

Purva Gautam

20184014

CS-7B



Motilal Nehru National Institute of Technology Allahabad,
Prayagraj – 211004, UP, INDIA

[Fill up the entries carefully]

Registration Number: 20184014

Group (if any) : ~~CS-7B~~ CS-7B

Name of Student: Purva Gautam

Programme: B.Tech./M.Tech./M.B.A./M.Sc./M.C.A./M.S.W./Ph.D.: B.Tech Semester: 7

Please Tick : End Semester (Odd) Examination Session 2021-22 (Lab/Theory)

Branch/ Specialization : CSE

Subject Code : CS-17201 Subject Name: Distributed Systems Lab

For Examiner's use Only [Marks/ Remarks]					
Q.No	a	b	c	d	Total
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
Marks Obtained					
Maximum Marks					

Date: 23-11-2021

Signature of Examiner

Distributed Systems Lab – Final Practical

Ans 1:

Client.c

```
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#include <arpa/inet.h>

int main(int argc, char *argv[])
{
    int fd = 0;
    char buff[1024];
    if (argc < 3)
    {
        printf("Less no of arguments !!");
        return 0;
    }
    memset(buff, '0', sizeof(buff));
    fd = socket(AF_INET, SOCK_STREAM, 0);
    if (fd < 0)
    {
        perror("Client Error: Socket not created succesfully");
        return 0;
    }
    struct sockaddr_in server;
    memset(&server, '0', sizeof(server));
    server.sin_family = AF_INET;
    server.sin_port = htons(atoi(argv[2]));
    int in = inet_pton(AF_INET, argv[1], &server.sin_addr);
    if (in < 0)
    {
        perror("Client Error: IP not initialized succesfully");
        return 0;
    }
    in = connect(fd, (struct sockaddr *)&server, sizeof(server));
    if (in < 0)
    {
        perror("Client Error: Connection Failed.");
        return 0;
    }
    while (1)
```

```

{
    printf("\nPlease enter the string: ");
    bzero(buff, 256);
    fgets(buff, 255, stdin);
    printf("\nSending string to SERVER: %s ", buff); /* Send message to the server */
    in = send(fd, buff, strlen(buff), 0);
    if (in < 0)
    {
        perror("\nClient Error: Writing to Server");
        return 0;
    }
    bzero(buff, 256);

    in = recv(fd, buff, 255, 0);
    if (in < 0)
    {
        perror("\nClient Error: Reading from Server");
        return 0;
    }
    printf("\nReceived FROM SERVER: %s ", buff);
}
printf("BYE!\n");
close(fd);
return 0;
}

```

Server.c

```

#include <arpa/inet.h>
#include <ctype.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

// make all vowels upper case
void upper_vowel_case(char *str)
{
    int i = 0;
    while (str[i] != '\0')
    {
        if (str[i] == 'a' || str[i] == 'e' || str[i] == 'i' || str[i] == 'o' ||
            str[i] == 'u')

```

```

        {
            str[i] = toupper(str[i]);
        }
        i++;
    }
}

int main()
{
    int fd = 0;
    char buff[1024];
    memset(buff, '0', sizeof(buff));
    fd = socket(AF_INET, SOCK_STREAM, 0);
    if (fd < 0)
    {
        perror("Client Error: Socket not created succesfully");
        return 0;
    }
    struct sockaddr_in server;
    memset(&server, '0', sizeof(server));
    server.sin_family = AF_INET;
    server.sin_port = htons(10011);
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    bind(fd, (struct sockaddr *)&server, sizeof(server));
    int in;
    listen(fd, 10);
    while (in = accept(fd, (struct sockaddr *)NULL, NULL))
    {
        int childpid, n;
        if ((childpid = fork()) == 0)
        {
            close(fd);
            bzero(buff, 256);
            while ((n = recv(in, buff, 256, 0)) > 0)
            {
                printf("\nServer Received: %s", buff);
                upper_vowel_case(buff);
                send(in, buff, strlen(buff), 0);
                bzero(buff, 256);
            }
            close(in);
            exit(0);
        }
    }
}

```

```
purgaut@Fiorella15: /mnt/c/users/purva/desktop/distributed lab/final practical
127.0.0.1 11001
Client Error: Connection Failed.: Connection refused
purgaut@Fiorella15: /mnt/c/users/purva/desktop/distributed lab/final practical$ ./CLIENT
127.0.0.1 10011

Please enter the string: hello_my_name_is_purva
Sending string to SERVER: hello_my_name_is_purva
Received FROM SERVER: hEllo_my_nAmE_Is_pUrVa

Please enter the string: i_am_from_mnnit_allahabad
Sending string to SERVER: i_am_from_mnnit_allahabad
Received FROM SERVER: I_Am_frOm_mnnIt_AlIaHAbAd

Please enter the string: my_regno_is_20184014
Sending string to SERVER: my_regno_is_20184014
Received FROM SERVER: my_rEgnO_Is_20184014

Please enter the string:

purgaut@Fiorella15: /mnt/c/users/purva/desktop/distributed lab/final practical$ gcc
server.c
purgaut@Fiorella15: /mnt/c/users/purva/desktop/distributed lab/final practical$ gcc
server.c -o SERVER
purgaut@Fiorella15: /mnt/c/users/purva/desktop/distributed lab/final practical$ ./S
ERVER

Server Received: hello_my_name_is_purva
Server Received: i_am_from_mnnit_allahabad
Server Received: my_regno_is_20184014
```

Ans 2)

Ricart-Agrawala Algorithm code:

```
purgaut@Fiorella15: /mnt/c/users/purva/desktop/distributed lab/final practical
Inside request queue delete..
1 REQUEST 2 16
SITE 1 REQUESTING FOR CS AS WELL..
So tie break..between 1 and 1..!!
Their seqno: 2 My seqno: 1 Their ID: 1 My ID: 1
@@@@ I WIN @@@@
Node 1 get the priority...So put 1 in the defer queue..!
Inside the defer queue insert..
Site 1 receiving REQUEST from site 1 with clock 21.
Receiving REQUEST message from site 1
Inside the request queue insert..
Inserting values in request queue..
1 REQUEST 2 21
Inside request queue delete..
1 REQUEST 2 21
SITE 1 REQUESTING FOR CS AS WELL..
So tie break..between 1 and 1..!!
Their seqno: 2 My seqno: 1 Their ID: 1 My ID: 1
@@@@ I WIN @@@@
Node 1 get the priority...So put 1 in the defer queue..!
Inside the defer queue insert..

purgaut@Fiorella15: /mnt/c/users/purva/desktop/distributed lab/final practical
Inside the process queue insert..
3
4
2
0
1
Inside process queue delete..
3
4
2
0
1
SITE 1 REQUESTING FOR CS..
Clock value is updated to 20
Site 1 sending REQUEST for process 3 to site 2 with timestamp 21
Site 1 has sent the request message..
Clock value is updated to 21
Site 1 sending REQUEST for process 3 to site 3 with timestamp 22
Site 1 has sent the request message..
Clock value is updated to 22
Site 1 sending REQUEST for process 3 to site 4 with timestamp 23
Site 1 has sent the request message..

purgaut@Fiorella15: /mnt/c/users/purva/desktop/distributed lab/final practical
Process thread starting to process requests..
Process thread starting to process requests..
Process thread starting to process requests..
ALL SERVERS ARE READY!!!
Inside process queue delete..
0
1
2
4
3
SITE 3 REQUESTING FOR CS..
Clock value is updated to 1
Site 3 sending REQUEST for process 0 to site 1 with timestamp 2
Site 3 has sent the request message..
Process thread starting to process requests..

purgaut@Fiorella15: /mnt/c/users/purva/desktop/distributed lab/final practical
3 REQUEST 1 10
SITE 1 REQUESTING FOR CS AS WELL..
So tie break..between 4 and 1..!!
Their seqno: 1 My seqno: 1 Their ID: 1 My ID: 4
Site 4 has sent the request message..
Sending REPLY message from site 4 to site 1
3 REQUEST 1 10
Inside request queue delete..
3 REQUEST 1 10
SITE 3 REQUESTING FOR CS AS WELL..
So tie break..between 4 and 3..!!
Their seqno: 1 My seqno: 1 Their ID: 3 My ID: 4
Site 4 has sent the request message..
Sending REPLY message from site 4 to site 3
Site 4 receiving REPLY from site 1 with clock 15.
Receiving REPLY message from site 1
```

```

purgaut@Fiorella15: /mnt/c/users/purva/desktop/distributed lab/final practical
purgaut@Fiorella15: /mnt/c/users/purva/desktop/distributed lab/final practical$ gcc -o raexe RA.c -pthread
collect2: error: ld returned 1 exit status
purgaut@Fiorella15: /mnt/c/users/purva/desktop/distributed lab/final practical$ gcc -o raexe RA.c -pthread
RA.c: In function 'main':
RA.c:250:57: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
    250 |     pthread_create(&proc1[s], NULL, &processes, (void *)s); //Creating pr
        |                                     ^
RA.c: In function 'processes':
RA.c:519:13: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
    519 |     pid = (int)msg;
        |           ^
purgaut@Fiorella15: /mnt/c/users/purva/desktop/distributed lab/final practical$ ./raexe
File opened successfully.
Segmentation fault
purgaut@Fiorella15: /mnt/c/users/purva/desktop/distributed lab/final practical$ ./raexe 1 loca
lhost 5001
File opened successfully.
4 4My ID is : 1 My Port : 5001 and My IP localhost
Configuration File
1 localhost 5001
2 localhost 6001
3 localhost 7001
4 localhost 8001
Inside the process queue insert..
0
Inside the process queue insert..
0
Inside the process queue insert..
1
Inside the process queue insert..
1

purgaut@Fiorella15: /mnt/c/users/purva/desktop/distributed lab/final practical$ ./raexe 3 localhost 7001
-bash: ./raexe.exe: No such file or directory
purgaut@Fiorella15: /mnt/c/users/purva/desktop/distributed lab/final practical$ ./raexe 3 localhost 7001
File opened successfully.
4 4My ID is : 3 My Port : 7001 and My IP localhost
Configuration File
1 localhost 5001
2 localhost 6001
3 localhost 7001
4 localhost 8001
Inside the process queue insert..
0
Inside the process queue insert..
0
Inside the process queue insert..
1
Inside the process queue insert..
1

purgaut@Fiorella15: /mnt/c/users/purva/desktop/distributed lab/final practical$ cd 'distributed lab'/'final practical'
purgaut@Fiorella15: /mnt/c/users/purva/desktop/distributed lab/final practical$ ./raexe 4 localhos
t 8001
File opened successfully.
4 4My ID is : 4 My Port : 8001 and My IP localhost
Configuration File
1 localhost 5001
2 localhost 6001
3 localhost 7001
4 localhost 8001
Inside the process queue insert..
0
Inside the process queue insert..
0
Inside the process queue insert..
1
Inside the process queue insert..
1

```

```

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/times.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <signal.h>
#include <netdb.h>
#include <math.h>
#include <time.h>

#define BACKLOG 150 //Number of pending connections queue will hold
#define MAXDATASIZE 100 //Maximum number of bytes we can get at once
#define MAXLINE 750
#define TRUE 1
#define FALSE 0

```

```

#define nproc 4 //Total number of sites in the system
pthread_t tid1,tid2,tid3;
pthread_t proc1[5];
int argc1;
char argv1[50];
char argv[50];
int i;
int listenPort; //The process port on which it is receiving the messages
int count[25];
int serverFlag = 0; //flag to check if all servers/sites are ready
int requesttime[5]; //times at which the request message is sent
struct host //Structure to maintain the Id, Server name and Port number
{
    int id;
    char name[50];
    int port;
};
struct host hs[20];
typedef struct myinfo1 //Structure to maintain my information
{
    int id;
    int portno;
    char mac[50]; //machine or host name eg. net06
} myinfo;
myinfo my;
struct message //Structure that contains the message exchanged
{
    int id; //site ID
    int procid; //Process ID
    char type[10]; //Type of message sent
    int seq_no; //sequence number of the process
    int clock; //clock at which the message is sent
};
static int rfront=-1,rrrear=-1; //The pointers for REQ_QUEUE
static int dfront=-1,drear=-1; //The pointers for the DEFER_QUEUE
static int pfront=-1,prear=-1; //The pointers for the PROCESS_QUEUE
struct message REQ_QUEUE[200]; //The REQUEST QUEUE
struct message DEFER_QUEUE[200]; //The DEFER QUEUE
int PROCESS_QUEUE[200]; //The PROCESS QUEUE
sem_t proc[5];
sem_t site;
//Mutex variables used to lock various globally shared variables
pthread_mutex_t sequence;
pthread_mutex_t inCS;
pthread_mutex_t reqCS;
pthread_mutex_t ccounter;
pthread_mutex_t replycnt;
pthread_mutex_t signals;

```

```

pthread_mutex_t  types;
pthread_mutex_t  clk;
pthread_mutex_t  sending_mutex;
pthread_mutex_t  sema;
pthread_mutex_t  pqueue;
pthread_mutex_t  processth;
pthread_mutex_t  counts;
pthread_mutex_t  requestq;
pthread_mutex_t  deferq;
pthread_mutex_t  refront;
pthread_mutex_t  rerear;
pthread_mutex_t  defront;
pthread_mutex_t  derear;
//The threads used in this program
void * recv_reply_thread ( void *);
void * recv_request_thread ( void *);
void * process_thread (void *);
void * processes (void *);
void send_reply(struct message *msg); //Function to send reply messages
void rinsert(struct message); //Request queue functions
void rdisplay(void);
struct message rdelete(void);
void dinsert(struct message); //Defer queue functions
void ddisplay(void);
struct message ddelete(void);
void pinsert(int); //Process queue functions
void pdisplay(void);
int pdelete();
void sigchld_handler(int s) // reap all dead processes
{
    while(wait(NULL) > 0);
}
int me; //my id number
int our_seq_number=0; // My sequence number
int outstanding_reply_count = noproc-1; //outstanding reply count..Initially N-1
int counter=0; // counter for clock
int clockvalue=1;
int highest_sequence_number=0;
int counting=0;
int req_CS=0; // Request for the Critical section: initially FALSE
int in_CS=0; //Inside the Critical Section: initially FALSE
int SIGNAL;
int in; //to read if in CS
int req; // to read if req CS
int seqno; // to read seq no
int sendcount;
int recvcount;
int replycount; //to read current outstanding_reply_count

```



```

// SAVE CONNECTION - RECV
void saveconn(int sockfdr, int id, int counter)
{
    int n;
    FILE *file;
    char line[MAXLINE];
    struct message * msg, m;
    pthread_mutex_lock(&sending_mutex);
    msg = (struct message *)malloc(sizeof(struct message));
    n = recv(sockfdr,(void *)msg,sizeof(struct message),0);
    pthread_mutex_unlock(&sending_mutex);
    m = *((struct message *)msg);
    if(n == 0)
        return;
    else if(n < 0)
        printf("saveconn(): read error\n");
    else
        printf("Site %d receiving %s from site %d with clock %d. \n",me,m.type,m.id,m.clock);
        if(highest_sequence_number < m.seq_no)
        {
            highest_sequence_number = m.seq_no;
        }
        else
            highest_sequence_number = highest_sequence_number;
        pthread_mutex_lock(&clk);
        clockvalue++;
        if(clockvalue < (m.clock+1))
        {
            clockvalue = (m.clock+1);
        }
        pthread_mutex_unlock(&clk);
        if(strcmp(m.type,"REQUEST") == 0)
        {
            printf("Recieving REQUEST message from site %d\n",m.id);
            pthread_mutex_lock(&requestq);
            rinsert(m);
            rdisplay();
            pthread_mutex_unlock(&requestq);
            SIGNAL=1; //Process wakeup
        }
        else if(strcmp(m.type,"REPLY") == 0)
        {
            printf("Recieving REPLY message from site %d\n",m.id);
            pthread_mutex_lock(&replycnt);
            replycount++;
            printf("CURRENT REPLYCOUNT : %d\n",replycount);
            pthread_mutex_unlock(&replycnt);
        }
}

```

```

else
{
    printf("Improper message : message not received properly\n");
    rdisplay();
}
}
// CLIENT CONNECTION - SEND
void cliconn(FILE *file,int sockfds, char *mac, int portno, int id,struct message *messg,int counter)
{
    int n,i;
    char sendline[400],recvline[MAXLINE + 1];
    portno = my.portno;
    pthread_mutex_lock(&clk);
    messg->clock = clockvalue;
    pthread_mutex_unlock(&clk);
    if(send(sockfds,messg,sizeof(struct message),0) != sizeof(struct message))
        printf("cliconn(): write error on socket\n");
    printf("Site %d has sent the request message..\n",me);
}
// MAIN FUNCTION
int main(int argc, char **argv)
{
    struct message *msg;
    int s;
    pthread_mutex_init(&sequence,NULL);
    pthread_mutex_init(&inCS,NULL);
    pthread_mutex_init(&reqCS,NULL);
    pthread_mutex_init(&ccounter,NULL);
    pthread_mutex_init(&replycnt,NULL);
    pthread_mutex_init(&signals,NULL);
    pthread_mutex_init(&types,NULL);
    pthread_mutex_init(&clk,NULL);
    pthread_mutex_init(&sending_mutex,NULL);
    pthread_mutex_init(&sema,NULL);
    pthread_mutex_init(&processthld,NULL);
    pthread_mutex_init(&pqueue,NULL);
    pthread_mutex_init(&counts,NULL);
    pthread_mutex_init(&requestq,NULL);
    pthread_mutex_init(&deferq,NULL);
    pthread_mutex_init(&refront,NULL);
    pthread_mutex_init(&rerear,NULL);
    pthread_mutex_init(&defront,NULL);
    pthread_mutex_init(&derear,NULL);
    FILE *file;
    file = fopen("config.txt", "r"); //Open the configuration file
    if(file==NULL)
    {
        printf("Error: can't open file.\n");
    }
}

```

```

        return 1;
    }
    else
        printf("File opened successfully.\n");
    for(i=1;i<=noprocs;i++)
    {
        fscanf(file,"%d",&hs[i].id);//Reading host info from config file
        fscanf(file,"%s",hs[i].name);
        fscanf(file,"%d",&hs[i].port);
    }
    argc1 = argc;
    printf("%d %d",argc1,argc);
    my.id = atoi(argv[1]);
    me = my.id;
    strcpy(my.mac,argv[2]);
    char t[9];
    strcpy(t,argv[3]);
    my.portno = atoi(t);
    listenPort = atoi(t);
    printf("My ID is : %s My Port : %s and My IP %s\n",argv[1],argv[3],argv[2]);
    printf("Configuration File\n"); //Printing the configuration file details
    for(i=1;i<=noprocs;i++)
    {
        printf("%d %s %d\n",hs[i].id,hs[i].name,hs[i].port);
    }
    fclose(file);
    for(s=0;s<5;s++)
    {
        sem_init( &proc[s],0,0);
    }
    for(s=0;s<5;s++)
    {
        pthread_create( &proc1[s], NULL, &processes, (void *)s); //Creating processes in site
    }
    pthread_create( &tid3, NULL, &recv_request_thread, &msg); //Creating send thread
    pthread_create( &tid2, NULL, &recv_reply_thread, &msg); //Creating receive thread
    pthread_create( &tid1, NULL, &process_thread, &msg); //Creating process thread
    pthread_join( tid1, NULL ); //Join all process threads
    pthread_join( tid2, NULL ); //Join all receive reply threads
    pthread_join( tid3, NULL ); //Join all receive request threads
    for(s=0;s<5;s++)
    {
        pthread_join( proc1[s], NULL); //Join all processes in the site
    }
}
// RECIEVE REQUESTS THREAD
void * recv_request_thread(void *msg)
{

```

```

    struct sockaddr_in their_addr; // Connector's address information
    struct hostent *h;
    int sockfd;
    int pid;
    int j;
    int check, procid;
    struct message m;
    struct message tm;
    m = *((struct message *)msg);
    for(j=0;j<noproc; j++)
    {
        count[j];
    }
    if (argc1 != 4) //The command line should have the output file,machine name and
                    //my port address as the runtime parameters
    {
        //Error check the command line
        fprintf(stderr,"usage: getip address\n");
        exit(1);
    }
    int liveServers = 1; //Initialising number of live processes counting for itself
    while(liveServers <= noproc)
        //Checks for number of processes that are alive before sending
        //the messages. It is similar to the initialization message sent
        //to all the proceses
    {
        liveServers = 1;
        int j;
        for(j=1;j<=noproc;j++)
        {
            if ((sockfds = socket(AF_INET, SOCK_STREAM, 0)) == -1)
                //Opens a connection to check for the live processes
            {
                perror("socket");
                exit(1);
            }
            if ((h=gethostbyname(hs[j].name)) == NULL)
            {
                perror("gethostbyname");
                exit(1);
            }
            their_addr.sin_family = AF_INET;
            their_addr.sin_port = htons(hs[j].port);
            their_addr.sin_addr = *((struct in_addr *)h->h_addr);
            memset(&(their_addr.sin_zero), '\0', 8);
            if (connect(sockfds, (struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1)
                //Connects to the process
            {
            }
        }
    }

```

```

        else
        {
            liveServers++;
            //if connection is setup increments liveserver count by one
            //everytime it establishes a connection with a process
        }
        close(sockfds); //Connection closed after checking is done
    }
}
serverFlag = 1; //When all processes are alive sets serverFlag to 1.
printf("\nALL SERVERS ARE READY!!! \n"); //Processes are ready to listen now.
/*REQUESTING ENTRY TO THE CRITICAL SECTION*/
while(1)
{
    if(pfront== -1)
    {
        check = 0;
        break;
    }
    else
        check = 1;
    if(check)
    {
        pthread_mutex_lock(&processthld);
        procid = pdelete();
        printf("SITE %d REQUESTING FOR CS..\n",me);
        pthread_mutex_lock(&reqCS);
        req_CS = 1;
        req = req_CS;
        pthread_mutex_unlock(&reqCS);
        //preparing the structure for sending
        m.id = me;
        pthread_mutex_lock(&types);
        strcpy(m.type,"REQUEST");
        pthread_mutex_unlock(&types);
        pthread_mutex_lock(&sequence);
        our_seq_number = highest_sequence_number+1;
        m.seq_no = our_seq_number;
        pthread_mutex_unlock(&sequence);
        for(i=1; i<=nproc; i++)
        {
            if(i == me) //Checking request not sending to myself
            {
                continue;
            }
            if ((h=gethostbyname(hs[i].name)) == NULL)
            {
                perror("gethostbyname");
            }
        }
    }
}

```

```

        exit(1);
    }
    if ((sockfds = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        //Opens socket to send messages
    {
        perror("socket");
        exit(1);
    }
    their_addr.sin_family = AF_INET;    // Host byte order
    their_addr.sin_port = htons(hs[i].port); // Short, network byte order
    their_addr.sin_addr = *((struct in_addr *)h->h_addr);
    memset(&(their_addr.sin_zero), '\0', 8); // Zero the rest of the struct
    sleep(1);
    if (connect(sockfds, (struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1)
    {
        perror("connect in send thread\n");
        exit(1);
    }
    printf("Clock value is updated to %d\n", clockvalue);
    clockvalue = clockvalue + 1;
    m.clock = clockvalue;
    requesttime[i] = m.clock;
    counter = counter + 1;
    requesttime[i] = m.clock;
    printf("Site %d sending REQUEST for process %d to site %d with timestamp
%d\n", me, procid, i, m.clock);
    cliconn(stdin, sockfds, my.mac, my.portno, my.id, &m, counter);
    close(sockfds);
}
printf("Waiting for reply from other sites...");
while(1)
{
    if (replycount == outstanding_reply_count)
    {
        pthread_mutex_lock(&replycnt);
        replycount = 0;
        pthread_mutex_unlock(&replycnt);
        break;
    }
    else
        sleep(2);
}
/*ENTERING THE CRITICAL SECTION*/
//Is entering inside the CS
pthread_mutex_lock(&inCS);
in_CS = 1;
in = in_CS;
pthread_mutex_unlock(&inCS);

```

```

//Is not requesting for CS again
pthread_mutex_lock(&reqCS);
    req_CS = 0;
    req = req_CS;
pthread_mutex_unlock(&reqCS);
sem_post(&proc[procid]);
sem_wait(&site);
//Entering CS
pthread_mutex_lock(&inCS);
    in_CS = 0;
    in = in_CS;
pthread_mutex_unlock(&inCS);
/*RELEASING THE CRITICAL SECTION*/
sendcount = 0;
// Pop from the defer queue
while(drear!=-1)
{
    pthread_mutex_lock(&types);
        strcpy(m.type,"REPLY"); //copy my node id and the message type
    pthread_mutex_unlock(&types);
    m.id = me;
    pthread_mutex_lock(&deferq);
        tm = ddelete(); //tm is the buffer in which the values are stored in message
        pid = tm.id;
    pthread_mutex_unlock(&deferq);
    sendcount++;
    printf("Send Reply Message count: %d\n",sendcount);
    if ((h=gethostbyname(hs[pid].name)) == NULL)
    {
        perror("gethostbyname");
        exit(1);
    }
    if ((sockfds = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        //Opens socket to send messages
    {
        perror("socket");
        exit(1);
    }
    their_addr.sin_family = AF_INET;    // Host byte order
    their_addr.sin_port = htons(hs[pid].port); // Short, network byte order
    their_addr.sin_addr = *((struct in_addr *)h->h_addr);
    memset(&(their_addr.sin_zero), '\0', 8); // Zero the rest of the struct
    sleep(1);
    if (connect(sockfds, (struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1)
    {
        perror("connect in send thread\n");
        //exit(1);
    }
}

```

```

        cliconn(stdin, sockfds, my.mac, my.portno, my.id, &m, 0);
        close(sockfds);
    }
    pthread_mutex_unlock(&processthld);
}
}
// RECIEVE REPLY THREAD
void * recv_reply_thread(void *msg)
{
    int sockfdr, new_fd;          // Listen on sock_fd, new connection on new_fd
    struct sockaddr_in my_addr;    // My address information
    struct sockaddr_in their_addr; // Connector's address information
    int sin_size;
    int yes=1;
    FILE *file;
    struct message m;
    m = *((struct message *)msg);
    if ((sockfdr = socket(AF_INET, SOCK_STREAM, 0)) == -1) //Opening socket connection
    {
        perror("socket");          // Checking for any in case if connection failed
        exit(1);
    }
    if (setsockopt(sockfdr, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1)
    {
        perror("setsockopt");
        exit(1);
    }
    my_addr.sin_family = AF_INET;    // Host byte order
    my_addr.sin_port = htons(listenPort); // Short, network byteorder
    my_addr.sin_addr.s_addr = INADDR_ANY; // Automatically fill with myIP
    memset(&(my_addr.sin_zero), '\0', 8); // Zero the rest of the struct
    if (bind(sockfdr, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1)
    {
        // Bind to my address
        perror("bind");          // Check for errors
        exit(1);
    }
    if (listen(sockfdr, BACKLOG) == -1) // Listening from the other processes
    {
        perror("listen");          // Checking for errors
        exit(1);
    }
    for(;;)
    {
        int numbytes;
        char buf[MAXDATASIZE];
        sin_size = sizeof(struct sockaddr_in);
        if ((new_fd = accept(sockfdr, (struct sockaddr *)&their_addr, (socklen_t *)&sin_size)) == -1)

```



```

    {
        perror("In server accept");
        continue;
    }
    else
    {
        saveconn(new_fd,my.id,counter);
        close(new_fd);
    }
}
}
// THE SITE CONTROLLER THREAD
void * processes(void *msg)
{
    int pid,mycount;
    pid = (int)msg;
    for(mycount=1; mycount<=20 ; mycount++)
    {
        pthread_mutex_lock(&pqueue);
        pinsert(pid);
        counting++;
        pdisplay();
        pthread_mutex_unlock(&pqueue);
        sem_wait(&proc[pid]);
        //entering crictical section
        printf("Starting CS execution at time : %ld\n",time(NULL));
        printf("*****SITE %d PROCESS %d ENTERING THE CS*****\n",me,pid);
        printf("*****INSIDE THE CS*****\n");
        sleep(1);
        printf("*****SITE %d PROCESS %d EXITING THE CS*****\n",me,pid);
        printf("Exiting CS at time : %ld\n",time(NULL));
        sem_post(&site);
        printf("\nProcess %d is in CS for %d times\n",pid,mycount);
    }
    printf("*** Total Message count: %d ***\n",counting);
}
// THE PROCESSING THREAD
void * process_thread(void *msg)
{
    int nodeseq;
    int pid; //use it for ripping the process to b sent to
    struct message m;
    m = *((struct message *)msg);
    while(1)
    {
        printf("Process thread starting to process requests..\n ");
        sleep(3);
        while(SIGNAL == 1)

```

```

{
pthread_mutex_lock(&rfront);
while(1)
{
    //pop data from the request queue
    if(rfront!=-1)
    {
        rdisplay();
        pthread_mutex_lock(&inCS);
        in = in_CS;
        pthread_mutex_unlock(&inCS);
        pthread_mutex_lock(&reqCS);
        req = req_CS;
        pthread_mutex_unlock(&reqCS);
        pid=REQ_QUEUE[rfront].id;
        nodeseq = REQ_QUEUE[rfront].seq_no;
        pthread_mutex_lock(&requestq);
        m = rdelete();
        pthread_mutex_unlock(&requestq);
        if (in == 1)
        {
            printf("PROCESS ALREADY IN CS..So putting in defer queue.!!!\n");
            pthread_mutex_lock(&deferq);
            dinsert(m);
            pthread_mutex_unlock(&deferq);
        }
        else if( in == 0)
        {
            if (req == 1)
            {
                printf("SITE %d REQUESTING FOR CS AS WELL..\nSo tie break..between %d and
%d..!!\n",pid,me,pid);
                pthread_mutex_lock(&sequence);
                seqno = our_seq_number;
                pthread_mutex_unlock(&sequence);
                printf("Their seqno: %d My seqno: %d Their ID: %d My ID:
%d\n",nodeseq,seqno,pid,me);
                if ((nodeseq < seqno) || (nodeseq == seqno && pid < me))
                {
                    m.id = pid;
                    m.seq_no = 0;
                    strcpy(m.type, "REPLY");
                    send_reply(&m); //send reply to that node with my structure (node id and type)
                }
            }
            else
            {
                printf("@@@@@ I WIN @@@@@ \nNode %d get the priority...So put %d in the
defer queue..!!\n",me,pid);
            }
        }
    }
}
}

```

```

        pthread_mutex_lock(&deferq);
        dinsert(m);
        pthread_mutex_unlock(&deferq);
    }
}
else
{
    m.id = pid;
    m.seq_no = 0;
    strcpy(m.type, "REPLY");
    send_reply(&m); //send reply to that node with my structure (node id and type)
}
}
else
{
    sleep(5);
}
}
}
pthread_mutex_unlock(&refront);
pthread_mutex_lock(&signals);
    SIGNAL = 0;
pthread_mutex_unlock(&signals);
}
}
// SEND REPLY FUNCTION
void send_reply(struct message *msg)
{
    struct sockaddr_in their_addr; // Connector's address information
    struct hostent *h;
    int sockfds;
    int pid;
    struct message m;
    m = *((struct message *)msg);
    pid = m.id;
    pthread_mutex_lock(&types);
    strcpy(m.type, "REPLY");
    pthread_mutex_unlock(&types);
    m.id = me;
    if ((h=gethostbyname(hs[pid].name)) == NULL)
    {
        perror("gethostbyname");
        exit(1);
    }
    if ((sockfds = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        //Opens socket to send messages
    {

```

```

        perror("socket");
        exit(1);
    }
    their_addr.sin_family = AF_INET;    // Host byte order
    their_addr.sin_port = htons(hs[pid].port); // Short, network byte order
    their_addr.sin_addr = *((struct in_addr *)h->h_addr);
    memset(&(their_addr.sin_zero), '\0', 8); // Zero the rest of the struct
    sleep(1);
    if (connect(sockfds, (struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1)
    {
        perror("connect in send thread\n");
        exit(1);
    }
    cliconn(stdin, sockfds, my.mac, my.portno, my.id, &m, 0);
    printf("Sending REPLY message from site %d to site %d\n", me, pid);
    close(sockfds); // Socket closed after sending the message to the process
}
// THE REQUEST QUEUE
//REQUEST QUEUE INSERT
void rinsert(struct message temp)
{
    printf("Inside the request queue insert..\n");
    if(rfront==rrear)
    {
        rfront = 0;
        rrear = 0;
    }
    printf("Inserting values in request queue..\n");
    REQ_QUEUE[rrear].id = temp.id;
    strcpy(REQ_QUEUE[rrear].type, temp.type);
    REQ_QUEUE[rrear].seq_no = temp.seq_no;
    REQ_QUEUE[rrear].clock = temp.clock;
    rrear++;
}
//REQUEST QUEUE DISPLAY
void rdisplay()
{
    int i;
    if(rfront== -1)
        printf("CAUTION: Request Queue is Empty..!!\n");
    for(i=rfront; i<rrear; i++)
        printf("%d %s %d\n", REQ_QUEUE[i].id, REQ_QUEUE[i].type, REQ_QUEUE[i].seq_no, REQ_QUEUE[i].clock);
}
//REQUEST QUEUE DELETE
struct message rdelete()
{
    struct message tempvar;

```

```

printf("Inside request queue delete.. \n");
rdisplay();
if(rfront==-1)
{
    printf("CAUTION: Request Queue Underflow !!\n");
    exit(1);
}
else if(rfront==rrear-1)
{
    tempvar.id = REQ_QUEUE[rfront].id;
    strcpy(tempvar.type,REQ_QUEUE[rfront].type);
    tempvar.seq_no = REQ_QUEUE[rfront].seq_no;
    tempvar.clock = REQ_QUEUE[rfront].clock;
    rfront = -1;
    rrear = -1;
}
else
{
    tempvar.id = REQ_QUEUE[rfront].id;
    strcpy(tempvar.type,REQ_QUEUE[rfront].type);
    tempvar.seq_no = REQ_QUEUE[rrear].seq_no;
    tempvar.clock = REQ_QUEUE[rfront].clock;
    rfront++;
}
return tempvar;
}
// THE DEFER QUEUE
//DEFER QUEUE INSERT
void dinsert(struct message temp)
{
    printf("Inside the defer queue insert..\n");
    if(dfront==drear)
    {
        dfront = 0;
        drear = 0;
    }
    DEFER_QUEUE[drear].id = temp.id;
    strcpy(REQ_QUEUE[drear].type,temp.type);
    DEFER_QUEUE[drear].seq_no = temp.seq_no;
    DEFER_QUEUE[drear].clock = temp.clock;
    drear++;
}
//DEFER QUEUE DISPLAY
void ddisplay()
{
    int i;
    if(dfront==-1)
    printf("Defer Queue is Empty..!!\n");

```

```

    for(i=dfront;i<drear;i++)
        printf("%d %s %d\n",DEFER_QUEUE[i].id,DEFER_QUEUE[i].type,DEFER_QUEUE[i].seq_no,DEFER_QUEUE[i].clock);
    }
//DEFER QUEUE DELETE
struct message ddelete()
{
    struct message tempvar;
    printf("Inside the defer queue delete..\n");
    ddisplay();
    if(dfront== -1)
    {
        printf("CAUTION: Defer queue Underflow !!\n");
        exit(1);
    }
    else if(dfront==drear-1)
    {
        tempvar.id = DEFER_QUEUE[dfont].id;
        strcpy(tempvar.type,DEFER_QUEUE[dfont].type);
        tempvar.seq_no = DEFER_QUEUE[drear].seq_no;
        tempvar.clock = DEFER_QUEUE[dfont].clock;
        dfront = -1;
        drear = -1;
    }
    else
    {
        tempvar.id = DEFER_QUEUE[dfont].id;
        strcpy(tempvar.type, DEFER_QUEUE[dfont].type);
        tempvar.seq_no = DEFER_QUEUE[drear].seq_no;
        tempvar.clock = DEFER_QUEUE[dfont].clock;
        dfront++;
    }
    return tempvar;
}
// THE PROCESS QUEUE
//PROCESS QUEUE INSERT
void pinsert(int temp)
{
    printf("Inside the process queue insert..\n");
    if(pfront==prear)
    {
        pfront = 0;
        prear = 0;
    }
    PROCESS_QUEUE[prear] = temp;
    prear++;
}
//PROCESS QUEUE DISPLAY

```

```

void pdisplay()
{
    int i;
    if(pfront== -1)
        printf("Process Queue is Empty\n");
    for(i=pfront;i<prear;i++)
        printf("%d \n",PROCESS_QUEUE[i]);
}
//PROCESS QUEUE DELETE
int pdelete()
{
    int tempvar;
    printf("Inside process queue delete..\n");
    pdisplay();
    if(pfront== -1)
    {
        printf("CAUTION: Process Queue Underflow !!\n");
        exit(1);
    }
    else if(pfront==prear-1)
    {
        tempvar = PROCESS_QUEUE[pfront];
        pfront = -1;
        prear = -1;
    }
    else
    {
        tempvar = PROCESS_QUEUE[pfront];
        pfront++;
    }
    return tempvar;
}

```

Config.txt

```

1  localhost  5001
2  localhost  6001
3  localhost  7001
4  localhost  8001

```