**1.Implement Producer Consumer Problem using stack**

Code:-

#include<stdio.h>

#include<stdlib.h>

int mutex=1,full=0,empty=3,x=0;

int main()

{

int n;

void producer();

void consumer();

int wait(int);

int signal(int);

printf("\n1.Producer\n2.Consumer\n3.Exit");

while(1)

{

printf("\nEnter your choice:");

scanf("%d",&n);

switch(n)

{

case 1: if((mutex==1)&&(empty!=0))

producer();

else

printf("Buffer is full!!");

break;

case 2: if((mutex==1)&&(full!=0))

consumer();

else

printf("Buffer is empty!!");

break;

case 3:

exit(0);

break;

}

}

return 0;

}

int wait(int s)

{

return (--s);

}

int signal(int s)

{

return(++s);

}

void producer()

{

mutex=wait(mutex);

full=signal(full);

empty=wait(empty);

x++;

printf("\nProducer produces the item %d",x);

mutex=signal(mutex);

}

void consumer()

{

mutex=wait(mutex);

full=wait(full);

empty=signal(empty);

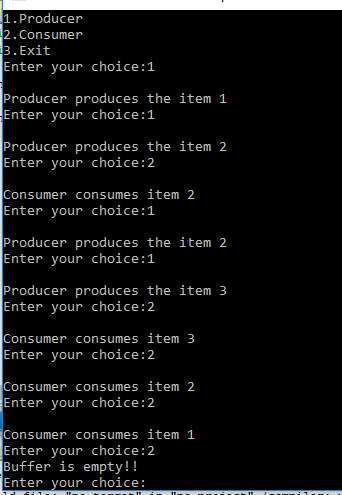
printf("\nConsumer consumes item %d",x);

x--;

mutex=signal(mutex);

}

Output:-



**2. To implement reader writer problem using monitors.**

Code:-

#include<stdio.h>

#include<semaphore.h>

#include<pthread.h>

int rc=0,val,wc=0;

pthread\_mutex\_t mutex1,mread;

pthread\_t tr1,tw1,tr2,tw2,tw3;

pthread\_attr\_t tr1attr,tw1attr,tr2attr,tw2attr,tw3attr;

void \*reader();

void \*writer();

int main()

{

pthread\_mutex\_init(&mutex1,NULL);

pthread\_mutex\_init(&mread,NULL);

pthread\_attr\_init(&tr1attr);

pthread\_attr\_init(&tr2attr);

pthread\_attr\_init(&tw1attr);

pthread\_attr\_init(&tw2attr);

pthread\_attr\_init(&tw3attr);

printf("\n Writer 1 created");

pthread\_create(&tw1,&tw1attr,writer,NULL);

printf("\n Reader 1 created");

pthread\_create(&tr1,&tr1attr,reader,NULL);

printf("\n Writer 2 created");

pthread\_create(&tw2,&tw2attr,writer,NULL);

printf("\n Reader 2 created");

pthread\_create(&tr2,&tr2attr,reader,NULL);

printf("\n Writer 3 created");

pthread\_create(&tw3,&tw3attr,writer,NULL);

pthread\_join(tw1,NULL);

pthread\_join(tr1,NULL);

pthread\_join(tw2,NULL);

pthread\_join(tr2,NULL);

pthread\_join(tw3,NULL);

return 0;

}

void \*writer()

{

pthread\_mutex\_lock(&mutex1);

wc++;

printf("\n Enter data: ");

scanf("%d",&val);

pthread\_mutex\_unlock(&mutex1);

printf("\n writer %d levng",wc);

pthread\_exit(0);

}

void \*reader()

{

pthread\_mutex\_lock(&mread);

rc++;

if(rc==1)

pthread\_mutex\_lock(&mutex1);

pthread\_mutex\_unlock(&mread);

printf("\n Value read by reader %d: %d",rc,val);

pthread\_mutex\_lock(&mread);

rc--;

if(rc==0)

pthread\_mutex\_unlock(&mutex1);

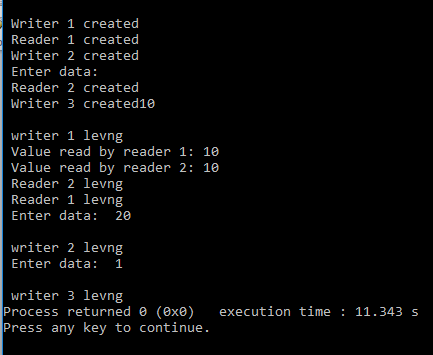
pthread\_mutex\_unlock(&mread);

printf("\n Reader %d levng",rc+1);

pthread\_exit(0);

}

Output:-



**3. To implement Lamport’s Logical Clock.**

Code:-

#include<stdio.h>

#include<conio.h>

int max1(int a, int b) //to find the maximum timestamp between two events

{

if (a>b)

return a;

else

return b;

}

int main()

{

int i,j,k,p1[20],p2[20],e1,e2,dep[20][20];

printf("enter the events : ");

scanf("%d %d",&e1,&e2);

for(i=0;i<e1;i++)

p1[i]=i+1;

for(i=0;i<e2;i++)

p2[i]=i+1;

printf("enter the dependency matrix:\n");

for(i=0;i<e1;i++)

{

for(j=0;j<e2;j++)

scanf("%d",&dep[i][j]);

}

for(i=0;i<e1;i++)

{

for(j=0;j<e2;j++)

{

if(dep[i][j]==1)

{ p2[j]=max1(p2[j],p1[i]+1);

for(k=j;k<e2;k++)

p2[k+1]=p2[k]+1;

}

if(dep[i][j]==-1)

{

p1[i]=max1(p1[i],p2[j]+1);

for(k=i;k<e1;k++)

p2[k+1]=p1[k]+1;

}

}

}

printf("P1 : ");

for(i=0;i<e1;i++)

{

printf("%d ",p1[i]);

}

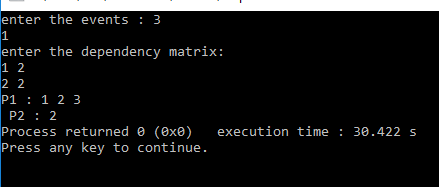
printf("\n P2 : ");

for(j=0;j<e2;j++)

printf("%d ",p2[j]);

}

Output:-



**4. To implement Vector Clock Algorithm.**

Code:-

#include<bits/stdc++.h>

using namespace std;

int main()

{

int n;

cout<<"enter number of process\n";

cin>>n;

vector<int>event(n);

cout<<"enter number of events for each process\n";

for(int i=0;i<n;i++)

cin>>event[i];

vector<vector<vector<int> > >arr(n);

vector<int>prr(n,0);

for(int i=0;i<n;i++)

for(int j=0;j<event[i];j++)

arr[i].push\_back(prr);

for(int i=0;i<n;i++)

for(int j=0;j<event[i];j++)

arr[i][j][i]=j+1;

int msg;

cout<<"enter no of message\n";

cin>>msg;

int senp,sene,recp,rece;

cout<<"enter sender process,sender event,receiver process,receiver event\n";

for(int i=0;i<msg;i++)

{

cin>>senp>>sene>>recp>>rece;

for(int j=rece-1;j<event[recp-1];j++)

{

arr[recp-1][j][senp-1]=sene;

}

}

cout<<"vector timestamp for given processes is given as:-\n";

for(int i=0;i<n;i++)

{

for(int j=0;j<event[i];j++)

{

for(int k=0;k<n;k++)

cout<<arr[i][j][k]<<",";

cout<<" ";

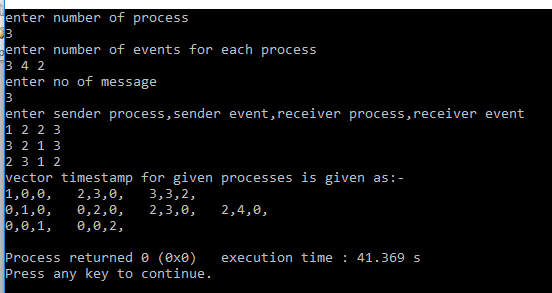
}

cout<<endl;

}

}

Output:-



**5.To implement program for Bully Election Algorithm.**

Code:-

|  |  |
| --- | --- |
|  | #include<stdio.h>  #include<conio.h>  #include<alloc.h>  #include<stdlib.h>    struct process {     int no;     int priority;     int active;  struct process \*next;  };  typedef struct process proc;    struct priority {     int pri;     struct priority \*next;     proc \*pp;  };  typedef struct priority pri;    pri\* find\_priority(proc \*head, pri \*head1) {     proc \*p1;     pri \*p2, \*p3;     p1 = head;       while (p1->next != head) {        if (p1->active == 1) {           if (head1 == NULL) {              head1 = (pri\*) malloc(sizeof(pri));              head1->pri = p1->priority;              head1->next = NULL;              head1->pp = p1;              p2 = head1;           } else {              p3 = (pri\*) malloc(sizeof(pri));              p3->pri = p1->priority;              p3->pp = p1;              p3->next = NULL;              p2->next = p3;              p2 = p2->next;           }           p1 = p1->next;        } else           p1 = p1->next;     } //end while       p3 = (pri\*) malloc(sizeof(pri));     p3->pri = p1->priority;     p3->pp = p1;     p3->next = NULL;     p2->next = p3;     p2 = p2->next;     p3 = head1;       return head1;  } //end find\_priority()    int find\_max\_priority(pri \*head) {     pri \*p1;     int max = -1;     int i = 0;     p1 = head;       while (p1 != NULL) {        if (max < p1->pri && p1->pp->active == 1) {           max = p1->pri;           i = p1->pp->no;        }        p1 = p1->next;     }     return i;  }    void bully() {     proc \*head;     proc \*p1;     proc \*p2;     int n, i, pr, maxpri, a, pid, max, o;     char ch;       head = p1 = p2 = NULL;       printf("\nnEnter how many process: ");     scanf("%d", &n);       for (i = 0; i < n; i++) {        printf("\nEnter priority of process %d: ", i + 1);        scanf("%d", &pr);          printf("\nIs process with id %d is active ?(0/1) :", i + 1);        scanf("%d", &a);      if (head == NULL) {           head = (proc\*) malloc(sizeof(proc));           if (head == NULL) {              printf("\nMemory cannot be allocated");              getch();              exit(0);           }           head->no = i + 1;           head->priority = pr;           head->active = a;           head->next = head;           p1 = head;        } else {           p2 = (proc\*) malloc(sizeof(proc));           if (p2 == NULL) {              printf("\nMemory cannot be allocated");              getch();              exit(0);           }           p2->no = i + 1;           p2->priority = pr;           p2->active = a;           p1->next = p2;           p2->next = head;           p1 = p2;        }     } //end for       printf("\nEnter the process id that invokes election algorithm: ");     scanf("%d", &pid);     p2 = head;     while (p2->next != head) {        if (p2->no == pid) {           p2 = p2->next;           break;        }        p2 = p2->next;     }       printf("\nProcess with id %d has invoked election algorithm", pid);     printf("\t\nElection message is sent to processes");       while (p2->next != head) {        if (p2->no > pid)           printf("%d", p2->no);        p2 = p2->next;     }       printf("%d", p2->no);     p2 = head;     max = 0;       while (1) {        if (p2->priority > max && p2->active == 1)           max = p2->no;        p2 = p2->next;        if (p2 == head)           break;     }       printf("\n\tProcess with the id %d is the co-ordinator", max);       while (1) {        printf("\nDo you want to continue?(y/n): ");        flushall();        scanf("%c", &ch);        if (ch == 'n' || ch == 'N')           break;        p2 = head;          while (1) {           printf("\nEnter the process with id %d is active or not (0/1): ",                 p2->no);           scanf("%d", &p2->active);           p2 = p2->next;           if (p2 == head)              break;        }          printf("\nEnter the process id that invokes election algorithm: ");        scanf("%d", &pid);          printf("\n\tElection message is sent to processes ");          while (p2->next != head) {           if (p2->no > pid)              printf("%d", p2->no);           p2 = p2->next;        }        printf("%d", p2->no);        p2 = head;        max = 0;          while (1) {           if (p2->no > max && p2->active == 1)              max = p2->no;           p2 = p2->next;           if (p2 == head)              break;        }        printf("\n\tProcess with id %d is the co-ordinator", max);     }  }    void main() {     clrscr();     bully();     getch();  } |
|  |  |

**6. To implement Ricarts 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>   
  
#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

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   
//  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 %sn",argv[1],argv[3],argv[2]);   
     printf("Configuration Filen"); //Printing the configuration file details   
     for(i=1;i<=noproc;i++)   
     {   
          printf("%d %s %dn",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 addressn");   
          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), '', 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), '', 8);  // Zero the rest of the struct   
                  sleep(1);   
  
                if (connect(sockfds, (struct sockaddr \*)&their\_addr, sizeof(struct sockaddr)) == -1)   
                    {   
                        perror("connect in send threadn");   
                      exit(1);   
                   }   
  
               printf("Clock value is updated to %dn",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 %dn",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);   
     }   
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   
//\*\* 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 Emptyn");   
  
     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;   
}

**7. To implement Dining Philosophers Problem in C/C++**

Code:-

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#include <unistd.h>

#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (phnum + 4) % N

#define RIGHT (phnum + 1) % N

int state[N];

int phil[N] = { 0, 1, 2, 3, 4 };

sem\_t mutex;

sem\_t S[N];

void test(int phnum)

{

if (state[phnum] == HUNGRY

&& state[LEFT] != EATING

&& state[RIGHT] != EATING) {

// state that eating

state[phnum] = EATING;

Sleep(2);

printf("Philosopher %d takes fork %d and %d\n",

phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is Eating\n", phnum + 1);

// sem\_post(&S[phnum]) has no effect

// during takefork

// used to wake up hungry philosophers

// during putfork

sem\_post(&S[phnum]);

}

}

// take up chopsticks

void take\_fork(int phnum)

{

sem\_wait(&mutex);

// state that hungry

state[phnum] = HUNGRY;

printf("Philosopher %d is Hungry\n", phnum + 1);

// eat if neighbours are not eating

test(phnum);

sem\_post(&mutex);

// if unable to eat wait to be signalled

sem\_wait(&S[phnum]);

Sleep(1);

}

// put down chopsticks

void put\_fork(int phnum)

{

sem\_wait(&mutex);

// state that thinking

state[phnum] = THINKING;

printf("Philosopher %d putting fork %d and %d down\n",

phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is thinking\n", phnum + 1);

test(LEFT);

test(RIGHT);

sem\_post(&mutex);

}

void\* philospher(void\* num)

{

while (1) {

int\* i = num;

Sleep(1);

take\_fork(\*i);

Sleep(0);

put\_fork(\*i);

}

}

int main()

{

int i;

pthread\_t thread\_id[N];

// initialize the semaphores

sem\_init(&mutex, 0, 1);

for (i = 0; i < N; i++)

sem\_init(&S[i], 0, 0);

for (i = 0; i < N; i++) {

// create philosopher processes

pthread\_create(&thread\_id[i], NULL,

philospher, &phil[i]);

printf("Philosopher %d is thinking\n", i + 1);

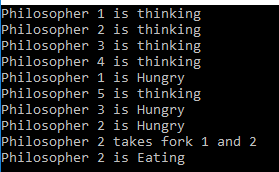
}

for (i = 0; i < N; i++)

pthread\_join(thread\_id[i], NULL);

}

Output:-



**8. To implement Banker’s Deadlock Avoidance algorithm.**

Code:-

#include <stdio.h>

int current[5][5], maximum\_claim[5][5], available[5];

int allocation[5] = {0, 0, 0, 0, 0};

int maxres[5], running[5], safe = 0;

int counter = 0, i, j, exec, resources, processes, k = 1;

int main()

{

printf("\nEnter number of processes: ");

scanf("%d", &processes);

for (i = 0; i < processes; i++)

{

running[i] = 1;

counter++;

}

printf("\nEnter number of resources: ");

scanf("%d", &resources);

printf("\nEnter Claim Vector:");

for (i = 0; i < resources; i++)

{

scanf("%d", &maxres[i]);

}

printf("\nEnter Allocated Resource Table:\n");

for (i = 0; i < processes; i++)

{

for(j = 0; j < resources; j++)

{

scanf("%d", &current[i][j]);

}

}

printf("\nEnter Maximum Claim Table:\n");

for (i = 0; i < processes; i++)

{

for(j = 0; j < resources; j++)

{

scanf("%d", &maximum\_claim[i][j]);

}

}

printf("\nThe Claim Vector is: ");

for (i = 0; i < resources; i++)

{

printf("\t%d", maxres[i]);

}

printf("\nThe Allocated Resource Table:\n");

for (i = 0; i < processes; i++)

{

for (j = 0; j < resources; j++)

{

printf("\t%d", current[i][j]);

}

printf("\n");

}

printf("\nThe Maximum Claim Table:\n");

for (i = 0; i < processes; i++)

{

for (j = 0; j < resources; j++)

{

printf("\t%d", maximum\_claim[i][j]);

}

printf("\n");

}

for (i = 0; i < processes; i++)

{

for (j = 0; j < resources; j++)

{

allocation[j] += current[i][j];

}

}

printf("\nAllocated resources:");

for (i = 0; i < resources; i++)

{

printf("\t%d", allocation[i]);

}

for (i = 0; i < resources; i++)

{

available[i] = maxres[i] - allocation[i];

}

printf("\nAvailable resources:");

for (i = 0; i < resources; i++)

{

printf("\t%d", available[i]);

}

printf("\n");

while (counter != 0)

{

safe = 0;

for (i = 0; i < processes; i++)

{

if (running[i])

{

exec = 1;

for (j = 0; j < resources; j++)

{

if (maximum\_claim[i][j] - current[i][j] > available[j])

{

exec = 0;

break;

}

}

if (exec)

{

printf("\nProcess%d is executing\n", i + 1);

running[i] = 0;

counter--;

safe = 1;

for (j = 0; j < resources; j++)

{

available[j] += current[i][j];

}

break;

}

}

}

if (!safe)

{

printf("\nThe processes are in unsafe state.\n");

break;

}

else

{

printf("\nThe process is in safe state");

printf("\nAvailable vector:");

for (i = 0; i < resources; i++)

{

printf("\t%d", available[i]);

}

printf("\n");

}

}

return 0;

}

Output:-

Enter number of processes: 3

Enter number of resources: 2

Enter Claim Vector:3 4

Enter Allocated Resource Table:

1 2

2 0

0 0

Enter Maximum Claim Table:

2 3

2 1

1 3

The Claim Vector is: 3 4

The Allocated Resource Table:

1 2

2 0

0 0

The Maximum Claim Table:

2 3

2 1

1 3

Allocated resources: 3 2

Available resources: 0 2

Process2 is executing

The process is in safe state

Available vector: 2 2

Process1 is executing

The process is in safe state

Available vector: 3 4

Process3 is executing

The process is in safe state

Available vector: 3 4

**9. To implement deadlock detection algorithm.**

Code:-

#include<stdio.h>  
static int mark[20];  
int i,j,np,nr;  
  
int main()  
{  
int alloc[10][10],request[10][10],avail[10],r[10],w[10];  
  
printf("\nEnter the no of process: ");  
scanf("%d",&np);  
printf("\nEnter the no of resources: ");  
scanf("%d",&nr);  
for(i=0;i<nr;i++)  
{  
printf("\nTotal Amount of the Resource R%d: ",i+1);  
scanf("%d",&r[i]);  
}  
printf("\nEnter the request matrix:");  
  
for(i=0;i<np;i++)  
for(j=0;j<nr;j++)  
scanf("%d",&request[i][j]);  
  
printf("\nEnter the allocation matrix:");  
for(i=0;i<np;i++)  
for(j=0;j<nr;j++)  
scanf("%d",&alloc[i][j]);  
/\*Available Resource calculation\*/  
for(j=0;j<nr;j++)  
{  
avail[j]=r[j];  
for(i=0;i<np;i++)  
{  
avail[j]-=alloc[i][j];  
  
}  
}  
  
//marking processes with zero allocation  
  
for(i=0;i<np;i++)  
{  
int count=0;  
 for(j=0;j<nr;j++)  
   {  
      if(alloc[i][j]==0)  
        count++;  
      else  
        break;  
    }  
 if(count==nr)  
 mark[i]=1;  
}  
// initialize W with avail  
  
for(j=0;j<nr;j++)  
    w[j]=avail[j];  
  
//mark processes with request less than or equal to W  
for(i=0;i<np;i++)  
{  
int canbeprocessed=0;  
 if(mark[i]!=1)  
{  
   for(j=0;j<nr;j++)  
    {  
      if(request[i][j]<=w[j])  
        canbeprocessed=1;  
      else  
         {  
         canbeprocessed=0;  
         break;  
          }  
     }  
if(canbeprocessed)  
{  
mark[i]=1;  
  
for(j=0;j<nr;j++)  
w[j]+=alloc[i][j];  
}  
}  
}  
  
//checking for unmarked processes  
int deadlock=0;  
for(i=0;i<np;i++)  
if(mark[i]!=1)  
deadlock=1;  
  
  
if(deadlock)  
printf("\n Deadlock detected");  
else  
printf("\n No Deadlock possible");  
}

Output:-

Enter the no of process: 4  
Enter the no of resources: 5  
  
Total Amount of the Resource R1: 2  
Total Amount of the Resource R2: 1  
Total Amount of the Resource R3: 1  
Total Amount of the Resource R4: 2  
Total Amount of the Resource R5: 1  
  
Enter the request matrix:0 1 0 0 1  
0 0 1 0 1  
0 0 0 0 1  
1 0 1 0 1  
  
Enter the allocation matrix:1 0 1 1 0  
1 1 0 0 0  
0 0 0 1 0  
0 0 0 0 0  
  
 Deadlock detected

**10. To implement Sleeping Barber Problem of mutual exclusion.**

Code:-

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#define seats 6

void \*customerMaker();

void \*barberShop();

void \*waitingRoom();

void checkQueue();

pthread\_mutex\_t queue\_mutex = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_mutex\_t wait\_mutex = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_mutex\_t sleep\_mutex = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_cond\_t barberSleep\_cond = PTHREAD\_COND\_INITIALIZER;

pthread\_cond\_t barberWorking\_cond = PTHREAD\_COND\_INITIALIZER;

int returnTime=5,current=0, sleeping=0, iseed;

int main(int argc, char \*argv[])

{

iseed=time(NULL);

srand(iseed);

//declare barber thread;

pthread\_t barber,customerM,timer\_thread;

pthread\_attr\_t barberAttr, timerAttr;

pthread\_attr\_t customerMAttr;

//define barber, and cutomerMaker default attributes

pthread\_attr\_init(&timerAttr);

pthread\_attr\_init(&barberAttr);

pthread\_attr\_init(&customerMAttr);

printf("\n");

//create cutomerMaker

pthread\_create(&customerM,&customerMAttr,customerMaker,NULL);

//create barber

pthread\_create(&barber,&barberAttr,barberShop,NULL);

pthread\_join(barber,NULL);

pthread\_join(customerM,NULL);

return 0;

}

void \*customerMaker()

{

int i=0;

printf("\*Customer Maker Created\*\n\n");

fflush(stdout);

pthread\_t customer[seats+1];

pthread\_attr\_t customerAttr[seats+1];

while(i<(seats+1))

{

i++;

pthread\_attr\_init(&customerAttr[i]);

while(rand()%2!=1)

{

Sleep(1);

}

pthread\_create(&customer[i],&customerAttr[i],waitingRoom,NULL);

}

pthread\_exit(0);

}

void \*waitingRoom()

{

//take seat

pthread\_mutex\_lock(&queue\_mutex);

checkQueue();

Sleep(returnTime);

waitingRoom();

}

void \*barberShop()

{

int loop=0;

printf("The barber has opened the store.\n");

fflush(stdout);

while(loop==0)

{

if(current==0)

{

printf("\tThe shop is empty, barber is sleeping.\n");

fflush(stdout);

pthread\_mutex\_lock(&sleep\_mutex);

sleeping=1;

pthread\_cond\_wait(&barberSleep\_cond,&sleep\_mutex);

sleeping=0;

pthread\_mutex\_unlock(&sleep\_mutex);

printf("\t\t\t\tBarber wakes up.\n");

fflush(stdout);

}

else

{

printf("\t\t\tBarber begins cutting hair.\n");

fflush(stdout);

Sleep((rand()%20)/5);

current--;

printf("\t\t\t\tHair cut complete, customer leaving store.\n");

pthread\_cond\_signal(&barberWorking\_cond);

}

}

pthread\_exit(0);

}

void checkQueue()

{

current++;

printf("\tCustomer has arrived in the waiting room.\t\t\t\t\t\t\t%d Customers in store.\n",current);

fflush(stdout);

printf("\t\tCustomer checking chairs.\n");

fflush(stdout);

if(current<seats)

{

if(sleeping==1)

{

printf("\t\t\tBarber is sleeping, customer wakes him.\n");

fflush(stdout);

pthread\_cond\_signal(&barberSleep\_cond);

}

printf("\t\tCustomer takes a seat.\n");

fflush(stdout);

pthread\_mutex\_unlock(&queue\_mutex);

pthread\_mutex\_lock(&wait\_mutex);

pthread\_cond\_wait(&barberWorking\_cond,&wait\_mutex);

pthread\_mutex\_unlock(&wait\_mutex);

return;

}

if(current>=seats)

{

printf("\t\tAll chairs full, leaving store.\n");

fflush(stdout);

current--;

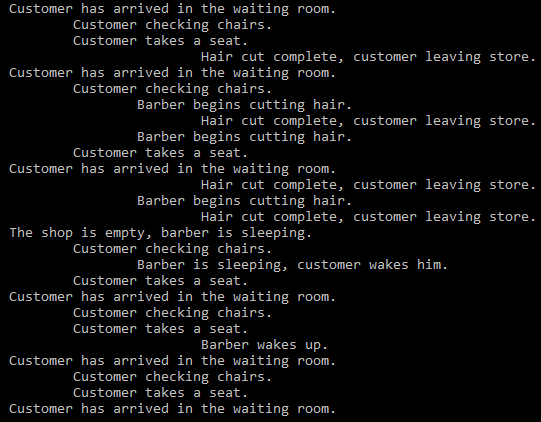
pthread\_mutex\_unlock(&queue\_mutex);

return;

}

}

Output:-



**List of experiment’s**

|  |  |
| --- | --- |
| S.No | Contents |
| 1. | To implement Producer Consumer using common stack |
| 2. | Implement reader writer problem using monitors |
| 3. | Implement Lamport’s Logical Clock Algorithm |
| 4. | Implement Vector Clock Algorithm |
| 5. | Implement Bully Election’s Algorithm |
| 6. | Implement Ricarts Agrawala Algorithm |
| 7. | Implement Dinning Philosopher Problem |
| 8. | Implement Bankers Deadlock Avoidance Algorithm |
| 9. | Implement Deadlock Detection Algorithm |
| 10. | Implement Sleeping Barber Problem in C/C++. |

Department of Computer Science & Engineering

Dr. B R Ambedkar National Institute of Technology

Jalandhar-144011,Punjab (India)



ADVANCED OPERATING SYSTEMS LABORATORY

(CSX-352)

LAB PRACTICALS RECORD

COMPUTER SCIENCE AND ENGINEERING