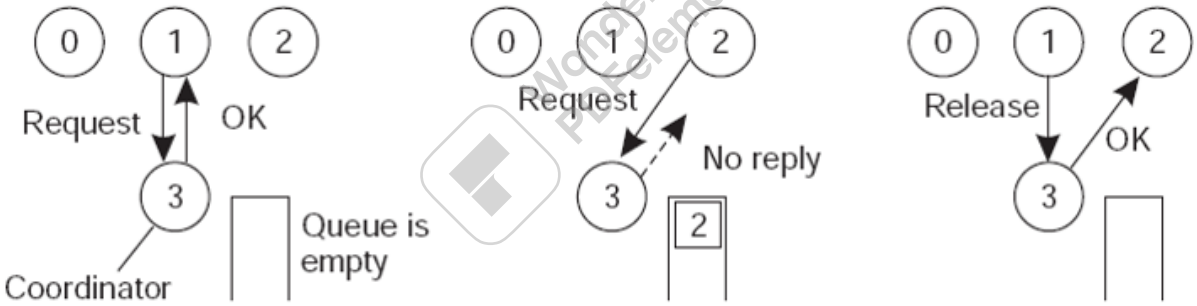
Program – 3

AIM: Implement Mutual Exclusion using Centralized Algorithm.

Introduction and Theory

In centralized algorithm one process is elected as the coordinator which may be the machine. Whenever a process wants to enter a critical region, it sends a request message to the coordinator stating which critical region it wants to enter and asking for permission. If no other process is currently in that critical region, the coordinator sends back a reply granting permission). When the reply arrives, the requesting process enters the critical region. When another process asks for permission to enter the same critical region. Now the coordinator knows that a different process is already in the critical region, so it cannot grant permission. The coordinator just refrains from replying, thus blocking process 2, which is waiting for a reply or it could send a reply 'permission denied.' When process 1 exits the critical region, it sends a message to the coordinator releasing its exclusive access. The coordinator takes the first item off the queue of deferred requests and sends that process a grant message. If the process was still blocked it unblocks and enters the critical region. If an explicit message has already been sent denying permission, the process will have to poll for incoming traffic or block later. When it sees the grant, it can enter the critical region



- Advantages
  - Algorithm guarantees mutual exclusion by letting one process at a time into each critical region.
  - It is also fair as requests are granted in the order in which they are received.
  - No process ever waits forever so no starvation.
  - Easy to implement so it requires only three messages per use of a critical region (request, grant, release).
  - Used for more general resource allocation rather than just managing critical regions.
- Disadvantages
  - The coordinator is a single point of failure, the entire system may go down if it crashes.
  - If processes normally block after making a request, they cannot distinguish a dead coordinator from ''permission denied'' since no message comes back.
  - In a large system a single coordinator can become a performance bottleneck.

Program –3

Code

Client

```c
#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;
int main()
{
        struct sockaddr_in sa;  // Socket address data structure
    resources R;
        int n, sockfd; // read and source
        char buff[1025], obuff[256]; // buffer to store the read
stream
        int snded, rec;

        sockfd = socket(PF_INET, SOCK_STREAM, 0); // New socket
created
          // Checking for valid socket
        if (sockfd < 0)
        {
                printf("Error in creation\n");
                exit(0);
        }
        else
                printf("Socket created\n");

        // Clearing and assigning type and address to the socket
        bzero(&sa, sizeof(sa));
        sa.sin_family = AF_INET;
        sa.sin_port = htons(8888);

        // establishing and verifying the connection
        if (connect(sockfd, (struct sockaddr_in*)&sa, sizeof(sa)) <
0)
        {
                printf("Connection failed\n");
                exit(0);
        }
        else
                printf("Connection made\n");

```

Program – 3

```
54          while (1)
55          {
56                  snded = write(sockfd, "PING", 5);
57                  if (snded > -1)
58                          printf("SENT PING\n");
59                  rec = read(sockfd, obuff, 256);
60                  obuff[rec] = '\0';
61                  if (strcmp(obuff, "PONG") == 0)
62          {
63              usleep(750);
64              FILE *f;
65              f = fopen("shared_mem.txt", "r");
66              fread(&R, sizeof(R), 1, f);
67              fclose(f);
68              printf("read %d, %d, %d, %d from server\n", R.A, R.B,
69  R.C, R.D );
70              R.A += 1;
71              R.B += 1;
72              R.C += 1;
73              R.D += 1;
74              f = fopen("shared_mem.txt", "w");
75              fwrite(&R, sizeof(R), 1, f);
76              fclose(f);
77              printf("Got access to CS\n");
78                          snded = write(sockfd, "DONE", 4);
79              printf("Freeing Lock\n");
80              break;
81          }
82          }
83          // Reading and priting data from the server after
84  verification
85          close(sockfd); // Closing the socket
86          return 0;
```


Controller

```
1  #include <sys/socket.h>
2  #include <netinet/in.h>
3  #include <arpa/inet.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <unistd.h>
7  #include <errno.h>
8  #include <string.h>
9  #include <sys/types.h>
10 #include <time.h>
11 #define TRUE 1
12 #define FALSE 0
13 typedef struct resources{
14     int A;
15     int B;
16     int C;
17     int D;
18 }resources;
19 int main(){
20     resources R, temp;
```

Program – 3

```
21      R.A = 1;
22      R.B = 2;
23      R.C = 3;
24      R.D = 4;
25      FILE *fle;
26      fle = fopen("shared_mem.txt", "w");
27      fwrite(&R, sizeof(R), 1, fle);
28      fclose(fle);
29          struct sockaddr_in sa; // Socket address data structure
30          int opt = TRUE, addrlen;
31          int sockfd, clients[50]; // Source and destination addresses
32          char buff[256]; // Buffer to hold the out-going stream
33          int rec, i, sd, activity, new_sock, sended;
34          int max_sd;
35      int flag = 0;
36          sockfd = socket(AF_INET, SOCK_STREAM, 0); // New socket
37  created
38
39                              // Checking for valid socket
40          memset(clients, 0, sizeof(clients));
41
42          fd_set readfds;
43          if (sockfd < 0)
44          {
45                  printf("Error in creating socket\n");
46                  exit(0);
47          }
48          else
49          {
50                  printf("Socket Created\n");
51          }
52          if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (char
53  *)&opt, sizeof(opt)) < 0)
54          {
55                  printf("error\n");
56          }
57          // Clearing and assigning type and address to the socket
58          printf("Socket created\n");
59          bzero(&sa, sizeof(sa));
60          sa.sin_family = AF_INET;
61          sa.sin_port = htons(8888);
62          sa.sin_addr.s_addr = htonl(INADDR_ANY);
63
64          // binding and verifying the socket to address
65          if (bind(sockfd, (struct sockaddr*)&sa, sizeof(sa))<0)
66          {
67                  printf("Bind Error\n");
68          }
69          else
70                  printf("Binded\n");
71          // starts the server with a max client queue size set as 10
72          listen(sockfd, 10);
73          addrlen = sizeof(sa);
74          // server run
75          while (TRUE)
76          {
77                  // Clearing socket set
```

Program – 3

```
78                  FD_ZERO(&readfds);
79
80                  FD_SET(sockfd, &readfds);
81                  max_sd = sockfd;
82                  for (i = 0; i < 50; i++){
83                          sd = clients[i];
84                          if (sd > 0)
85                                  FD_SET(sd, &readfds);
86                          if (sd > max_sd)
87                                  max_sd = sd;
88                  }
89                  activity = select(max_sd + 1, &readfds, NULL, NULL,
90  NULL);
91                  if (activity < 0)
92                          printf("Select error\n");
93                  if (FD_ISSET(sockfd, &readfds))
94                  {
95                          if ((new_sock = accept(sockfd, (struct
96  sockaddr*)NULL, NULL)) < 0)
97                                  perror("accept");
98                          else
99                          {
100                                 printf("New connection, sock fd
101 %d\n", new_sock);
102                         }
103                         sended = send(new_sock, buff, strlen(buff),
104 0);
105                         if (sended < 0)
106                                 perror("Send");
107                         for (i = 0; i < 50; i++){
108                                 if (clients[i] == 0){
109                                         clients[i] = new_sock;
110                                         break;
111                                 }
112                         }
113                 }
114                 for (i = 0; i < 50; i++){
115                         sd = clients[i];
116                         if (FD_ISSET(sd, &readfds))
117                         {
118                  FILE *fle;
119                  fle = fopen("shared_mem.txt", "r");
120                  fread(&temp, sizeof(temp), 1, fle);
121                  fclose(fle);
122                                 rec = read(sd, buff, 256);
123                                 if (rec == 0){
124                                         getpeername(sd, (struct
125 sockaddr*)&sa, \
126                                                         (socklen_t*)&sa);
127                                         printf("%d has disconnected
128 unexpectedly with ip %s and port %d\n", sd, inet_ntoa(sa.sin_addr),
129 ntohs(sa.sin_port));
130                     printf("recovering data\n");
131                     FILE *fle;
132                     fle = fopen("shared_mem.txt", "w+");
133                     fwrite(&temp, sizeof(temp), 1, fle);
134                     fclose(fle);
```

Program – 3

```
                                        close(sd);
                                        clients[i] = 0;
                                }
                                else
                                {
                                        buff[rec] = '\0';
                                        printf("recieved %s from
%d\n", buff, sd);

                        if (strcmp(buff, "PING") == 0 && flag == 1)
                        {
                            printf("Read buffer = %s, from %d and send
NACK\n", buff, sd);
                            sended = write(sd, "NACK", 4);
                        }

                        else if (strcmp(buff, "PING") == 0 && flag ==
0){
                            printf("Read Buffer = %s, from %d\n", buff,
sd);
                            flag = 1;
                            sended = write(sd, "PONG", 4);
                            // rec = read(sd, buff, 4);
                            // if (rec > 0 && strcmp(buff, "DONE") == 0)
                            // {
                            //     printf("Lock freed\n");
                            //     flag = 0;
                            //     FILE *fle;
                            //     fle = fopen("shared_mem.txt", "r");
                            //     fread(&temp, sizeof(temp), 1, fle);
                            //     printf("Read %d, %d, %d, %d from
%d\n", temp.A, temp.B, temp.C, temp.D, sd );
                            //     fclose(fle);
                            //     clients[i] = 0;
                            //     close(sd);
                            //     break;
                            // }
                        }
                        else if (strcmp(buff, "DONE") == 0){
                            printf("Lock freed\n");
                                flag = 0;
                                FILE *fle;
                                fle = fopen("shared_mem.txt", "r");
                                fread(&temp, sizeof(temp), 1, fle);
                                printf("Read %d, %d, %d, %d from %d\n",
temp.A, temp.B, temp.C, temp.D, sd );
                                fclose(fle);
                                clients[i] = 0;
                                close(sd);
                                break;
                        }

                                }
                        }
                }
        }
        close(sockfd); // close the socket
```

Program – 3

```
        return 0;
}
```

Results and Outputs:

```
c.Semesters/College.Stuff.Academic.Semesters.YEAR_4/SEM 7/C0403_Distributed_Syst
ems/DiSLAB$ ./outs/l2server
Socket Created
Socket created
Binded
New connection, sock fd 4
recieved PING from 4
Read Buffer = PING, from 4
recieved DONE from 4
Lock freed
Read 2, 3, 4, 5 from 4
New connection, sock fd 4
recieved PING from 4
Read Buffer = PING, from 4
recieved DONE from 4
Lock freed
Read 3, 4, 5, 6 from 4
New connection, sock fd 4
recieved PING from 4
Read Buffer = PING, from 4
recieved DONE from 4
Lock freed
Read 4, 5, 6, 7 from 4
```

Figure 1 Controller

Figure 2 Clients


```
rinzler@Jarvis:/mnt/h/College stuff/College Stuff.Academic/College Stuff.Academi
c.Semesters/College.Stuff.Academic.Semesters.YEAR_4/SEM 7/C0403_Distributed_Syst
ems/DiSLAB$ ./outs/l2client
Socket created
Connection made
SENT PING
read 1, 2, 3, 4 from server
Got access to CS
Freeing Lock
```

```
rinzler@Jarvis:/mnt/h/College stuff/College Stuff.Academic/College Stuff.Academi
c.Semesters/College.Stuff.Academic.Semesters.YEAR_4/SEM 7/C0403_Distributed_Syst
ems/DiSLAB$ ./outs/l2client
Socket created
Connection made
SENT PING
read 1, 2, 3, 4 from server
Got access to CS
Freeing Lock
```

```
rinzler@Jarvis:/mnt/h/College stuff/College Stuff.Academic/College Stuff.Academi
c.Semesters/College.Stuff.Academic.Semesters.YEAR_4/SEM 7/C0403_Distributed_Syst
ems/DiSLAB$ ./outs/l2client
Socket created
Connection made
SENT PING
read 1, 2, 3, 4 from server
Got access to CS
Freeing Lock
```

Program – 3

Findings and Learnings:

1. We successfully implemented Centralized Mutual Exclusion.
2. The system is better at handling crashed processes
3. The Entire system fails for Crashed Clients.