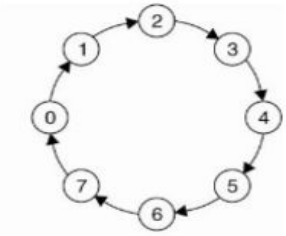
For this algorithm, we assume that there is a group of processes with no inherent ordering of processes, but that some ordering can be imposed on the group. For example, we can identify each process by its machine address and process ID to obtain an ordering. Using this imposed ordering, a logical ring is constructed in software. Each process is assigned a position in the ring and each process must know who is next to it in the ring. Here is how the algorithm works:

1. The ring is initialized by giving a token to process 0. The token circulates around the ring: process n passes it to process (n+1) mod ring size.
2. When a process acquires the token, it checks to see if it is waiting to use the resource. If so, it uses it and does its work. On exit, it passes the token to its neighbouring process.
3. If a process is not interested in grabbing the lock on the resource, it simply passes the token along to its neighbour.

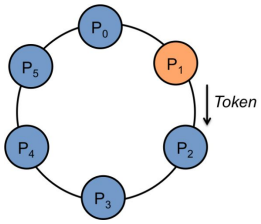
Only one process has the token, and hence the lock on the resource, at a time. Therefore, mutual exclusion is guaranteed. Order is also well-defined, so starvation cannot occur. The biggest drawback of this algorithm is that if a token is lost, it will have to be generated. Determining that a token is lost can be difficult.



Unordered group of processes on a network



A logical ring constructed in software



ALGORITHM

1. Processes are arranged in a logical ring.
2. A token is passed from process to process around the ring.
3. A process must obtain the token before entering the critical section; it passes the token to its neighbor when it exits the critical section.
4. A process passes the token to its neighbor if it does not require to enter the critical section.

SOURCE CODE:

```
#include <stdio.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
void CS(int MY_PORT)
{
 time_t s = time(NULL);
 struct tm *current_time;
 printf("Entering the critical section at: ");
 current_time = localtime(&s);
 printf("%02d:%02d:%02d\n", current_time->tm_hour,
current_time->tm_min,current_time->tm_sec);
 sleep(5);
 FILE *fp;
 fp = fopen("token_based.txt", "a");
 fprintf(fp, "Written by process %d\n", MY_PORT);
 fprintf(fp,"At time %02d:%02d:%02d\n", current_time->tm_hour,
current_time->tm_min,current_time->tm_sec);
 fclose(fp);
}
int makeconnection(int PORT)
{
 int sockfd;
 struct sockaddr_in servaddr;
 if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
 {perror("socket creation failed");
 exit(EXIT_FAILURE);
 }
 int optval = 1;
 setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (const void *)&optval,sizeof(int));
 memset(&servaddr, 0, sizeof(servaddr));
 servaddr.sin_family = AF_INET;
```

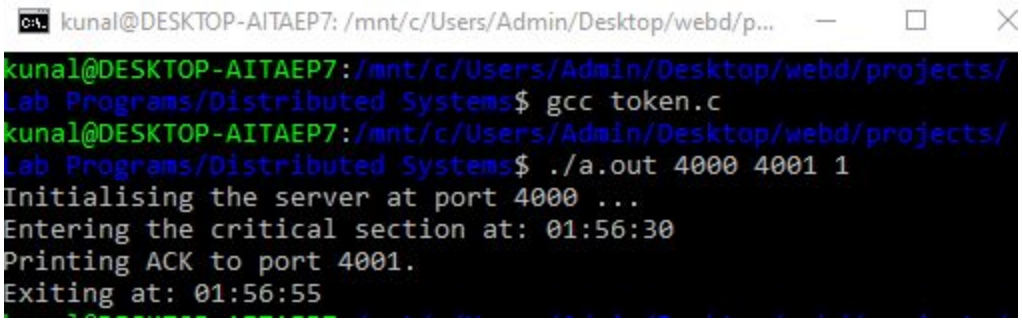
```
 servaddr.sin_addr.s_addr = inet_addr("127.0.0.1"); // loopback address
 servaddr.sin_port = htons(PORT);
 if (bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
 {perror("bind failed");
 exit(EXIT_FAILURE);
 }
 return sockfd;
}
int main(int argc, char *argv[])
{
 int len, n;
 char buffer[1024];
 int MY_PORT = atoi(argv[1]);
 int NEXT_CLIENT_PORT = atoi(argv[2]);
 int IS_INITIATOR = atoi(argv[3]);
 printf("Initialising the server at port %d ...\n", MY_PORT);
 int sockfd = makeconnection(MY_PORT);
 struct sockaddr_in next_client_addr, prev_client_addr;
 char response[1024];
 memset(&next_client_addr, 0, sizeof(next_client_addr));
 next_client_addr.sin_family = AF_INET;
 next_client_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
 next_client_addr.sin_port = htons(NEXT_CLIENT_PORT);
 if (IS_INITIATOR)
 {
 CS(MY_PORT);
 strcpy(response, "ACK");
 printf("Printing %s to port %d. \n", response, NEXT_CLIENT_PORT);
 int check = sendto(sockfd, (const char *)response, strlen(response),
MSG_CONFIRM, (const struct sockaddr *)&next_client_addr,sizeof(next_client_addr));
 memset(&prev_client_addr, 0, sizeof(prev_client_addr));
 n = recvfrom(sockfd, (char *)buffer, 1024, MSG_WAITALL, (struct sockaddr
*)&prev_client_addr, &len);
 buffer[n] = '\0';
 if (!strcmp(buffer, "ACK"))
 {
 strcpy(response, "TERM");
 sendto(sockfd, (const char *)response, strlen(response),
MSG_CONFIRM, (const struct sockaddr *)&next_client_addr,sizeof(next_client_addr));
```

```
 time_t s = time(NULL);
 struct tm *current_time;
 printf("Exiting at: ");
 current_time = localtime(&s);
 printf("%02d:%02d:%02d\n", current_time->tm_hour, current_time->tm_min, current_time->tm_sec);
 }
 else
 { printf("Invalid message received\n");}
 exit(0);
 }
 else
 { while (1)
 {
 memset(&prev_client_addr, 0, sizeof(prev_client_addr));
 printf("Ready to Listen \n");
 n = recvfrom(sockfd, (char *)buffer, 1024, MSG_WAITALL, (struct sockaddr
*)&prev_client_addr, &len);
 buffer[n] = '\0';
 if (!strcmp(buffer, "ACK"))
 {
 CS(MY_PORT);
printf("Printing %s to port %d. \n", buffer,NEXT_CLIENT_PORT);
 sendto(sockfd, (const char *)buffer, strlen(buffer),MSG_CONFIRM, (const struct sockaddr
*)&next_client_addr,sizeof(next_client_addr));
 }
 else if (!strcmp(buffer, "TERM"))
 {
 printf("Printing %s to port %d. \n", buffer,NEXT_CLIENT_PORT);
 sendto(sockfd, (const char *)buffer, strlen(buffer),MSG_CONFIRM, (const struct sockaddr
*)&next_client_addr,sizeof(next_client_addr));
 time_t s = time(NULL);
 struct tm *current_time;
 printf("Exiting at: ");
current_time = localtime(&s);
 printf("%02d:%02d:%02d\n", current_time->tm_hour,current_time->tm_min,
current_time->tm_sec);
 exit(0);
 }
 else
```

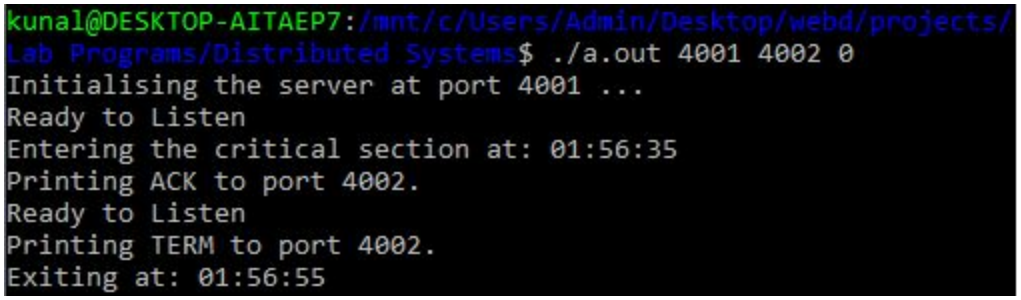
```
 {printf("Invalid message received\n");}
        }
    }
}
```

OUTPUT:

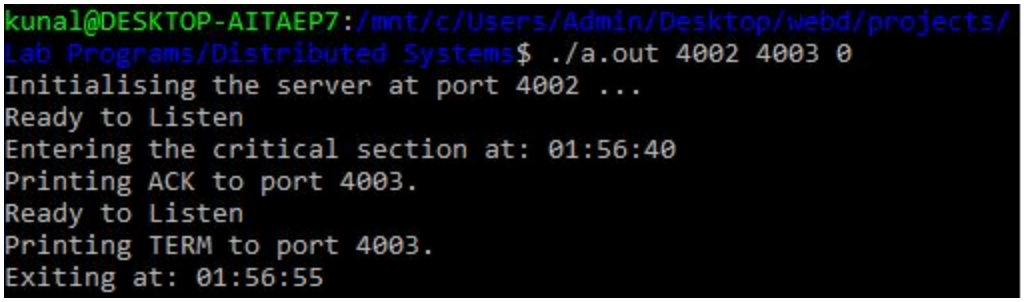
Node 0



Node 1



Node 2



Node 3

```
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/webd/projects/
Lab Programs/Distributed Systems$ ./a.out 4003 4004 0
Initialising the server at port 4003 ...
Ready to Listen
Entering the critical section at: 01:56:45
Printing ACK to port 4004.
Ready to Listen
Printing TERM to port 4004.
Exiting at: 01:56:55
```

Node 4

```
kunal@DESKTOP-AITAEP7:/mnt/c/Users/Admin/Desktop/webd/projects/
Lab Programs/Distributed Systems$ ./a.out 4004 4000 0
Initialising the server at port 4004 ...
Ready to Listen
Entering the critical section at: 01:56:50
Printing ACK to port 4000.
Ready to Listen
Printing TERM to port 4000.
Exiting at: 01:56:55
```

Output file

token_based.txt - Notepad

File Edit Format View Help

```
Written by process 4000
At time 01:56:30
Written by process 4001
At time 01:56:35
Written by process 4002
At time 01:56:40
Written by process 4003
At time 01:56:45
Written by process 4004
At time 01:56:50
```

LEARNING OUTCOMES:

The above program demonstrates how mutual exclusion using a token based algorithm is implemented in C. This algorithm is easy to implement and verify. But it has certain drawbacks too: (i) Long synchronization delay - need to wait for up to (N-1) messages, for N processors since the token passes in one direction through the ring, and (ii) Very unreliable - as any process failure breaks the ring.