11. Illustrate the concept of multithreading using a C program.

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<pthread.h>

void\*myThreadFun(void \*vargp)

{

sleep(1);

printf("Printing geeksQuiz from Thread \n");

return NULL;

}

int main()

{

pthread\_t thread\_id;

printf("Befor Thread\n");

pthread\_create(&thread\_id,NULL,myThreadFun,NULL);

pthread\_join(thread\_id,NULL);

printf("After Thread\n");

exit(0);

}

12. Design a C program to simulate the concept of Dining-Philosophers problem.

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<semaphore.h>

#include<unistd.h>

sem\_t room;

sem\_t chopstick[5];

void\*philosopher(void \*);

void eat(int);

int main()

{

int i,a[5];

pthread\_t tid[5];

sem\_init(&room,0,4);

for(i=0;i<5;i++)

sem\_init(&chopstick[i],0,1);

for(i=0;i<5;i++)

{

a[i]=i;

pthread\_create(&tid[i],NULL,philosopher,(void\*)&a[i]);

}

for(i=0;i<5;i++)

pthread\_join(tid[i],NULL);

}

void \*philosopher(void \* num)

{

int phil=\*(int\*)num;

sem\_wait(&room);

printf("\nPhilosopher %d has entered room",phil);

sem\_wait(&chopstick[phil]);

sem\_wait(&chopstick[(phil+1)%5]);

eat(phil);

sleep(2);

printf("\nPhilosopher %d has finished eating",phil);

sem\_post(&chopstick[(phil+1)%5]);

sem\_post(&chopstick[phil]);

sem\_post(&room);

}

void eat(int phil)

{

printf("\nPhilosopher %d is eating",phil);

}

13. Construct a C program for implementation the various memory allocation strategies.

#include<stdio.h>

int main()

{

int bsize[10],psize[10],bno,pno,flags[10],allocation[10],i,j;

for(i=0;i<10;i++)

{

flags[i]=0;

allocation[i]=-1;

}

printf("enter no.of blocks: ");

scanf("%d",&bno);

printf("\nenter size of each block: ");

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

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

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

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

if(flags[j]==0&&bsize[j]>=psize[i])

{

allocation[j]=i;

flags[j]=1;

break;

}

printf("\nBlock no. \tsize\t\tprocess no.\t\tsize");

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

{

printf("\n%d\t\t%d\t\t",i+1,bsize);

if(flags[i]==1)

printf("%d\t\t%d",allocation[i]+1,psize[allocation[i]]);

else

printf("not allowed");

}

}

14. Construct a C program to organize the file using single level directory.

#include<stdio.h>

#include<conio.h>

#include<string.h>

int main()

{

int nf=0,i=0,j=0,ch;

char mdname[10],fname[10][10],name[10];

printf("enter the directory name: ");

scanf("%s",mdname);

printf("enter the number of files: ");

scanf("%d",&nf);

do

{

printf("enter file name to be created: ");

scanf("%s",name);

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

{

if(!strcmp(name,fname[i]))

break;

}

if(i==nf)

{

strcpy(fname[j++],name);

nf++;

}

else

printf("there is already %s\n",name);

printf("do you want to enter another file(yes -1 or n0 -0):");

scanf("%d",&ch);

}

while(ch==1);

printf("directory name is:%s\n",mdname);

printf("files names are: ");

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

printf("\n%s",fname[i]);

getch();

}

15. Design a C program to organize the file using two level directory structure.

#include<stdio.h>

#include<conio.h>

struct st

{

char dname[10];

char sdname[10][10];

char fname[10][10][10];

int ds,sds[10];

}dir[10];

int main()

{

int i,j,k,n;

printf("enter number of directories: ");

scanf("%d",&n);

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

{

printf("enter directory %d names:",i+1);

scanf("%s",&dir[i].dname);

printf("enter size of directories: ");

scanf("%d",&dir[i].ds);

for(j=0;j<dir[i].ds;j++)

{

printf("enter subdirectory name and size:");

scanf("%s",&dir[i].sdname[j]);

scanf("%d",&dir[i].sds[j]);

for(k=0;k<dir[i].sds[j];k++)

{

printf("enter file name: ");

scanf("%s",&dir[i].fname[j][k]);

}

}

}

printf("\ndirname\t\tsize\tsubdirname\tsize\tfiles");

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

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

{

printf("%s\t\t%d",dir[i].dname,dir[i].ds);

for(j=0;j<dir[i].ds;j++)

{

printf("\t%s\t\t%d\t",dir[i].sdname[j],dir[i].sds[j]);

for(k=0;k<dir[i].sds[j];k++)

printf("%s\t",dir[i].fname[j][k]);

printf("\n\t\t");

}

printf("\n");

}

getch();

}

16. Develop a C program for implementing random access file for processing the employee

details.

#include<stdio.h>

#include<stdlib.h>

struct employee

{

int empid;

char empname[50];

float empsalary;

};

void addemployee(FILE \*file)

{

struct employee emp;

printf("enter employee id: ");

scanf("%d",&emp.empid);

printf("enter employee name: ");

scanf("%s",&emp.empname);

printf("enter employee salsry: ");

scanf("%f",&emp.empsalary);

fseek(file,0,SEEK\_END);

fwrite(&emp,sizeof(struct employee),1,file);

printf("employee added successfully!\n");

}

void displayallemployees(FILE \*file)

{

struct employee emp;

fseek(file,0,SEEK\_SET);

while(fread(&emp,sizeof(struct employee),1,file)==1)

{

printf("employee id:%d\n",emp.empid);

printf("employee name:%s\n",emp.empname);

printf("employee salary:%.2f\n",emp.empsalary);

printf("---------------------\n");

}

}

int main()

{

FILE \*file;

int choice;

file=fopen("employee.dat","rb+");

if(file==NULL)

{

file=fopen("employee.dat","wb+");

if(file==NULL)

{

perror("error creating the file");

exit(EXIT\_FAILURE);

}

}

do

{

printf("\nEmployee Management System\n");

printf("1.Add employee\n");

printf("2.Display all employees\n");

printf("3.Exit\n");

printf("enter your choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1: addemployee(file);

break;

case 2:

displayallemployees(file);

break;

case 3:

printf("Exiting program.Goodbye!\n");

break;

default:

printf("Invalid choice.please try again.\n");

}

}

while(choice!=3);

fclose(file);

return 0;

}

17. Illustrate the deadlock avoidance concept by simulating Banker’s algorithm with C.

#include<stdio.h>

#include<conio.h>

int max[100][100];

int alloc[100][100];

int need[100][100];

int avail[100];

int n,r;

void input();

void show();

void cal();

int main()

{

int i,j;

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Banker's Algo\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n'");

input();

show();

cal();

getch();

return 0;

}

void input()

{

int i,j;

printf("enter the number of processes\t");

scanf("%d",&n);

printf("enter the number of resourses instances\t");

scanf("%d",&r);

printf("enter the max matrix\n");

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

{

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

{

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

}

}

printf("enter the allocation matrix\n");

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

{

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

{

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

}

}

printf("enter the available resourses\n");

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

{

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

}

}

void show()

{

int i,j;

printf("process\t allocation\t max\t available\t");

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

{

printf("\nP%d\t",i+1);

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

{

printf("%d",alloc[i][j]);

}

printf("\t");

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

{

printf("%d",max[i][j]);

}

printf("\t");

if(i==0)

{

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

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

}

}

}

void cal()

{

int finish[100],temp,need[100][100],flag=1,k,cl=0;

int safe[100];

int i,j;

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

{

finish[i]=0;

}

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

{

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

{

need[i][j]=max[i][j]-alloc[i][j];

}

}

printf("\n");

while(flag)

{

flag=0;

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

{

int c=0;

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

{

if((finish[i]==0)&&(need[i][j]<=avail[j]))

{

c++;

if(c==r)

{

for(k=0;k<r;k++)

{

avail[k]+=alloc[i][j];

finish[i]=1;

flag=1;

}

printf("p%d->",i);

if(finish[i]==1)

{

i=n;

}

}

}

}

}

}

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

{

if(finish[i]==1)

{

cl++;

}

else

{

printf("P%d->",i);

}

}

if(cl==n)

{

printf("\n the system is in safe");

}

else

{

printf("\n process are in dead lock");

printf("\n system is in unsafe state");

}

18 Construct a C program to simulate producer-consumer problem using semaphores.

#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;

}

}

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 produce 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);

}

19. Design a C program to implement process synchronization using mutex locks.

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define NUM\_THREADS 2

#define NUM\_INCREMENTS 1000000

int shared\_variable = 0;

pthread\_mutex\_t mutex;

void \*thread\_function(void \*arg) {

for (int i = 0; i < NUM\_INCREMENTS; i++) {

pthread\_mutex\_lock(&mutex);

shared\_variable++;

pthread\_mutex\_unlock(&mutex);

}

pthread\_exit(NULL);

}

int main() {

pthread\_t threads[NUM\_THREADS];

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

for (int i = 0; i < NUM\_THREADS; i++) {

if (pthread\_create(&threads[i], NULL, thread\_function, NULL) != 0) {

perror("Error creating thread");

exit(1);

}

}

for (int i = 0; i < NUM\_THREADS; i++) {

if (pthread\_join(threads[i], NULL) != 0) {

perror("Error joining thread");

exit(1);

}

pthread\_mutex\_destroy(&mutex);

printf("Shared variable: %d\n", shared\_variable);

return 0;

}

}

20. Construct a C program to simulate Reader-Writer problem using Semaphores.

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define MAX\_READERS 5

#define MAX\_WRITERS 2

sem\_t mutex, writeMutex;

int readersCount = 0;

void \*reader(void \*arg) {

int readerID = \*(int \*)arg;

while (1) {

sem\_wait(&mutex);

readersCount++;

if (readersCount == 1) {

sem\_wait(&writeMutex);

}

sem\_post(&mutex);

printf("Reader %d is reading\n", readerID);

sleep(1);

sem\_wait(&mutex);

readersCount--;

if (readersCount == 0) {

sem\_post(&writeMutex);

}

sem\_post(&mutex);

sleep(1);

}

}

void \*writer(void \*arg) {

int writerID = \*(int \*)arg;

while (1) {

sem\_wait(&writeMutex);

printf("Writer %d is writing\n", writerID);

sleep(2);

sem\_post(&writeMutex);

sleep(2);

}

}

int main() {

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

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

pthread\_t readers[MAX\_READERS], writers[MAX\_WRITERS];

int readerIDs[MAX\_READERS], writerIDs[MAX\_WRITERS];

for (int i = 0; i < MAX\_READERS; i++) {

readerIDs[i] = i + 1;

pthread\_create(&readers[i], NULL, reader, &readerIDs[i]);

}

for (int i = 0; i < MAX\_WRITERS; i++) {

writerIDs[i] = i + 1;

pthread\_create(&writers[i], NULL, writer, &writerIDs[i]);

}

for (int i = 0; i < MAX\_READERS; i++) {

pthread\_join(readers[i], NULL);

}

for (int i = 0; i < MAX\_WRITERS; i++) {

pthread\_join(writers[i], NULL);

}

sem\_destroy(&mutex);

sem\_destroy(&writeMutex);

return 0;

}