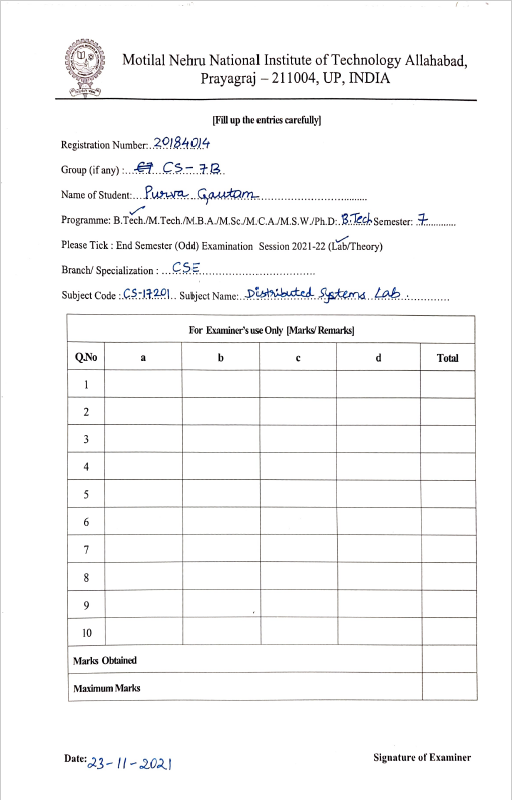
**Purva Gautam**

**20184014**

**CS-7B**



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);

}

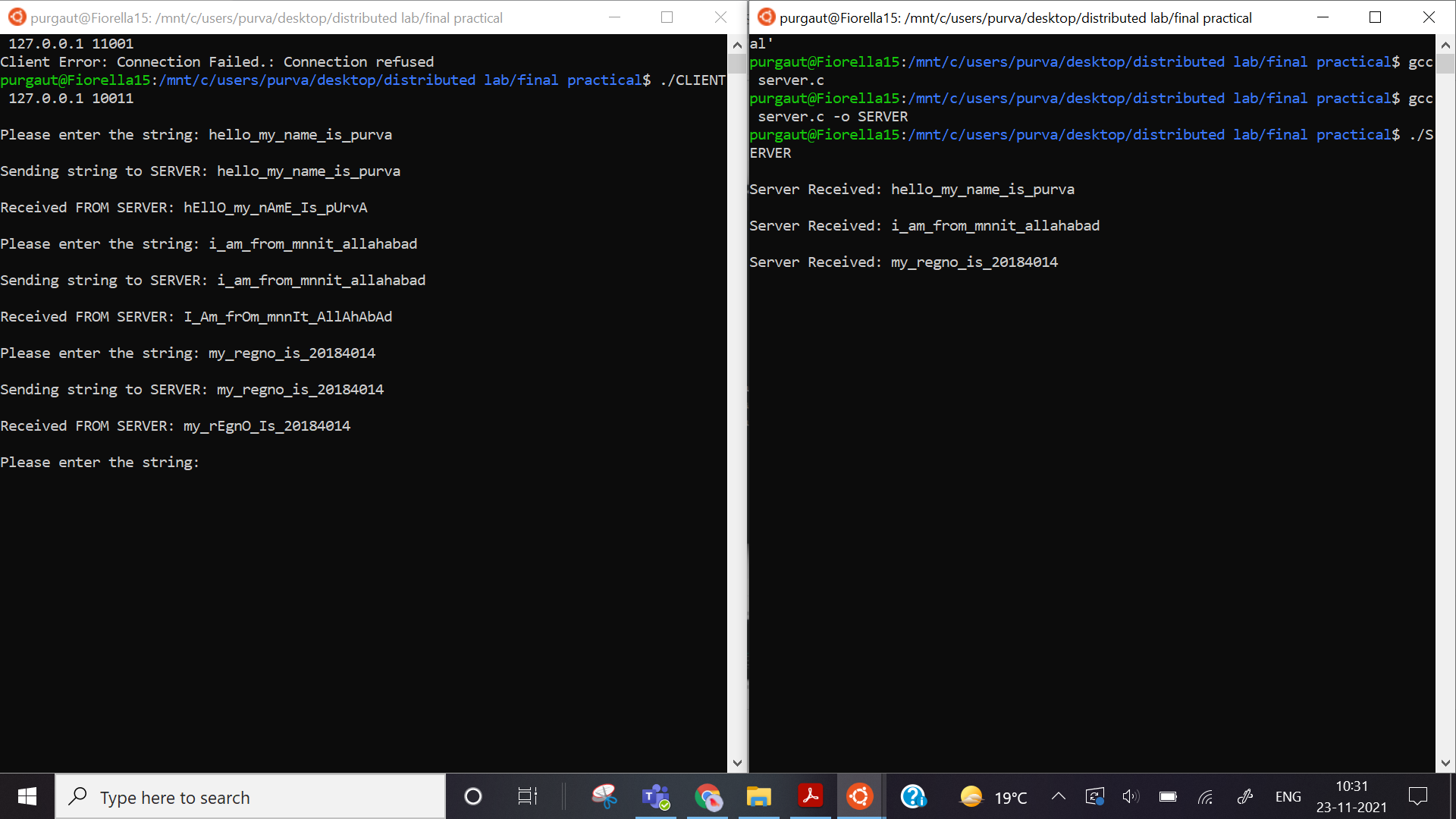
close(in);

exit(0);

}

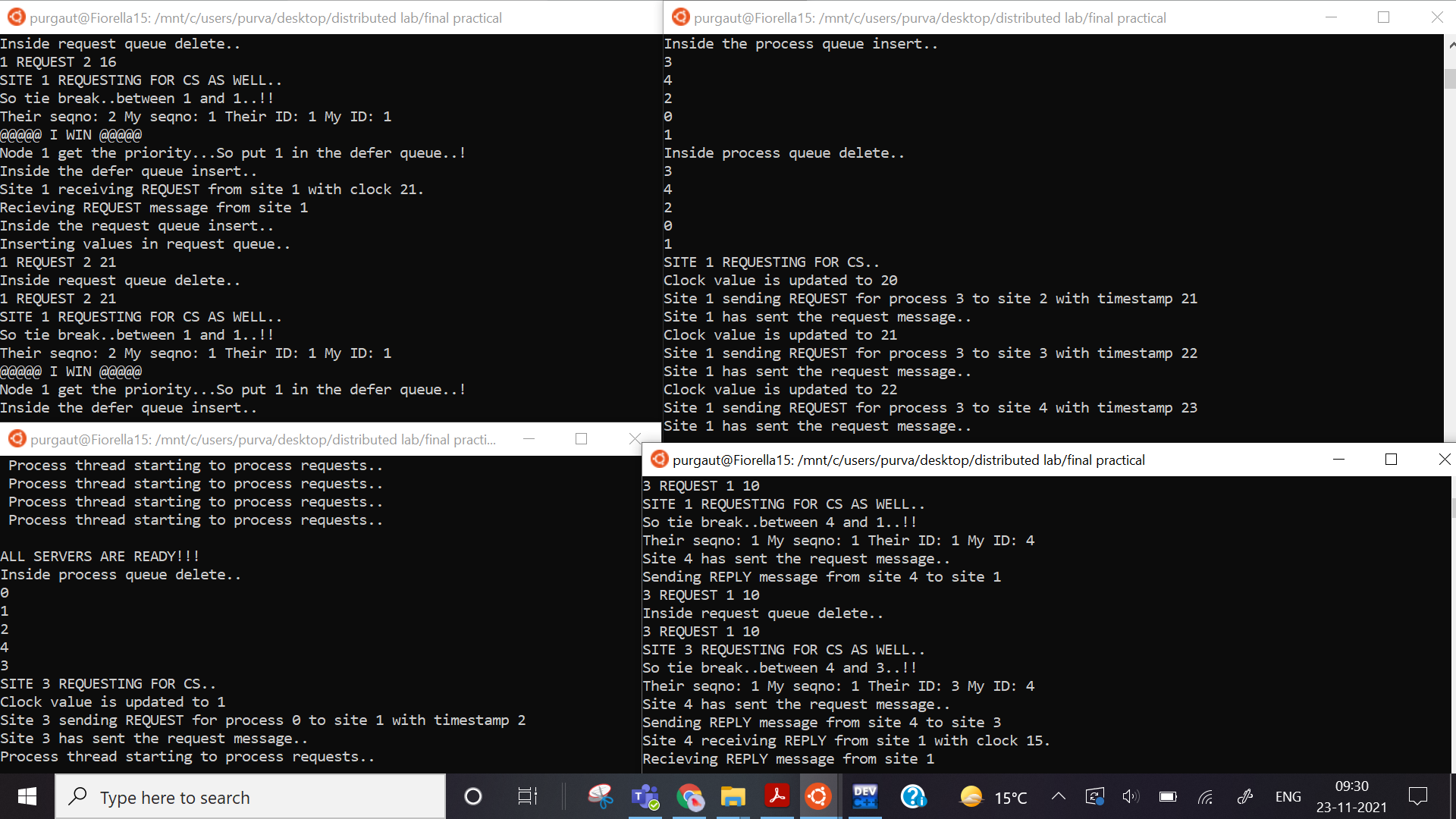
}

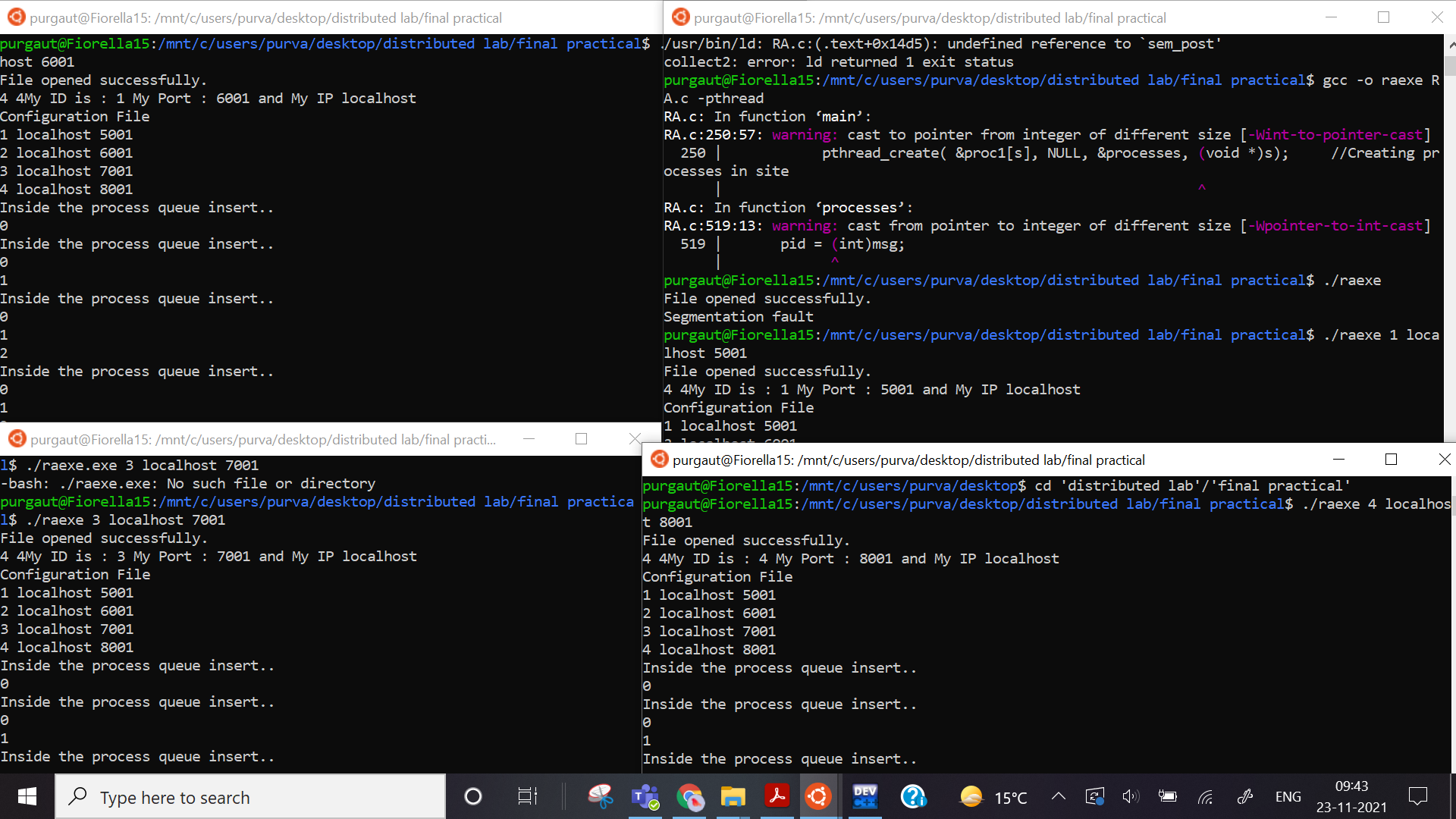
}



**Ans 2)**

**Ricart-Agrawala Algorithm code:**





#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 noproc 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 recieving 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 comtains 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,rrear=-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 varialbes used to lock variuos 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 processthd;

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(&processthd,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<=noproc;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<=noproc;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 recieve 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 recieve reply threads

pthread\_join( tid3, NULL ); //Join all recieve 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 sockfds;

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 extablishes 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(&processthd);

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<=noproc; 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,networbyteorder

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,networbyteorder

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(&processthd);

}

}

}

// RECIEVE REPLYS 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(&refront);

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 REPLYS 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,networbyteorder

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 %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 %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[dfront].id;

strcpy(tempvar.type,DEFER\_QUEUE[dfront].type);

tempvar.seq\_no = DEFER\_QUEUE[drear].seq\_no;

tempvar.clock = DEFER\_QUEUE[dfront].clock;

dfront = -1;

drear = -1;

}

else

{

tempvar.id = DEFER\_QUEUE[dfront].id;

strcpy(tempvar.type, DEFER\_QUEUE[dfront].type);

tempvar.seq\_no = DEFER\_QUEUE[drear].seq\_no;

tempvar.clock = DEFER\_QUEUE[dfront].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