**PONDICHERRY UNIVERSITY**

Kalapet, Puducherry, 605014

**SCHOOL OF ENGINEERING AND TECHNOLOGY**

DEPARTMENT OF COMPUTER SCIENCE



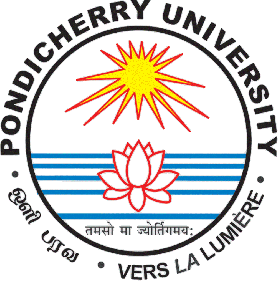
# CSSC 423-OPERATING SYSTEM LAB

**RECORD BOOK**

NAME : Pavithra G REGISTER NUMBER: 17384109

COURSE : Five Year Integrated M.sc Fourth Year

DEPARTMENT : COMPUTER SCIENCE SEMESTER : 8th



**DEPARTMENT OF COMPUTER SCIENCE**

**PONDICHERRY UNIVERSITY**

PUDUCHERRY-605014

**BONAFIDE CERTIFICATE**

Certified that this is a bonafide record of practical work done by

Ms. **Pavithra G** with register number **17384109** of eighth Semester Five Year Integrated Master of computer science program in the

subject of **CSSC 423 – OPERATING SYSTEM LAB** during the academic year 2020-2021.

In charge: **Dr. K. SURESH JOSEPH**

Date: **02/08/2021**

Submitted for M.Sc. Degree Practical examination held on **02/08/2021**

at the department of Computer Science, Pondicherry University, Puducherry.

Date: **02/08/2021**

Examiners:

**Internal Examiner External Examiner**

INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.No.** | **Date** | **Content** |  |
| 1 | 05/05/2021 | Average waiting time and average turnaround time |  |
| 2 | 26/05/2021 | File allocation statergies |  |
| 3 | 02/06/2021 | Paging technique |  |
| 4 | 09/06/2021 | File organisation techniques |  |
| 5 | 16/06/2021 | Banker’s algorithm |  |
| 6 | 23/06/2021 | Disk scheduling algorithm |  |
| 7 | 30/06/2021 | Page replacement algorithms |  |
| 8 | 07/07/2021 | Producer-customer problem |  |
| 9 | 07/07/2021 | Dining philosopher problem |  |

|  |  |
| --- | --- |
| **Ex.No:1** | **AVERAGE WAITING TIME AND AVERAGE TURNAROUND TIME** |
| **05/05/2021** |

# AIM:

To simulate the following non-pre-emptive CPU scheduling algorithms to find

turnaround time and waiting time.

1. FCFS
2. SJF
3. Round Robin (pre-emptive)
4. Priority

# PROGRAMING CODE:

1. **FCFS** #include<iostream> using namespace std;

// Function to find the waiting time for all

// processes

void findWaitingTime(int processes[], int n,

int bt[], int wt[])

{

// waiting time for first process is 0 wt[0] = 0;

// calculating waiting time for (int i = 1; i < n ; i++ )

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

}

// Function to calculate turn around time

void findTurnAroundTime( int processes[], int n, int bt[], int wt[], int tat[])

{

// calculating turnaround time by adding

// bt[i] + wt[i]

for (int i = 0; i < n ; i++) tat[i] = bt[i] + wt[i];

}

//Function to calculate average time

void findavgTime( int processes[], int n, int bt[])

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

//Function to find waiting time of all processes findWaitingTime(processes, n, bt, wt);

//Function to find turn around time for all processes findTurnAroundTime(processes, n, bt, wt, tat);

//Display processes along with all details cout << "Processes "<< " Burst time "

<< " Waiting time " << " Turn around time\n";

// Calculate total waiting time and total turn

// around time

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

{

total\_wt = total\_wt + wt[i]; total\_tat = total\_tat + tat[i];

cout << " " << i+1 << "\t\t" << bt[i] <<"\t "

<< wt[i] <<"\t\t " << tat[i] <<endl;

}

cout << "Average waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

// Driver code int main()

{

//process id's

int processes[] = { 1, 2, 3};

int n = sizeof processes / sizeof processes[0];

//Burst time of all processes int burst\_time[] = {10, 5, 8};

findavgTime(processes, n, burst\_time); return 0;

}

# SJF:

#include<iostream> using namespace std; int main()

{

int n,temp,tt=0,min,d,i,j;

float atat=0,awt=0,stat=0,swt=0;

cout<<"enter no of process"<<endl; cin>>n;

int a[n],b[n],e[n],tat[n],wt[n];

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

{

cout<<"enter arival time "; //input cin>>a[i];

}

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

{

cout<<"enter brust time "; //input cin>>b[i];

}

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

{

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

{

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

{

temp=a[i]; a[i]=a[j]; a[j]=temp;

temp=b[i];

b[i]=b[j]; b[j]=temp;

}

}

}

min=a[0]; for(i=0;i<n;i++)

{

if(min>a[i])

{

min=a[i]; d=i;

}

}

tt=min; e[d]=tt+b[d]; tt=e[d];

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

{

if(a[i]!=min)

{

e[i]=b[i]+tt; tt=e[i];

}

}

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

{

tat[i]=e[i]-a[i]; stat=stat+tat[i]; wt[i]=tat[i]-b[i]; swt=swt+wt[i];

}

atat=stat/n; awt=swt/n;

cout<<"Process Arrival-time(s) Burst-time(s) Waiting-time(s) Turnaround- time(s)\n";

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

{

cout<<"P"<<i+1<<" "<<a[i]<<" "<<b[i]<<"

"<<wt[i]<<" "<<tat[i]<<endl;

}

cout<<"awt="<<awt<<" atat="<<atat; //average waiting time and turn around time

}

1. **Round Robin:** #include<iostream> using namespace std;

// Function to find the waiting time for all

// processes

void findWaitingTime(int processes[], int n, int bt[], int wt[], int quantum)

{

// Make a copy of burst times bt[] to store remaining

// burst times. int rem\_bt[n];

for (int i = 0 ; i < n ; i++) rem\_bt[i] = bt[i];

int t = 0; // Current time

// Keep traversing processes in round robin manner

// until all of them are not done. while (1)

{

bool done = true;

// Traverse all processes one by one repeatedly for (int i = 0 ; i < n; i++)

{

// If burst time of a process is greater than 0

// then only need to process further if (rem\_bt[i] > 0)

{

done = false; // There is a pending process

if (rem\_bt[i] > quantum)

{

// Increase the value of t i.e. shows

// how much time a process has been processed t += quantum;

// Decrease the burst\_time of current process

// by quantum rem\_bt[i] -= quantum;

}

// If burst time is smaller than or equal to

// quantum. Last cycle for this process else

{

// Increase the value of t i.e. shows

// how much time a process has been processed t = t + rem\_bt[i];

// Waiting time is current time minus time

// used by this process wt[i] = t - bt[i];

// As the process gets fully executed

// make its remaining burst time = 0 rem\_bt[i] = 0;

}

}

}

// If all processes are done if (done == true)

break;

}

}

// Function to calculate turn around time

void findTurnAroundTime(int processes[], int n,

int bt[], int wt[], int tat[])

{

// calculating turnaround time by adding

// bt[i] + wt[i]

for (int i = 0; i < n ; i++) tat[i] = bt[i] + wt[i];

}

// Function to calculate average time

void findavgTime(int processes[], int n, int bt[],

int quantum)

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

// Function to find waiting time of all processes findWaitingTime(processes, n, bt, wt, quantum);

// Function to find turn around time for all processes findTurnAroundTime(processes, n, bt, wt, tat);

// Display processes along with all details cout << "Processes "<< " Burst time "

<< " Waiting time " << " Turn around time\n";

// Calculate total waiting time and total turn

// around time

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

{

total\_wt = total\_wt + wt[i]; total\_tat = total\_tat + tat[i];

cout << " " << i+1 << "\t\t" << bt[i] <<"\t "

<< wt[i] <<"\t\t " << tat[i] <<endl;

}

cout << "Average waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

// Driver code int main()

{

// process id's

int processes[] = { 1, 2, 3};

int n = sizeof processes / sizeof processes[0];

// Burst time of all processes int burst\_time[] = {10, 5, 8};

// Time quantum int quantum = 2;

findavgTime(processes, n, burst\_time, quantum);

return 0;

}

1. **Priority scheduling:** #include<bits/stdc++.h> using namespace std;

struct Process

{

int pid; // Process ID

int bt; // CPU Burst time required int priority; // Priority of this process

};

// Function to sort the Process acc. to priority bool comparison(Process a, Process b)

{

return (a.priority > b.priority);

}

// Function to find the waiting time for all

// processes

void findWaitingTime(Process proc[], int n, int wt[])

{

// waiting time for first process is 0 wt[0] = 0;

// calculating waiting time for (int i = 1; i < n ; i++ )

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

}

// Function to calculate turn around time

void findTurnAroundTime( Process proc[], int n, int wt[], int tat[])

{

// calculating turnaround time by adding

// bt[i] + wt[i]

for (int i = 0; i < n ; i++) tat[i] = proc[i].bt + wt[i];

}

//Function to calculate average time void findavgTime(Process proc[], int n)

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

//Function to find waiting time of all processes findWaitingTime(proc, n, wt);

//Function to find turn around time for all processes findTurnAroundTime(proc, n, wt, tat);

//Display processes along with all details cout << "\nProcesses "<< " Burst time "

<< " Waiting time " << " Turn around time\n";

// Calculate total waiting time and total turn

// around time

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

{

total\_wt = total\_wt + wt[i]; total\_tat = total\_tat + tat[i];

cout << " " << proc[i].pid << "\t\t"

<< proc[i].bt << "\t " << wt[i]

<< "\t\t " << tat[i] <<endl;

}

cout << "\nAverage waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

void priorityScheduling(Process proc[], int n)

{

// Sort processes by priority sort(proc, proc + n, comparison);

cout<< "Order in which processes gets executed \n"; for (int i = 0 ; i < n; i++)

cout << proc[i].pid <<" " ;

findavgTime(proc, n);

}

// Driver code int main()

{

Process proc[] = {{1, 10, 2}, {2, 5, 0}, {3, 8, 1}};

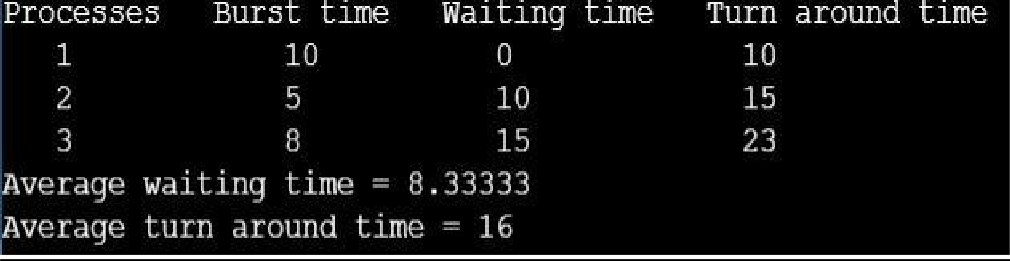
int n = sizeof proc / sizeof proc[0]; priorityScheduling(proc, n);

return 0;

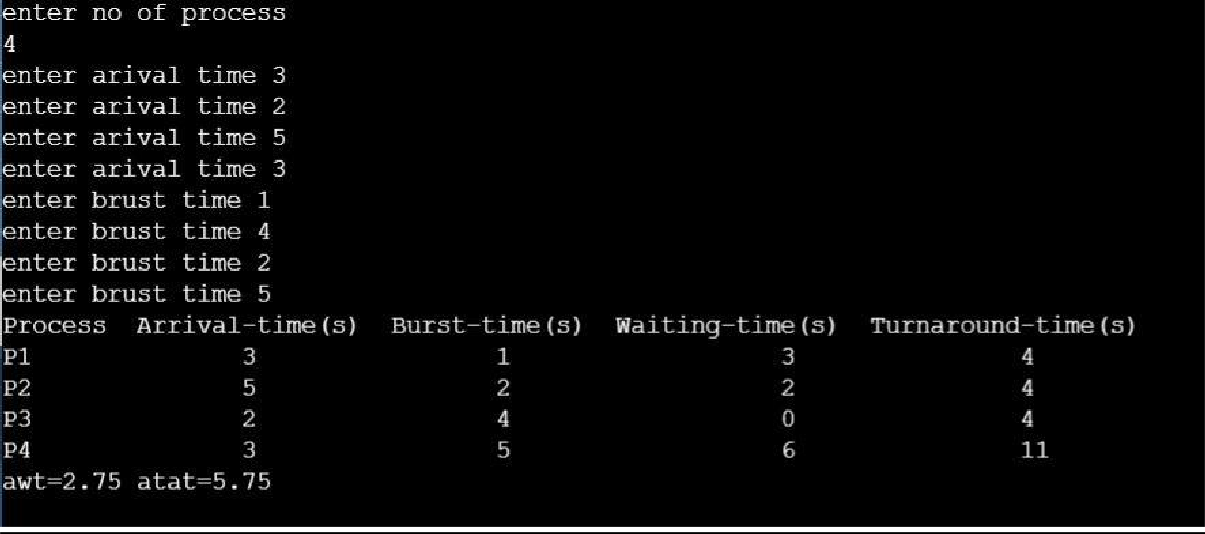
}

# OUTPUT:

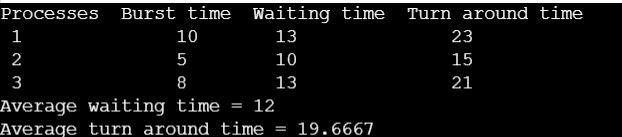
1. **FCFS**



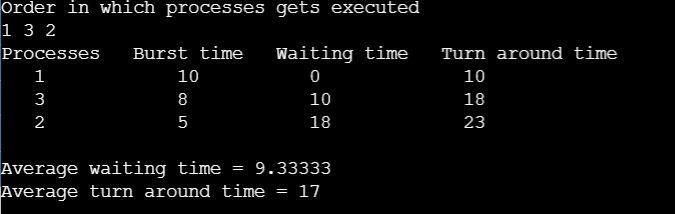
1. **SJF**



1. **Round Robin (pre-emptive)**



1. **Priority scheduling**



**RESULT:**

Program to find Turnaround time and Waiting time for different CPU scheduling algorithm has been executed and the output is verified.

|  |  |
| --- | --- |
| **Ex.No:2** | **FILE ALLOCATION STATERGIES** |
| **26/05/2021** |

# AIM:

To simulate the following file allocation stratergies.

* 1. Sequential
  2. Indexed
  3. Linked

# PROGRAMMING CODE:

1. **Sequential:**

#include <iostream> #include <conio.h> using namespace std;

void recurse(int files[]){

int flag = 0, startBlock, len, k;

cout << "Enter the starting block and the length of the files: "; cin >> startBlock >> len;

for (int j=startBlock; j<(startBlock+len); j++){ if (files[j] == 0)

flag++;

}

if(len == flag){

for (int k=startBlock; k<(startBlock+len); k++){ if (files[k] == 0){

files[k] = 1;

cout << k <<"\t" << files[k] << endl;

}

}

if (k != (startBlock+len-1))

cout << "The file is allocated to the disk" << endl;

}

else

cout << "The file is not allocated to the disk" << endl;

cout << "Do you want to enter more files?" << endl; int ch;

cout << "Press 1 for YES, 0 for NO: ";

cin >> ch; if (ch == 1)

recurse(files); else

exit(0); return;

}

int main()

{

int files[50];

for(int i=0;i<50;i++) files[i]=0;

cout << "Files Allocated are :" << endl;

recurse(files); getch(); return 0;

}

# Indexed:

#include <iostream> #include <conio.h> #include <stdlib.h>

using namespace std;

int files[50], indexBlock[50], indBlock, n; void recurse1();

void recurse2();

void recurse1(){

cout << "Enter the index block: "; cin >> indBlock;

if (files[indBlock] != 1){

cout << "Enter the number of blocks and the number of files needed for the index " << indBlock << " on the disk: ";

cin >> n;

}

else{

cout << indBlock << " is already allocated" << endl; recurse1();

}

recurse2();

}

void recurse2(){ int flag = 0;

for (int i=0; i<n; i++){ cin >> indexBlock[i];

if (files[indexBlock[i]] == 0) flag++;

}

if (flag == n){

for (int j=0; j<n; j++){ files[indexBlock[j]] = 1;

}

cout << "Allocated" << endl; cout << "File Indexed" << endl; for (int k=0; k<n; k++){

cout << indBlock << " > " << indexBlock[k] << ": " <<

files[indexBlock[k]] << endl;

}

}

else{

cout << "File in the index is already allocated" << endl; cout << "Enter another indexed file" << endl; recurse2();

}

cout << "Do you want to enter more files?" << endl; cout << "Enter 1 for Yes, Enter 0 for No: ";

int ch;

cin >> ch; if (ch == 1)

recurse1(); else

exit(0); return;

}

int main()

{

for(int i=0;i<50;i++) files[i]=0;

recurse1(); return 0;

}

# Linked:

#include <iostream> #include <conio.h> using namespace std;

void recursivePart(int pages[]){ int st, len;

cout << "Enter the index of the starting block and its length: "; cin >> st >> len;

int k = len;

if (pages[st] == 0){

for (int j = st; j < (st + k); j++){ if (pages[j] == 0){

pages[j] = 1;

cout << j << " >" << pages[j] << endl;

}

else {

cout << "The block "<< j << " is already allocated" << endl; k++;

}

}

}

else

cout <<"The block " << st << " is already allocated" << endl; cout << "Do you want to enter more files?" << endl;

cout << "Enter 1 for Yes, Enter 0 for No: "; int c;

cin >> c; if (c==1)

recursivePart(pages); else

exit(0);

return;

}

int main(){

int pages[50], p, a;

for (int i = 0; i < 50; i++) pages[i] = 0;

cout << "Enter the number of blocks already allocated: "; cin >> p;

cout << "Enter the blocks already allocated: "; for (int i = 0; i < p; i++){

cin >> a; pages[a] = 1;

}

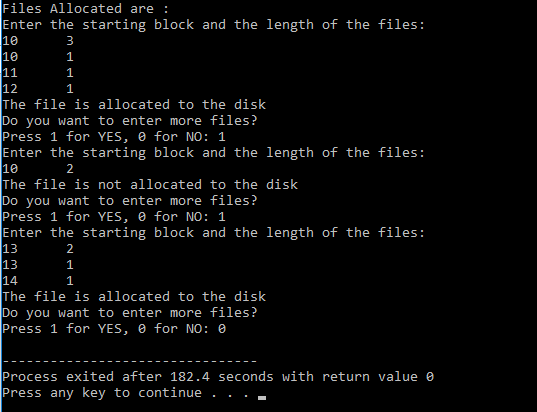
recursivePart(pages); getch();

return 0;

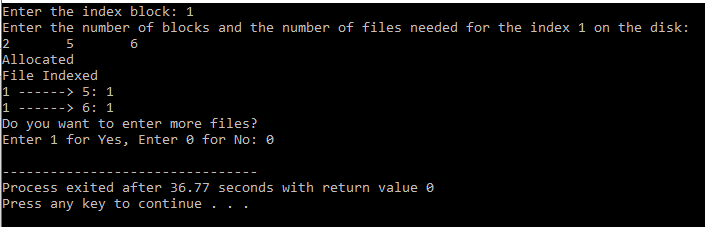
}

# OUTPUT:

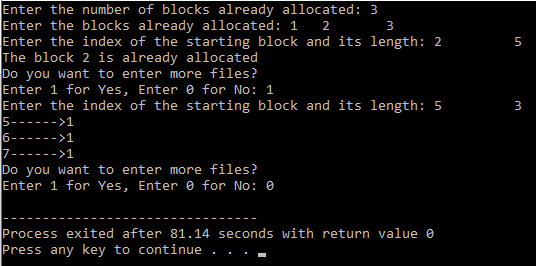
1. **Sequential:**



1. **Indexed:**



1. **Linked:**



**RESULT:**

The program to simulate different file allocation stratergies has been executed and generated the desired output.

|  |  |
| --- | --- |
| **Ex.No:3** | **PAGING TECHNIQUE** |
| **02/06/2021** |

# AIM:

To simulate paging technique of memory management.

# PROGRAMMING CODE:

#include<stdio.h> #define MAX 50 int main()

{

int page[MAX],i,n,f,ps,off,pno; int choice=0;

printf("\nEnter the no of peges in memory: "); scanf("%d",&n);

printf("\nEnter page size: "); scanf("%d",&ps); printf("\nEnter no of frames: "); scanf("%d",&f); for(i=0;i<n;i++)

page[i]=-1;

printf("\nEnter the page table\n");

printf("(Enter frame no as -1 if that page is not present in any frame)\n\n"); printf("\npageno\tframeno\n-------\t ");

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

{

printf("\n\n%d\t\t",i); scanf("%d",&page[i]);

}

do

{

printf("\n\nEnter the logical address(i.e,page no & offset):"); scanf("%d%d",&pno,&off);

if(page[pno]==-1)

printf("\n\nThe required page is not available in any of frames"); else

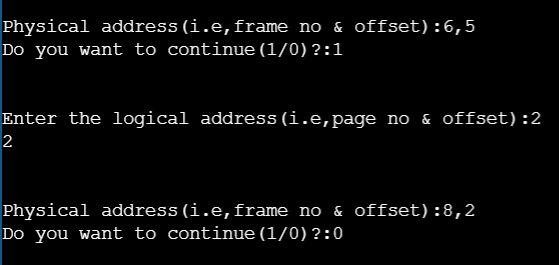
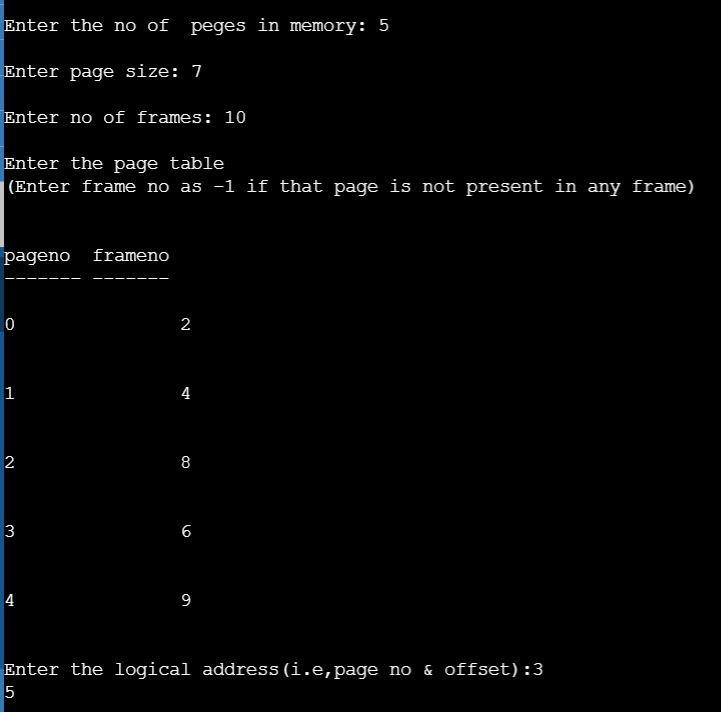
printf("\n\nPhysical address(i.e,frame no & offset):%d,%d",page[pno],off); printf("\nDo you want to continue(1/0)?:");

scanf("%d",&choice);

}while(choice==1); return 1;

}

# OUTPUT:



**RESULT:**

Program to simulate Paging technique of memory management has been executed and generated the desired output.

|  |  |
| --- | --- |
| **Ex.No:4** | **FILE ORGANISATION TECHNIQUES** |
| **09/06/2021** |

# AIM:

To simulate the following file organization techniques.

* 1. Single level directory
  2. Two level directory
  3. Hierarchical.

# PROGRAMMING CODE:

1. **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]; clrscr();

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 no - 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();

return 0;

}

# Two level directory:

#include<stdio.h> struct

{

char dname[10],fname[10][10]; int fcnt;

}dir[10]; void main()

{

int i,ch,dcnt,k; char f[30], d[30]; clrscr();

dcnt=0; while(1)

{

printf("\n\n 1. Create Directory\t 2. Create File\t 3. Delete File"); printf("\n 4. Search File \t \t 5. Display \t 6. Exit \t Enter your choice -- "); scanf("%d",&ch);

switch(ch)

{

case 1: printf("\n Enter name of directory -- "); scanf("%s", dir[dcnt].dname); dir[dcnt].fcnt=0;

dcnt++;

printf("Directory created"); break;

case 2: printf("\n Enter name of the directory -- "); scanf("%s",d);

for(i=0;i<dcnt;i++) if(strcmp(d,dir[i].dname)==0)

{

printf("Enter name of the file -- "); scanf("%s",dir[i].fname[dir[i].fcnt]); dir[i].fcnt++;

printf("File created"); break;

}

if(i==dcnt)

printf("Directory %s not found",d); break;

case 3: printf("\nEnter name of the directory -- "); scanf("%s",d);

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

{

if(strcmp(d,dir[i].dname)==0)

{

printf("Enter name of the file -- "); scanf("%s",f); for(k=0;k<dir[i].fcnt;k++)

{

if(strcmp(f, dir[i].fname[k])==0)

{

printf("File %s is deleted ",f); dir[i].fcnt--;

strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]); goto jmp;

}

}

printf("File %s not found",f); goto jmp;

}

}

printf("Directory %s not found",d); jmp : break;

case 4: printf("\nEnter name of the directory -- "); scanf("%s",d);

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

{

if(strcmp(d,dir[i].dname)==0)

{

printf("Enter the name of the file -- "); scanf("%s",f); for(k=0;k<dir[i].fcnt;k++)

{

if(strcmp(f, dir[i].fname[k])==0)

{

printf("File %s is found ",f); goto jmp1;

}

}

printf("File %s not found",f); goto jmp1;

}

}

printf("Directory %s not found",d); jmp1: break;

case 5: if(dcnt==0) printf("\nNo Directory's "); else

{

printf("\nDirectory\tFiles"); for(i=0;i<dcnt;i++)

{

printf("\n%s\t\t",dir[i].dname); for(k=0;k<dir[i].fcnt;k++) printf("\t%s",dir[i].fname[k]);

}

}

break; default:exit(0);

}

}

getch();

}

# Hierarchical:

#include<stdio.h> #include<graphics.h>

struct tree\_element

{

char name[20];

int x,y,ftype,lx,rx,nc,level; struct tree\_element \*link[5];

};

typedef struct tree\_element node; void main()

{

int gd=DETECT,gm; node \*root; root=NULL; clrscr();

create(&root,0,"root",0,639,320); clrscr(); initgraph(&gd,&gm,"c:\\tc\\BGI"); display(root);

getch(); closegraph();

}

create(node \*\*root,int lev,char \*dname,int lx,int rx,int x)

{

int i,gap; if(\*root==NULL)

{

(\*root)=(node \*)malloc(sizeof(node)); printf("Enter name of dir/file(under %s) :",dname); fflush(stdin);

gets((\*root)->name);

printf("enter 1 for Dir/2 forfile :"); scanf("%d",&(\*root)->ftype); (\*root)->level=lev;

(\*root)->y=50+lev\*50; (\*root)->x=x;

(\*root)->lx=lx; (\*root)->rx=rx; for(i=0;i<5;i++)

(\*root)->link[i]=NULL; if((\*root)->ftype==1)

{

printf("No of sub directories/files(for %s):",(\*root)->name); scanf("%d",&(\*root)-

>nc);

if((\*root)->nc==0) gap=rx-lx;

else gap=(rx-lx)/(\*root)->nc; for(i=0;i<(\*root)->nc;i++) create(&((\*root)->link[i]),lev+1,(\*root)-

>name,lx+gap\*i,lx+gap\*i+gap,lx+gap\*i+gap/2);

}

else (\*root)->nc=0;

}

}

display(node \*root)

{

int i; settextstyle(2,0,4); settextjustify(1,1); setfillstyle(1,BLUE);

setcolor(14); if(root!=NULL)

{

for(i=0;i<root->nc;i++)

{

line(root->x,root->y,root->link[i]->x,root->link[i]->y);

}

if(root->ftype==1) bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0); else fillellipse(root->x,root->y,20,20);

outtextxy(root->x,root->y,root->name); for(i=0;i<root->nc;i++)

{

display(root->link[i]);

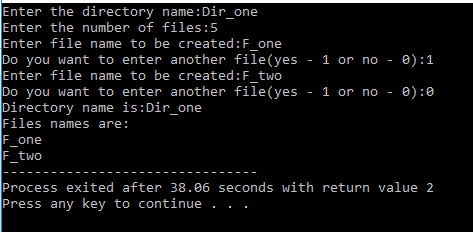
}

}

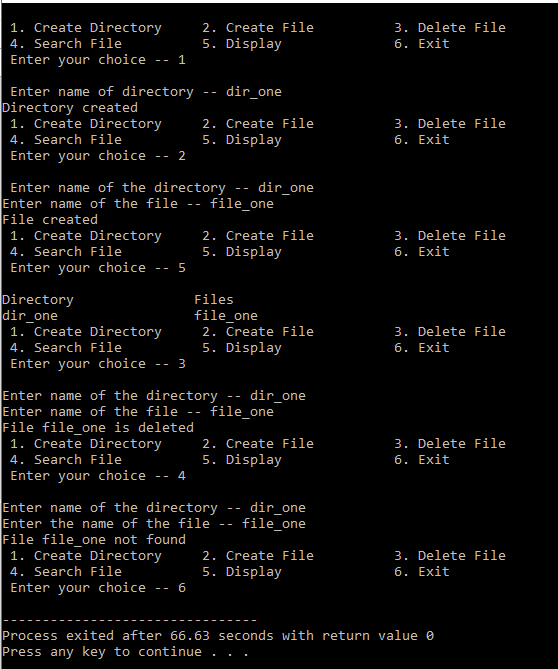
}

# OUTPUT:

1. **Single level directory:**



1. **Two level directory:**



1. **Hierarchical:**

Enter Name of dir/file (under root): ROOT Enter 1 for Dir / 2 For File : 1

No of subdirectories / files (for ROOT) :2 Enter Name of dir/file (under ROOT):USER 1 Enter 1 for Dir /2 for file:1

No of subdirectories /files (for USER 1):1

Enter Name of dir/file (under USER 1):SUBDIR Enter 1 for Dir /2 for file:1

No of subdirectories /files (for SUBDIR):2 Enter Name of dir/file (under USER 1): JAVA Enter 1 for Dir /2 for file:1

No of subdirectories /files (for JAVA): 0 Enter Name of dir/file (under SUBDIR):VB Enter 1 for Dir /2 for file:1

No of subdirectories /files (for VB): 0

Enter Name of dir/file (under ROOT):USER2 Enter 1 for Dir /2 for file:1

No of subdirectories /files (for USER2):2 Enter Name of dir/file (under ROOT):A Enter 1 for Dir /2 for file:2

Enter Name of dir/file (under USER2):SUBDIR 2 Enter 1 for Dir /2 for file:1

No of subdirectories /files (for SUBDIR 2):2

Enter Name of dir/file (under SUBDIR2):PPL Enter 1 for Dir /2 for file:1

No of subdirectories /files (for PPL):2 Enter Name of dir/file (under PPL):B Enter 1 for Dir /2 for file:2

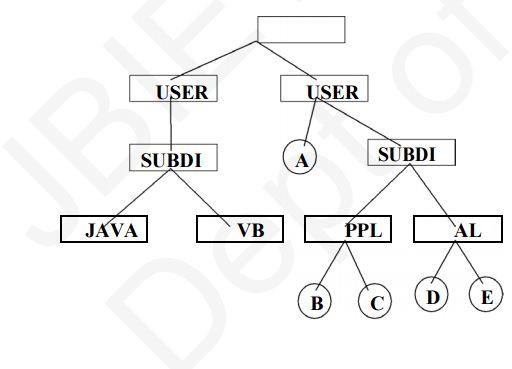
Enter Name of dir/file (under PPL):C Enter 1 for Dir /2 for file:2

Enter Name of dir/file (under SUBDIR):AI Enter 1 for Dir /2 for file:1

No of subdirectories /files (for AI): 2 Enter Name of dir/file (under AI):D Enter 1 for Dir /2 for file:2

Enter Name of dir/file (under AI):E

Enter 1 for Dir /2 for file:2



# RESULT:

Program to simulate different file organisation techniques has been executed and generated the desired outputs.

|  |  |
| --- | --- |
| **Ex.No:5** | **BANKER’S ALGORITHM** |
| **16/06/2021** |

# AIM:

To simulate Bankers algorithm for the purpose of deadlock avoidance.

# PROGRAMMING CODE:

#include<iostream> using namespace std;

// Number of processes const int P = 5;

// Number of resources const int R = 3;

// Function to find the need of each process int allot[P][R])

{

// Calculating Need of each P for (int i = 0 ; i < P ; i++)

for (int j = 0 ; j < R ; j++)

// Need of instance = maxm instance -

// allocated instance need[i][j] = maxm[i][j] - allot[i][j];

}

// Function to find the system is in safe state or not bool isSafe(int processes[], int avail[], int maxm[][R],

int allot[][R])

{

int need[P][R];

// Function to calculate need matrix calculateNeed(need, maxm, allot);

// Mark all processes as infinish bool finish[P] = {0};

// To store safe sequence int safeSeq[P];

// Make a copy of available resources int work[R];

for (int i = 0; i < R ; i++) work[i] = avail[i];

// While all processes are not finished

// or system is not in safe state. int count = 0;

while (count < P)

{

// Find a process which is not finish and

// whose needs can be satisfied with current

// work[] resources.

bool found = false;

for (int p = 0; p < P; p++)

{

// First check if a process is finished,

// if no, go for next condition if (finish[p] == 0)

{

// Check if for all resources of

// current P need is less

// than work int j;

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

if (need[p][j] > work[j]) break;

// If all needs of p were satisfied. if (j == R)

{

// Add the allocated resources of

// current P to the available/work

// resources i.e.free the resources for (int k = 0 ; k < R ; k++)

work[k] += allot[p][k];

// Add this process to safe sequence. safeSeq[count++] = p;

// Mark this p as finished

finish[p] = 1; found = true;

}

}

}

// If we could not find a next process in safe

// sequence.

if (found == false)

{

cout << "System is not in safe state"; return false;

}

}

// If system is in safe state then

// safe sequence will be as below

cout << "System is in safe state.\nSafe" " sequence is: ";

for (int i = 0; i < P ; i++) cout << safeSeq[i] << " ";

return true;

}

// Driver code int main()

{

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

// Available instances of resources int avail[] = {3, 3, 2};

// Maximum R that can be allocated

// to processes

int maxm[][R] = {{7, 5, 3},

{3, 2, 2},

{9, 0, 2},

{2, 2, 2},

{4, 3, 3}};

// Resources allocated to processes int allot[][R] = {{0, 1, 0},

{2, 0, 0},

{3, 0, 2},

{2, 1, 1},

{0, 0, 2}};

// Check system is in safe state or not isSafe(processes, avail, maxm, allot); return 0;

}

# OUTPUT:

Description: bankers

**RESULT:**

Program to simulate Bankers algorithm for the purpose of deadlock avoidance has been executed and generated the desired output.

|  |  |
| --- | --- |
| **Ex.No:6** | **DISK SCHEDULING ALGORITHM** |
| **23/06/2021** |

# AIM:

To simulate following disk scheduling algorithms

* 1. FCFS
  2. SCAN
  3. C-SCAN

# PROGRAMMING CODE:

1. **FCFS:**

#include <bits/stdc++.h> using namespace std;

int size = 8;

void FCFS(int arr[], int head)

{

int seek\_count = 0;

int distance, cur\_track;

for (int i = 0; i < size; i++) { cur\_track = arr[i];

// calculate absolute distance distance = abs(cur\_track - head);

// increase the total count seek\_count += distance;

// accessed track is now new head head = cur\_track;

}

cout << "Total number of seek operations = "

<< seek\_count << endl;

// Seek sequence would be the same

// as request array sequence

cout << "Seek Sequence is" << endl;

for (int i = 0; i < size; i++) { cout << arr[i] << endl;

}

}

// Driver code int main()

{

// request array

int arr[size] = { 176, 79, 34, 60, 92, 11, 41, 114 };

int head = 50;

FCFS(arr, head);

return 0;

}

1. **SCAN:** #include<bits/stdc++.h> using namespace std; int main(){

int i,j,k,n,m,sum=0,x,y,h; cout<<"Enter the size of disk\n"; cin>>m;

cout<<"Enter number of requests\n"; cin>>n;

cout<<"Enter the requests\n"; vector <int> a(n),b; for(i=0;i<n;i++){

cin>>a[i];

}

for(i=0;i<n;i++){ if(a[i]>m){

cout<<"Error, Unknown position "<<a[i]<<"\n"; return 0;

}

}

cout<<"Enter the head position\n"; cin>>h;

int temp=h; a.push\_back(h); a.push\_back(m); a.push\_back(0); sort(a.begin(),a.end());

for(i=0;i<a.size();i++){

if(h==a[i]) break;

}

k=i; if(k<n/2){

for(i=k;i<a.size();i++){

b.push\_back(a[i]);

}

for(i=k-1;i>=0;i--){

b.push\_back(a[i]);

}

}

else{

for(i=k;i>=0;i--){

b.push\_back(a[i]);

}

for(i=k+1;i<a.size();i++){

b.push\_back(a[i]);

}

}

temp=b[0]; cout<<temp; for(i=1;i<b.size();i++){

cout<<" -> "<<b[i]; sum+=abs(b[i]-temp); temp=b[i];

}

cout<<'\n';

cout<<"Total head movements = "<< sum<<'\n'; cout<<"Average head movement = "<<(float)sum/n<<'\n'; return 0;

}

1. **CSCAN:** #include<bits/stdc++.h> using namespace std; int main(){

int i,j,k,n,m,sum=0,x,y,h; cout<<"Enter the size of disk\n"; cin>>m;

cout<<"Enter number of requests\n"; cin>>n;

cout<<"Enter the requests\n"; vector <int> a(n),b; for(i=0;i<n;i++){

cin>>a[i];

}

for(i=0;i<n;i++){ if(a[i]>m){

cout<<"Error, Unknown position "<<a[i]<<"\n"; return 0;

}

}

cout<<"Enter the head position\n"; cin>>h;

int temp=h; a.push\_back(h);

a.push\_back(m); a.push\_back(0); sort(a.begin(),a.end());

for(i=0;i<a.size();i++){ if(h==a[i])

break;

}

k=i; if(k<n/2){

for(i=k;i<a.size();i++){

b.push\_back(a[i]);

}

for(i=0;i<=k-1;i++){ b.push\_back(a[i]);

}

}

else{

for(i=k;i>=0;i--){

b.push\_back(a[i]);

}

for(i=a.size()-1;i>=k+1;i--){ b.push\_back(a[i]);

}

}

temp=b[0]; cout<<temp; for(i=1;i<b.size();i++){

cout<<" -> "<<b[i];

sum+=abs(b[i]-temp); temp=b[i];

}

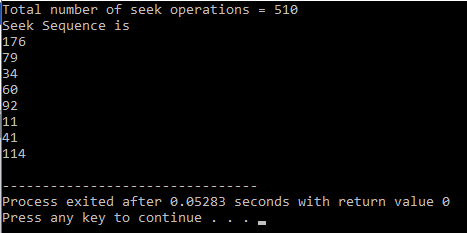
cout<<'\n';

cout<<"Total head movements = "<< sum<<'\n'; cout<<"Average head movement = "<<(float)sum/n<<'\n'; return 0;

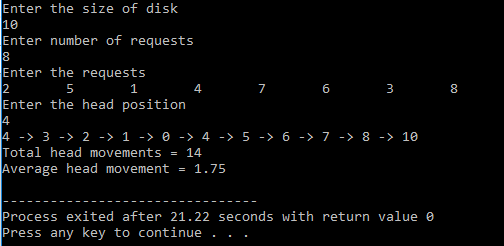
}

# OUTPUT:

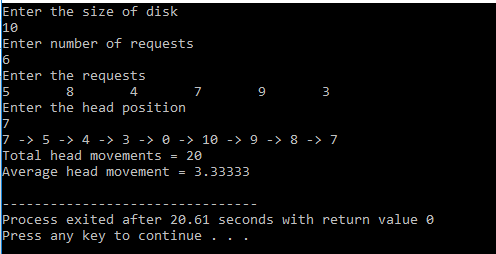
1. **FCFS:**



1. **SCAN:**



1. **CSCAN:**



**RESULT:**

Program to simulate different disk scheduling algorithms haa been executed and generated the desired output.

|  |  |
| --- | --- |
| **Ex.No:7** | **PAGE REPLACEMENT ALGORITHMS** |
| **30/06/2021** |

# AIM:

To simulate page replacement using

1. FIFO
2. LRU
3. LFU

# PROGRAMMING CODE:

* 1. **FIFO** #include<bits/stdc++.h> using namespace std;

int pageFaults(int pages[], int n, int capacity)

{

// To represent set of current pages. We use

// an unordered\_set so that we quickly check

// if a page is present in set or not unordered\_set<int> s;

// To store the pages in FIFO manner queue<int> indexes;

// Start from initial page int page\_faults = 0;

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

{

// Check if the set can hold more pages if (s.size() < capacity)

{

// Insert it into set if not present

// already which represents page fault if (s.find(pages[i])==s.end())

{

// Insert the current page into the set s.insert(pages[i]);

// increment page fault page\_faults++;

// Push the current page into the queue

indexes.push(pages[i]);

}

}

// If the set is full then need to perform FIFO

// i.e. remove the first page of the queue from

// set and queue both and insert the current page else

{

// Check if current page is not already

// present in the set

if (s.find(pages[i]) == s.end())

{

// Store the first page in the

// queue to be used to find and

// erase the page from the set int val = indexes.front();

// Pop the first page from the queue indexes.pop();

// Remove the indexes page from the set s.erase(val);

// insert the current page in the set s.insert(pages[i]);

// push the current page into

// the queue indexes.push(pages[i]);

// Increment page faults page\_faults++;

}

}

}

return page\_faults;

}

// Driver code int main()

{

int pages[] = {7, 0, 1, 2, 0, 3, 0, 4,

2, 3, 0, 3, 2};

int n = sizeof(pages)/sizeof(pages[0]); int capacity = 4;

cout <<"Total pageFaults are "<< pageFaults(pages, n, capacity); return 0;

}

# LRU

#include<bits/stdc++.h> using namespace std;

// Function to find page faults using indexes int pageFaults(int pages[], int n, int capacity)

{

// To represent set of current pages. We use

// an unordered\_set so that we quickly check

// if a page is present in set or not unordered\_set<int> s;

// To store least recently used indexes

// of pages.

unordered\_map<int, int> indexes;

// Start from initial page int page\_faults = 0;

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

{

// Check if the set can hold more pages if (s.size() < capacity)

{

// Insert it into set if not present

// already which represents page fault

if (s.find(pages[i])==s.end())

{

s.insert(pages[i]);

// increment page fault page\_faults++;

}

// Store the recently used index of

// each page indexes[pages[i]] = i;

}

// If the set is full then need to perform lru

// i.e. remove the least recently used page

// and insert the current page else

{

// Check if current page is not already

// present in the set

if (s.find(pages[i]) == s.end())

{

// Find the least recently used pages

// that is present in the set int lru = INT\_MAX, val;

for (auto it=s.begin(); it!=s.end(); it++)

{

if (indexes[\*it] < lru)

{

lru = indexes[\*it]; val = \*it;

}

}

// Remove the indexes page s.erase(val);

// insert the current page s.insert(pages[i]);

// Increment page faults page\_faults++;

}

// Update the current page index indexes[pages[i]] = i;

}

}

return page\_faults;

}

// Driver code int main()

{

int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

int n = sizeof(pages)/sizeof(pages[0]); int capacity = 4;

cout <<"Total PageFaults using LRU algorithm is "<<pageFaults(pages, n, capacity);

return 0;

}

* 1. **LFU** #include<bits/stdc++.h> using namespace std; int main(){

int n,m,i,j,k;

cout<<"Enter number of frames\n"; cin>>n;

cout<<"Enter number of processes\n"; cin>>m;

vector<int> p(m); cout<<"Enter processes\n"; for(i=0;i<m;i++){

cin>>p[i];

}

vector<vector<int>> a(n,vector<int>(m,-1)); map <int, int> mp,lfmp;

for(i=0;i<m;i++){ vector<int> op; vector<pair<int,int>> c,lf; for(auto q: mp){

c.push\_back({q.second,q.first});

}

for(auto q:lfmp){

lf.push\_back({q.second,q.first});

}

sort(lf.begin(),lf.end()); bool dontCall=true; if(lf.size()>2){

if(lf[0].first!=lf[1].first){ dontCall=false;

}

}

sort(c.begin(),c.end()); bool hasrun=false; for(j=0;j<n;j++){

if(a[j][i]==p[i]){ mp[p[i]]++;

lfmp[p[i]]++; hasrun=true; break;

}

if(a[j][i]==-1){ for(k=i;k<m;k++)

a[j][k]=p[i];

mp[p[i]]++;

lfmp[p[i]]++; hasrun=true; break;

}

}

if(j==n||hasrun==false){

for(j=0;j<n;j++){ if(dontCall==true){

int q;

if(lf[lf.size()-1].second==c[c.size()-1].second&&lf[lf.size()-1].first>1){ if(a[j][i]==c[c.size()-2].second){

mp.erase(a[j][i]);

lfmp.erase(a[j][i]); for(k=i;k<m;k++)

a[j][k]=p[i];

mp[p[i]]++;

lfmp[p[i]]++; break;

}

}

else{

if(a[j][i]==c[c.size()-1].second){ mp.erase(a[j][i]);

lfmp.erase(a[j][i]); for(k=i;k<m;k++)

a[j][k]=p[i];

mp[p[i]]++;

lfmp[p[i]]++; break;

}

}

}

else if(dontCall==false){

if(a[j][i]==lf[0].second){

mp.erase(a[j][i]);

lfmp.erase(a[j][i]); for(k=i;k<m;k++)

a[j][k]=p[i];

mp[p[i]]++;

lfmp[p[i]]++; break;

}

}

}

}

for(auto q:mp){ if(q.first!=p[i]){

mp[q.first]++;

}

}

}

int hit=0; vector<int> hitv(m); for(i=1;i<m;i++){

for(j=0;j<n;j++){ if(p[i]==a[j][i-1]){

hit++; hitv[i]=1; break;

}

}

}

cout<<"Process "; for(i=0;i<m;i++){

cout<<p[i]<<" ";

}

cout<<'\n'; for(i=0;i<n;i++){

cout<<"Frame "<<i<<" "; for(j=0;j<m;j++){

if(a[i][j]==-1) cout<<"E "; else

cout<<a[i][j]<<" ";

}

cout<<'\n';

}

cout<<"HIT "; for(i=0;i<hitv.size();i++){

if(hitv[i]==0) cout<<" ";

else cout<<hitv[i]<<" ";

}

cout<<"\n";

cout<<"Hit "<<hit<<'\n'<<"Page Fault "<<m-hit<<'\n'; return 0;

}

# OUTPUT:

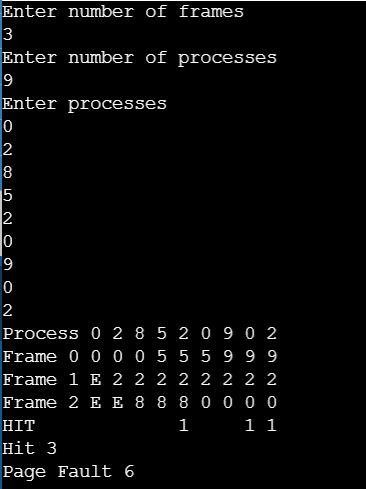
1. **FIFO:**

Description: FIFO pgflt

1. **LRU:**

Description: LRU

1. **LFU:**



**RESULT:**

Program to simulate different page replacement algorithms has been executed and generated the desired output.

|  |  |
| --- | --- |
| **Ex.No:8** | **PRODUCER-CUSTOMER PROBLEM** |
| **07/07/2021** |

# AIM:

To simulate Producer-Consumer problem using semaphores.

# PROGRAMMING CODE:

#include <stdio.h> #include <stdlib.h>

// Initialize a mutex to 1 int mutex = 1;

// Number of full slots as 0 int full = 0;

// Number of empty slots as size

// of buffer

int empty = 10, x = 0;

// Function to produce an item and

// add it to the buffer void producer()

{

// Decrease mutex value by 1

--mutex;

// Increase the number of full

// slots by 1

++full;

// Decrease the number of empty

// slots by 1

--empty;

// Item produced x++;

printf("\nProducer produces" "item %d",

x);

// Increase mutex value by 1

++mutex;

}

// Function to consume an item and

// remove it from buffer void consumer()

{

// Decrease mutex value by 1

--mutex;

// Decrease the number of full

// slots by 1

--full;

// Increase the number of empty

// slots by 1

++empty;

printf("\nConsumer consumes " "item %d",

x);

x--;

// Increase mutex value by 1

++mutex;

}

// Driver Code int main()

{

int n, i;

printf("\n1. Press 1 for Producer" "\n2. Press 2 for Consumer" "\n3. Press 3 for Exit");

// Using '#pragma omp parallel for'

// can give wrong value due to

// synchronisation issues.

// 'critical' specifies that code is

// executed by only one thread at a

// time i.e., only one thread enters

// the critical section at a given time

#pragma omp critical

for (i = 1; i > 0; i++) {

printf("\nEnter your choice:"); scanf("%d", &n);

// Switch Cases switch (n) { case 1:

// If mutex is 1 and empty

// is non-zero, then it is

// possible to produce if ((mutex == 1)

&& (empty != 0)) { producer();

}

// Otherwise, print buffer

// is full else {

printf("Buffer is full!");

}

break;

case 2:

// If mutex is 1 and full

// is non-zero, then it is

// possible to consume if ((mutex == 1)

&& (full != 0)) { consumer();

}

// Otherwise, print Buffer

// is empty else {

printf("Buffer is empty!");

}

break;

// Exit Condition case 3:

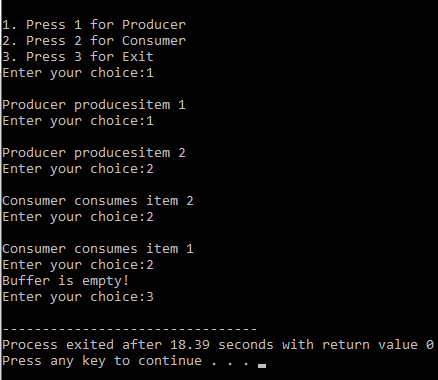
exit(0); break;

}

}

}

# OUTPUT:



**RESULT:**

Program to simulate Producer-Consumer problem has been executed and generated the desired output.

|  |  |
| --- | --- |
| **Ex.No:9** | **DINING PHILOSOPHER PROBLEM** |
| **07/07/2021** |

# AIM:

To implement Dining Philosopher problem.

# PROGRAMMING CODE:

#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);

}

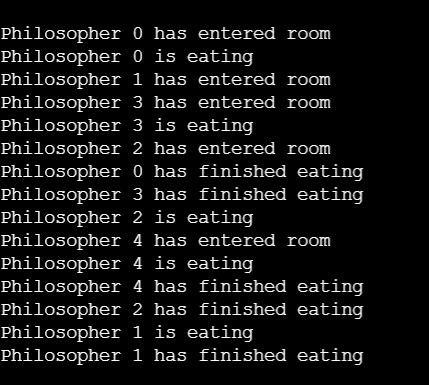
void eat(int phil)

{

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

}

# OUTPUT:



**RESULT:**

Program to simulate the concept of Dinning-Philosophers problem has been executed and generated desired output.