**FCFS CPU SCHEDULING ALGORITHM**

#include<stdio.h>

#include<conio.h>

main()

{

int bt[20], wt[20], tat[20], i, n;

float wtavg, tatavg;

clrscr();

printf("\nEnter the number of processes -- ");

scanf("%d", &n);

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

{

printf("\nEnter Burst Time for Process %d -- ", i);

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

}

wt[0] = wtavg = 0;

tat[0] = tatavg = bt[0];

for(i=1;i<n;i++)

{

wt[i] = wt[i-1] +bt[i-1];

tat[i] = tat[i-1] +bt[i];

wtavg = wtavg + wt[i];

tatavg = tatavg + tat[i];

}

printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");

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

printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]);

printf("\nAverage Waiting Time -- %f", wtavg/n);

printf("\nAverage Turnaround Time -- %f", tatavg/n);

getch();

}

INPUT

Enter the number of processes -- 3

Enter Burst Time for Process 0 -- 24

Enter Burst Time for Process 1 -- 3

Enter Burst Time for Process 2 -- 3

OUTPUT

PROCESS BURST TIME WAITING TIME TURNAROUND TIME

P0 24 0 24

P1 3 24 27

P2 3 27 30

Average Waiting Time-- 17.000000

Average Turnaround Time -- 27.000000

**SJF CPU SCHEDULING ALGORITHM**

#include<stdio.h>

#include<conio.h>

main()

{

int p[20], bt[20], wt[20], tat[20], i, k, n, temp;

float wtavg, tatavg;

clrscr();

printf("\nEnter the number of processes -- ");

scanf("%d", &n);

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

{

p[i]=i;

printf("Enter Burst Time for Process %d -- ", i);

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

}

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

for(k=i+1;k<n;k++)

if(bt[i]>bt[k])

{

temp=bt[i];

bt[i]=bt[k];

bt[k]=temp;

temp=p[i];

p[i]=p[k];

p[k]=temp;

}

wt[0] = wtavg = 0;

tat[0] = tatavg = bt[0];

for(i=1;i<n;i++)

{

wt[i] = wt[i-1] +bt[i-1];

tat[i] = tat[i-1] +bt[i];

wtavg = wtavg + wt[i];

tatavg = tatavg + tat[i];

}

printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");

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

printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);

printf("\nAverage Waiting Time -- %f", wtavg/n);

printf("\nAverage Turnaround Time -- %f", tatavg/n);

getch();

}

INPUT

Enter the number of processes -- 4

Enter Burst Time for Process 0 -- 6

Enter Burst Time for Process 1 -- 8

Enter Burst Time for Process 2 -- 7

Enter Burst Time for Process 3 -- 3

OUTPUT

PROCESS BURST TIME WAITING TIME TURNAROUND TIME

P3 3 0 3

P0 6 3 9

P2 7 9 16

P1 8 16 24

Average Waiting Time -- 7.000000

Average Turnaround Time -- 13.000000

**ROUND ROBIN CPU SCHEDULING ALGORITHM**

#include<stdio.h>

main()

{

int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;

float awt=0,att=0,temp=0;

clrscr();

printf("Enter the no of processes -- ");

scanf("%d",&n);

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

{

printf("\nEnter Burst Time for process %d -- ", i+1);

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

ct[i]=bu[i];

}

printf("\nEnter the size of time slice -- ");

scanf("%d",&t);

max=bu[0];

for(i=1;i<n;i++)

if(max<bu[i])

max=bu[i];

for(j=0;j<(max/t)+1;j++)

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

if(bu[i]!=0)

if(bu[i]<=t)

{

tat[i]=temp+bu[i];

temp=temp+bu[i];

bu[i]=0;

}

else

{

bu[i]=bu[i]-t;

temp=temp+t;

}

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

{

wa[i]=tat[i]-ct[i];

att+=tat[i];

awt+=wa[i];

}

printf("\nThe Average Turnaround time is -- %f",att/n);

printf("\nThe Average Waiting time is -- %f ",awt/n);

printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");

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

printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);

getch();

}

INPUT

Enter the no of processes – 3

Enter Burst Time for process 1 – 24

Enter Burst Time for process 2 -- 3

Enter Burst Time for process 3 -- 3

Enter the size of time slice – 3

OUTPUT

The Average Turnaround time is – 15.666667

The Average Waiting time is -- 5.666667

PROCESS BURST TIME WAITING TIME TURNAROUND TIME

1 24 6 30

2 3 4 7

3 3 7 10

**PRIORITY CPU SCHEDULING ALGORITHM**

#include<stdio.h>

main()

{

int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp;

float wtavg, tatavg;

clrscr();

printf("Enter the number of processes --- ");

scanf("%d",&n);

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

{

p[i] = i;

printf("Enter the Burst Time & Priority of Process %d --- ",i);

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

}

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

for(k=i+1;k<n;k++)

if(pri[i] > pri[k])

{

temp=p[i];

p[i]=p[k];

p[k]=temp;

temp=bt[i];

bt[i]=bt[k];

bt[k]=temp;

temp=pri[i];

pri[i]=pri[k];

pri[k]=temp;

}

wtavg = wt[0] = 0;

tatavg = tat[0] = bt[0];

for(i=1;i<n;i++)

{

wt[i] = wt[i-1] + bt[i-1];

tat[i] = tat[i-1] + bt[i];

wtavg = wtavg + wt[i];

tatavg = tatavg + tat[i];

}

printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME");

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

printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],pri[i],bt[i],wt[i],tat[i]);

printf("\nAverage Waiting Time is --- %f",wtavg/n);

printf("\nAverage Turnaround Time is --- %f",tatavg/n);

getch();

}

INPUT

Enter the number of processes -- 5

Enter the Burst Time & Priority of Process 0 --- 10 3

Enter the Burst Time & Priority of Process 1 --- 1 1

Enter the Burst Time & Priority of Process 2 --- 2 4

Enter the Burst Time & Priority of Process 3 --- 1 5

Enter the Burst Time & Priority of Process 4 --- 5 2

OUTPUT

PROCESS PRIORITY BURST TIME WAITING TIME TURNAROUND TIME

1 1 1 0 1

4 2 5 1 6

0 3 10 6 16

2 4 2 16 18

3 5 1 18 19

Average Waiting Time is --- 8.200000

Average Turnaround Time is --- 12.000000

**Write a C program to simulate paging technique of memory management**

**PROGRAM**

#include<stdio.h>

#include<conio.h>

main()

{

int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;

int s[10], fno[10][20];

clrscr();

printf("\nEnter the memory size -- ");

scanf("%d",&ms);

printf("\nEnter the page size -- ");

scanf("%d",&ps);

nop = ms/ps;

printf("\nThe no. of pages available in memory are -- %d ",nop);

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

scanf("%d",&np);

rempages = nop;

for(i=1;i<=np;i++)

{

printf("\nEnter no. of pages required for p[%d]-- ",i);

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

if(s[i] >rempages)

{

printf("\nMemory is Full");

break;

}

rempages = rempages - s[i];

printf("\nEnter pagetable for p[%d] --- ",i);

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

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

}

printf("\nEnter Logical Address to find Physical Address ");

printf("\nEnter process no. and pagenumber and offset -- ");

scanf("%d %d %d",&x,&y, &offset);

if(x>np || y>=s[i] || offset>=ps)

printf("\nInvalid Process or Page Number or offset");

else

{

pa=fno[x][y]\*ps+offset;

printf("\nThe Physical Address is -- %d",pa);

}

getch();

}

INPUT

Enter the memory size – 1000

Enter the page size -- 100

The no. of pages available in memory are -- 10

Enter number of processes -- 3

Enter no. of pages required for p[1] -- 4

Enter pagetable for p[1] --- 8 6 9 5

Enter no. of pages required for p[2] -- 5

Enter pagetable for p[2] --- 1 4 5 7 3

Enter no. of pages required for p[3] -- 5

OUTPUT

Memory is Full

Enter Logical Address to find Physical Address

Enter process no. and pagenumber and offset -- 2 3 60

The Physical Address is -- 760**Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.**

PROGRAM

#include<stdio.h>

struct file

{

int all[10];

int max[10];

int need[10];

int flag;

};

void main()

{

struct file f[10];

int fl;

int i, j, k, p, b, n, r, g, cnt=0, id, newr;

int avail[10],seq[10];

clrscr();

printf("Enter number of processes -- ");

scanf("%d",&n);

printf("Enter number of resources -- ");

scanf("%d",&r);

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

{

printf("Enter details for P%d",i);

printf("\nEnter allocation\t -- \t");

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

scanf("%d",&f[i].all[j]);

printf("Enter Max\t\t -- \t");

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

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

f[i].flag=0;

}

printf("\nEnter Available Resources\t -- \t");

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

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

printf("\nEnter New Request Details -- ");

printf("\nEnter pid \t -- \t");

scanf("%d",&id);

printf("Enter Request for Resources \t -- \t");

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

{

scanf("%d",&newr);

f[id].all[i] += newr;

avail[i]=avail[i] - newr;

}

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

{

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

{

f[i].need[j]=f[i].max[j]-f[i].all[j];

if(f[i].need[j]<0)

f[i].need[j]=0;

}

}

cnt=0;

fl=0;

while(cnt!=n)

{

g=0;

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

{

if(f[j].flag==0)

{

b=0;

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

{

if(avail[p]>=f[j].need[p])

b=b+1;

else

b=b-1;

}

if(b==r)

{

printf("\nP%d is visited",j);

seq[fl++]=j;

f[j].flag=1;

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

avail[k]=avail[k]+f[j].all[k];

cnt=cnt+1;

printf("(");

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

printf("%3d",avail[k]);

printf(")");

g=1;

}

}

}

if(g==0)

{

printf("\n REQUEST NOT GRANTED -- DEADLOCK OCCURRED");

printf("\n SYSTEM IS IN UNSAFE STATE");

goto y;

}

}

printf("\nSYSTEM IS IN SAFE STATE");

printf("\nThe Safe Sequence is -- (");

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

printf("P%d ",seq[i]);

printf(")");

y: printf("\nProcess\t\tAllocation\t\tMax\t\t\tNeed\n");

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

{

printf("P%d\t",i);

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

printf("%6d",f[i].all[j]);

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

printf("%6d",f[i].max[j]);

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

printf("%6d",f[i].need[j]);

printf("\n");

}

getch();

}

INPUT

Enter number of processes – 5

Enter number of resources -- 3

Enter details for P0

Enter allocation -- 0 1 0

Enter Max -- 7 5 3

Enter details for P1

Enter allocation -- 2 0 0

Enter Max -- 3 2 2

Enter details for P2

Enter allocation -- 3 0 2

Enter Max -- 9 0 2

Enter details for P3

Enter allocation -- 2 1 1

Enter Max -- 2 2 2

Enter details for P4

Enter allocation -- 0 0 2

Enter Max -- 4 3 3

Enter Available Resources -- 3 3 2

Enter New Request Details --

Enter pid -- 1

Enter Request for Resources -- 1 0 2

OUTPUT

P1 is visited( 5 3 2)

P3 is visited( 7 4 3)

P4 is visited( 7 4 5)

P0 is visited( 7 5 5)

P2 is visited( 10 5 7)

SYSTEM IS IN SAFE STATE

The Safe Sequence is -- (P1 P3 P4 P0 P2 )

Process Allocation Max Need

P0 0 1 0 7 5 3 7 4 3

P1 3 0 2 3 2 2 0 2 0

P2 3 0 2 9 0 2 6 0 0

P3 2 1 1 2 2 2 0 1 1

P4 0 0 2 4 3 3 4 3 1

**Write a C program to simulate page replacement algorithms a) FIFO b) LRU c) LFU**

**PROGRAM**

**FIFO PAGE REPLACEMENT ALGORITHM**

#include<stdio.h>

#include<conio.h>

main()

{

int i, j, k, f, pf=0, count=0, rs[25], m[10], n;

clrscr();

printf("\n Enter the length of reference string -- ");

scanf("%d",&n);

printf("\n Enter the reference string -- ");

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

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

printf("\n Enter no. of frames -- ");

scanf("%d",&f);

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

m[i]=-1;

printf("\n The Page Replacement Process is -- \n");

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

{

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

{

if(m[k]==rs[i])

break;

}

if(k==f)

{

m[count++]=rs[i];

pf++;

}

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

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

if(k==f)

printf("\tPF No. %d",pf);

printf("\n");

if(count==f)

count=0;

}

printf("\n The number of Page Faults using FIFO are %d",pf);

getch();

}

INPUT

Enter the length of reference string – 20

Enter the reference string -- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter no. of frames -- 3

OUTPUT

The Page Replacement Process is –

7 -1 -1 PF No. 1

7 0 -1 PF No. 2

7 0 1 PF No. 3

2 0 1 PF No. 4

2 0 1

2 3 1 PF No. 5

2 3 0 PF No. 6

4 3 0 PF No. 7

4 2 0 PF No. 8

4 2 3 PF No. 9

0 2 3 PF No. 10

0 2 3

0 2 3

0 1 3 PF No. 11

0 1 2 PF No. 12

0 1 2

0 1 2

7 1 2 PF No. 13

7 0 2 PF No. 14

7 0 1 PF No. 15

The number of Page Faults using FIFO are 15

**LRU PAGE REPLACEMENT ALGORITHM**

#include<stdio.h>

#include<conio.h>

main()

{

int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0, next=1;

clrscr();

printf("Enter the length of reference string -- ");

scanf("%d",&n);

printf("Enter the reference string -- ");

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

{

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

flag[i]=0;

}

printf("Enter the number of frames -- ");

scanf("%d",&f);

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

{

count[i]=0;

m[i]=-1;

}

printf("\nThe Page Replacement process is -- \n");

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

{

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

{

if(m[j]==rs[i])

{

flag[i]=1;

count[j]=next;

next++;

}

}

if(flag[i]==0)

{

if(i<f)

{

m[i]=rs[i];

count[i]=next;

next++;

}

else

{

min=0;

for(j=1;j<f;j++)

if(count[min] > count[j])

min=j;

m[min]=rs[i];

count[min]=next;

next++;

}

pf++;

}

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

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

if(flag[i]==0)

printf("PF No. -- %d" , pf);

printf("\n");

}

printf("\nThe number of page faults using LRU are %d",pf);

getch();

}

INPUT

Enter the length of reference string -- 20

Enter the reference string -- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter the number of frames -- 3

OUTPUT

The Page Replacement process is --

7 -1 -1 PF No. -- 1

7 0 -1 PF No. -- 2

7 0 1 PF No. -- 3

2 0 1 PF No. -- 4

2 0 1

2 0 3 PF No. -- 5

2 0 3

4 0 3 PF No. -- 6

4 0 2 PF No. -- 7

4 3 2 PF No. -- 8

0 3 2 PF No. -- 9

0 3 2

0 3 2

1 3 2 PF No. -- 10

1 3 2

1 0 2 PF No. -- 11

1 0 2

1 0 7 PF No. -- 12

1 0 7

1 0 7

The number of page faults using LRU are 12

**LFU PAGE REPLACEMENT ALGORITHM**

#include<stdio.h>

#include<conio.h>

main()

{

int rs[50], i, j, k, m, f, cntr[20], a[20], min, pf=0;

clrscr();

printf("\nEnter number of page references -- ");

scanf("%d",&m);

printf("\nEnter the reference string -- ");

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

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

printf("\nEnter the available no. of frames -- ");

scanf("%d",&f);

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

{

cntr[i]=0;

a[i]=-1;

}

Printf(“\nThe Page Replacement Process is – \n“);

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

{

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

if(rs[i]==a[j])

{

cntr[j]++;

break;

}

if(j==f)

{

min = 0;

for(k=1;k<f;k++)

if(cntr[k]<cntr[min])

min=k;

a[min]=rs[i];

cntr[min]=1;

pf++;

}

printf("\n");

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

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

if(j==f)

printf(“\tPF No. %d”,pf);

}

printf("\n\n Total number of page faults -- %d",pf);

getch();

}

INPUT

Enter number of page references --10

Enter the reference string -- 1 2 3 4 5 2 5 2 5 1 4 3

Enter the available no. of frames – 3

OUTPUT

The Page Replacement Process is –

1 -1 -1 PF No. 1

1 2 -1 PF No. 2

1 2 3 PF No. 3

4 2 3 PF No. 4

5 2 3 PF No. 5

5 2 3

5 2 3

5 2 1 PF No. 6

5 2 4 PF No. 7

5 2 3 PF No. 8

Total number of page faults – 8

**AIM: To implement deadlock prevention technique**

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

void main()

{

char job[10][10];

int time[10],avail,tem[10],temp[10]; int safe[10];

int ind=1,i,j,q,n,t;

clrscr();

printf("Enter no of jobs: ");

scanf("%d",&n);

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

{

printf("Enter name and time: ");

scanf("%s%d",&job[i],&time[i]);

}

printf("Enter the available resources:");

scanf("%d",&avail);

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

{

temp[i]=time[i];

tem[i]=i;

}

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

for(j=i+1;j<n;j++)

{

if(temp[i]>temp[j])

{

t=temp[i];

temp[i]=temp[j];

temp[j]=t; t=tem[i];

tem[i]=tem[j];

tem[j]=t;

}

}

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

{

q=tem[i];

if(time[q]<=avail)

{

safe[ind]=tem[i];

avail=avail-tem[q];

printf("%s",job[safe[ind]]);

ind++;

}

else

{

printf("No safe sequence\n");

}

}

printf("Safe sequence is:");

for(i=1;i<ind; i++)

printf("%s %d\n",job[safe[i]],time[safe[i]]);

getch();

}

OUTPUT:

Enter no of jobs:4

Enter name and time: A 1

Enter name and time: B 4

Enter name and time: C 2

Enter name and time: D 3

Enter the available resources: 20

Safe sequence is: A 1, C 2, D 3, B 4.

**Simulate how parent and child processes use shared memory and address space.**

**PROGRAM**

Client program to demonstrate shared memory

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/shm.h>

#include<stdio.h>

#include<stdlib.h>

#define SHMSZ 27

main()

{

int shmid;

key\_t key;

char \*shm, \*s;

key=5678;

if((shmid=shmget(key,SHMSZ,0666))<0)

{

perror("shmget");

exit(1);

}

if((shm=shmat(shmid,NULL,0))==(char\*)-1)

{

perror("shmat");

exit(1);

}

for(s=shm;\*s!= NULL;s++)

putchar(\*s);

putchar('\n');

\*shm='\*';

exit(0);

}

SERVER:

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/shm.h>

#include<stdio.h>

#include<stdlib.h>

#define SHMSZ 27

main()

{

char c;

int shmid;

key\_t key;

char\*shm,\*s;

key=5678;

if((shmid=shmget(key,SHMSZ,IPC\_CREAT | 0666))<0)

{

perror("shmget");

exit(1);

}

if((shm=shmat(shmid,NULL,0))==(char\*)-1)

{

perror("shmat");

exit(1);

}

s=shm;

for(c='a';c<='z';c++)

\*s++=c;

s=NULL;

while(\*shm!='\*')

sleep(1);

exit(0);

}

8.4 INPUT AND OUTPUT

Sh client.sh

Sh server.sh

Abcdefghijklmnopqrstuvwxyz

**Simulate producer and consumer problem using threads (use java).**

**PROGRAM**

import java.io.\*;

import java.lang.\*;

class Q

{

int n;

boolean valueSet=false;

synchronized int get()

{

if(!valueSet)

try

{

wait();

}

catch(InterruptedException e){

System.out.println("InterruptedException caught");

}

System.out.println("Got:"+n);

valueSet=false;

notify();

return n;

}

synchronized void put(int n)

{

if(valueSet)

try

{

wait();

}catch(InterruptedException e){

System.out.println("InterruptedException caught"); }

this.n=n;

valueSet=true;

System.out.println("put:"+n);

notify(); }}

class Producer implements Runnable {

Q q;

Producer(Q q) {

this.q=q;

new Thread(this,"Producer").start(); }

public void run() {

int i=0;

while(true) {

q.put(i++); }}}

class Consumer implements Runnable {

Q q;

Consumer(Q q) {

this.q=q;

new Thread(this,"Consumer").start(); }

public void run() {

while(true) {

q.get(); }}}

class PCFixed {

public static void main(String args[]) {

Q q=new Q();

new Producer(q);

new Consumer(q);

System.out.println("Press ctrl+c to stop"); }}

11.4 INPUT AND OUTPUT

Put 1

Get1

Put 2

Get 2

Put 3

Get 3

.

.

.

.

.

.**Develop a code to convert virtual address to physical address**

**PROGRAM**

#include <stdio.h>

#include <conio.h>

struct pstruct

{

int fno;

int pbit;

}ptable[10];

int pmsize,lmsize,psize,frame,page,ftable[20],frameno;

void info()

{

printf("\n\nMEMORY MANAGEMENT USING PAGING\n\n");

printf("\n\nEnter the Size of Physical memory: ");

scanf("%d",&pmsize);

printf("\n\nEnter the size of Logical memory: ");

scanf("%d",&lmsize);

printf("\n\nEnter the partition size: ");

scanf("%d",&psize);

frame = (int) pmsize/psize;

page = (int) lmsize/psize;

printf("\nThe physical memory is divided into %d no.of frames\n",frame);

printf("\nThe Logical memory is divided into %d no.of pages",page);

}

void assign()

{

int i;

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

{

ptable[i].fno = -1;

ptable[i].pbit= -1;

}

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

ftable[i] = 32555;

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

{

printf("\n\nEnter the Frame number where page %d must be placed: ",i);

scanf("%d",&frameno);

ftable[frameno] = i;

if(ptable[i].pbit == -1)

{

ptable[i].fno = frameno;

ptable[i].pbit = 1;

}

}

getch();

printf("\n\nPAGE TABLE\n\n");

printf("PageAddress FrameNo. PresenceBit\n\n");

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

printf("%d\t\t%d\t\t%d\n",i,ptable[i].fno,ptable[i].pbit);

printf("\n\n\n\tFRAME TABLE\n\n");

printf("FrameAddress PageNo\n\n");

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

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

}

void cphyaddr()

{

int laddr,paddr,disp,phyaddr,baddr;

getch();

printf("\n\n\n\tProcess to create the Physical Address\n\n");

printf("\nEnter the Base Address: ");

scanf("%d",&baddr);

printf("\nEnter theLogical Address: ");

scanf("%d",&laddr);

paddr = laddr / psize;

disp = laddr % psize;

if(ptable[paddr].pbit == 1 )

phyaddr = baddr + (ptable[paddr].fno\*psize) + disp;

printf("\nThe Physical Address where the instruction present: %d",phyaddr);

}

void main()

{

clrscr();

info();

assign();

cphyaddr();

getch();

}

INPUT AND OUTPUT

MEMORY MANAGEMENT USING PAGING

Enter the Size of Physical memory: 16

Enter the size of Logical memory: 8

Enter the partition size: 2

The physical memory is divided into 8 no.of frames

The Logical memory is divided into 4 no.of pages

Enter the Frame number where page 0 must be placed: 5

Enter the Frame number where page 1 must be placed: 6

Enter the Frame number where page 2 must be placed: 7

Enter the Frame number where page 3 must be placed: 2

PAGE TABLE

PageAddress FrameNo. PresenceBit

0 5 1

1 6 1

2 7 1

3 2 1

FRAME TABLE

FrameAddress PageNo

0 32555

1 32555

2 3

3 32555

4 32555

5 0

6 1

7 2

Process to create the Physical Address

Enter the Base Address: 1000

Enter theLogical Address: 3

The Physical Address where the instruction present: 1013

**Simulate how operating system allocates frame to process.**

**PROGRAM**

#include<stdio.h>

#include<conio.h>

Void main()

{

int i,s[10],S,m,a[10],n,ch,total=0;

char p[10];

printf(“enter total no of process”);

scanf(“%d”,&n);

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

{

printf(“enter the process name %d”,i)

scanf(“%s”,&p[i])

printf(“enter the size of each process %d”,i)

scanf(“%d”,&s[i])

}

printf(“enter total no of frames”);

scanf(“%d”,&m);

printf(“enter ur choice”);

scanf(“%d”,&ch);

switch(ch)

{

case 1: printf(“Fixed Allocation”);

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

{

a[i]=m/n;

printf(“Allocation of frames to process %c is %d”,p[i],a[i]);

}

Case 2: printf(“Proportional Allocation”);

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

{

total+=s[i];

a[i]=(s[i]/total)\*m;

Printf(“Allocation of frames for process %c is %d”,p[i],a[i]);

}

default: printf(“invalid choice”);

}

}

INPUT & OUTPUT:

Enter total no of process: 4

Enter process Names: a

Enter the size of each process: 5

Enter process Names: b

Enter the size of each process: 6

Enter process Names: c

Enter the size of each process: 7

Enter process Names: d

Enter the size of each process: 8

Enter total No of frames: 40

Enter ur choice: 1

Fixed Allocation

Allocation of frames to process a is 10

Allocation of frames to process b is 10

Allocation of frames to process c is 10

Allocation of frames to process d is 10

Enter ur choice 2

Proportional Allocation

Allocation of frames for process a is 7

Allocation of frames for process b is 9

Allocation of frames for process c is 10

Allocation of frames for process d is 12

**Simulate the prediction of deadlock in operating system when all the processes announce their resource requirement in advance.**

**PROGRAM**

#include<stdio.h>

#include<conio.h>

void main()

{

int found,flag,l,p[4][5],tp,e[4][5],i,j,k=1,m[5],r[5],a[5],temp[5],sum=0,tr;

clrscr();

pritnf(“enter total no of process”);

scanf(“%d”,&tp);

pritnf(“enter total no of resources”);

scanf(“%d”,&tr);

printf(“enter claim matrix”);

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

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

{

scanf(“%d”,&c[i][j]);

}

printf(“enter allocation matrix”);

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

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

{

scanf(“%d”,&p[i][j]);

}

printf(“enter the resource vectors”);

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

scanf(“%d”,&r[i]);

printf(“enter the availability matrix”));

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

{

scanf(“%d”,&r[i]);

temp[i]=a[i];

}

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

{

sum=0;

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

{

sum+=p[i][j];

}

if(sum==0) {

m[

k]=i;

k++; }

for(i=1;i<=tp;i++); {

for(l=1;l

<k;l++)

if(i!=m[l]) {

flag=1;

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

if(c[i][j]>temp[j]) {

flag=0;

break; }}

if(flag==1) {

m[k]=i;

++;

for(j=1;j<=tr;j++)

temp[j]+=p[i][j]; }}

printf(“

\n deadlock causing process are”);

for(j=1;j<=tp;j++) {

found=0;

for(i=1;i<k;i++) {

if(j==m[i])

found=1; }

if(found==0)

printf(“%d”,j); }}

15.4 INPUT AND OUTPUT:

Enter total no of process 4

Enter total no of resources 5

Enter claim matrix

0 1 0 0 1

0 0 1 0 1

0 0 0 0 1

1 0 1 0 1

Enter resources vector: 2 1 1 2 1

Enter availability vector

0 0 0 0 1

Deadlock causing process will be 2