Operating System Lab Manual

Practical Record

SRI CHAITANYA
TECHNICAL CAMPUS

COLLEGE OF ENGINEERING & TECHNOLOGY
COLLEGE OF BUSINESS MANAGEMENT
(Approved by AICTE, NEW DELHI & Affiliated to JNTU, Hyderabad)

www.srichaitanyaengg.com
E-mail : director8a.sctc@gmail.com

Sheriguda (V), Ibrahimpatnam (M), R.R. Dist. - 501 510 - A.P.
Ph : 08414 - 223222, 223223 Fax : 08414 - 222678

MCA I Yr. - I Semester

SRI CHAITANYA
TECHNICAL CAMPUS

COLLEGE OF ENGINEERING & TECHNOLOGY
COLLEGE OF BUSINESS MANAGEMENT

(Approved by AICTE, NEW DELHI & Affiliated to JNTU, Hyderabad)

Sheriguda (V), Ibrahimpatnam (M), R.R. Dist. - 501 510 - A.P.

CERTIFICATE

This is to certify that Mr / Ms _____ has satisfactorily completed experiments in Operating System Lab laboratory as prescribed by Jawaharlal Nehru Technological University, Hyderabad.

Department Master of Computer Applications Roll No_____

Branch MCA Academic Year 2025-2026

INTERNAL EXAMINER HEAD OF THE DEPT.

EXTERNAL EXAMINER PRINCIPAL

INDEX

Sl.No.	Date	Name of the Experiment	Page No.	Remarks

OPERATING SYSTEMS LAB

MCA I Year I Sem.

L	T	P	C
0	0	3	1.5

Prerequisites:

- A course on "Programming for Problem Solving".
- A course on "Computer Organization and Architecture".

Co-requisite:

- A course on "Operating Systems".

Course Objectives:

- To provide an understanding of the design aspects of operating system concepts through simulation
- Introduce basic Unix commands, system call interface for process management, interprocess communication and I/O in Unix

Course Outcomes:

- Simulate and implement operating system concepts such as scheduling, deadlock management, file management and memory management.
- Able to implement C programs using Unix system calls

List of Experiments:

1. Write C programs to simulate the following CPU Scheduling algorithms
   a) FCFS b) SJF c) Round Robin d) priority

2. Write programs using the I/O system calls of UNIX/LINUX operating system
   (open, read, write, close, fcntl, seek, stat, opendir, readdir)

3. Write a C program to simulate Bankers Algorithm for Deadlock Avoidance and Prevention.

4. Write a C program to implement the Producer – Consumer problem using semaphores using UNIX/ LINUX system calls.

5. Write C programs to illustrate the following IPC mechanisms
   a) Pipes b) FIFOs c) Message Queues d) Shared Memory

6. Write C programs to simulate the following memory management techniques
   a) Paging b) Segmentation

7. Write C programs to simulate Page replacement policies
   a) FCFS b) LRU c) Optimal

TEXT BOOKS:

1. Operating System Principles- Abraham Silberchatz, Peter B. Galvin, Greg Gagne 7th Edition, John Wiley
2. Advanced programming in the Unix environment, W.R. Stevens, Pearson education.

REFERENCES:

1. Operating Systems – Internals and Design Principles, William Stallings, Fifth Edition-2005, Pearson Education/PHI
2. Operating System - A Design Approach-Crowley, TMH.
3. Modern Operating Systems, Andrew S Tanenbaum, 2nd edition, Pearson/PHI
4. UNIX Programming Environment, Kernighan and Pike, PHI/Pearson Education
5. UNIX Internals: The New Frontiers, U. Vahalia, Pearson Education

DEPARTMENT OF Masters of Computer Application

Vision & Mission

Vision

* To achieve high quality in technical education that provides the skills and attitude to adapt to the global needs of the Information Technology sector, through academic and research excellence.

Mission

* To equip the students with the cognizance for problem solving and to improve the teaching learning pedagogy by using innovative techniques.

* To strengthen the knowledge base of the faculty and students with motivation towards possession of effective academic skills and relevant research experience.

* To promote the necessary moral and ethical values among the engineers, for the betterment of the society.

Quality Policy

* Strives to inculcate the students with the world class Technical Knowledge, Entrepreneurial Competence and Social Ethics by providing continual improvement and innovation in the curriculum; based upon well-defined measurements and best practices.

* Develop faculty competencies, creativity, empowerment and accountability through faculty development programs and show strong management involvement and commitment.

OPERATING SYSTEMS LAB

PROGRAM 1:

Simulate the FCFS scheduling algorithm.

PROGRAM 2:

Simulate the SJF scheduling algorithm

PROGRAM 3:

Simulate the PRIORITY scheduling algorithm

PROGRAM 4:

Simulate the ROUND ROBIN scheduling algorithm

PROGRAM 5

Simulate FIFO page replacement algorithms

PROGRAM 6

Simulate LRU page replacement algorithms

PROGRAM 7

Simulate OPTIMAL page replacement algorithms

PROGRAM 8

Simulate SINGLE LEVEL DIRECTORY File Organization Technique.

PROGRAM 9

Simulate TWO LEVEL DIRECTORY File Organization Technique

PROGRAM 10

Simulate all file allocation strategies Sequential

REFERENCE BOOKS:

1. An Introduction to Operating Systems, P.C.P Bhatt, 2nd edition, PHI.
2. Modern Operating Systems, Andrew S Tanenbaum, 3rd Edition, PHI

OUTCOMES:

At the end of the course the students are able to:

- Ability to implement inter process communication between two processes.
- Ability to design and solve synchronization problems.
- Ability to simulate and implement operating system concepts such as scheduling, Deadlock management, file management, and memory management.

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to starting time), thosewho come after 5 minutes will not be allowed into the lab.

2. Plan your task properly much before to the commencement, come prepared to the lab withthe synopsis / program / experiment details.

3. Student should enter into the laboratory with:

- **a.** Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
- **b.** Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.
- **c.** Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer systemallotted to you by the faculty.

5. Execute your task in the laboratory, and record the results / output in the lab observationnote book, and get certified by the concerned faculty.

6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.

7. Computer labs are established with sophisticated and high end branded systems, which shouldbe utilized properly.

8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.

9. Students must take the permission of the faculty in case of any urgency to go out ; if anybody found loitering outside the lab / class without permission during working hours willbe treated seriously and punished appropriately.

10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Lab In-charge **Head of the Department** **PRINCIPAL**

PROGRAM 1

FIRST COME FIRST SERVE:

AIM: To write a c program to simulate the CPU scheduling algorithm First Come First Serve (FCFS)

DESCRIPTION:

To calculate the average waiting time using the FCFS algorithm first the waiting time of the first process is kept zero and the waiting time of the second process is the burst time of the first process and the waiting time of the third process is the sum of the burst times of the first and the second process and so on. After calculating all the waiting times the average waiting time is calculated as the average of all the waiting times. FCFS mainly says first come first serve the algorithm which came first will be served first.

ALGORITHM:

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process name and the burst time Step 4: Set the waiting of the first process as _0'and its burst time as its turnaround time Step 5: for each process in the Ready Q calculate
a). Waiting time (n) = waiting time (n-1) + Burst time (n-1) b).
Turnaround time (n)= waiting time(n)+Burst time(n)
Step 6: Calculate
a) Average waiting time = Total waiting Time / Number of process

b) Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

SOURCE CODE:

```
#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

PROGRAM 2

SHORTEST JOB FIRST:

AIM: To write a program to stimulate the CPU scheduling algorithm Shortest job first (Non- Preemption)

DESCRIPTION:

To calculate the average waiting time in the shortest job first algorithm the sorting of the process based on their burst time in ascending order then calculate the waiting time of each process as the sum of the bursting times of all the process previous or before to that process.

ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time.

Step 5: Set the waiting time of the first process as _0' and its turnaround time as its burst time.

Step 6: Sort the processes names based on their Burt time

Step 7: For each process in the ready queue, calculate

a) Waiting time(n)= waiting time (n-1) + Burst time (n-1)

b) Turnaround time (n)= waiting time(n)+Burst time(n)

Step 8: Calculate

c) Average waiting time = Total waiting Time / Number of process

d) Average Turnaround time = Total Turnaround Time / Number of process

Step 9: Stop the process

SOURCE CODE :

```
#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	TURNARO UND 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

PROGRAM 3

ROUND ROBIN:

AIM: To simulate the CPU scheduling algorithm round-robin.

DESCRIPTION:

To aim is to calculate the average waiting time. There will be a time slice, each process should be executed within that time-slice and if not it will go to the waiting state so first check whether the burst time is less than the time-slice. If it is less than it assign the waiting time to the sum of the total times. If it is greater than the burst-time then subtract the time slot from the actual burst time and increment it by time-slot and the loop continues until all the processes are completed.

ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue and time quantum (or) time slice

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Calculate the no. of time slices for each process where No. of time slice for process (n) = burst time process (n)/time slice

Step 5: If the burst time is less than the time slice then the no. of time slices =1.

Step 6: Consider the ready queue is a circular Q, calculate

a) Waiting time for process (n) = waiting time of process(n-1)+ burst time of process(n-1) + the time difference in getting the CPU fromprocess(n-1)

b) Turnaround time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).

Step 7: Calculate

c) Average waiting time = Total waiting Time / Number of process

d) Average Turnaround time = Total Turnaround Time / Number ofprocess Step

8: Stop the process

SOURCE CODE

```
#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:

PROCESS	BURST TIME	WAITING TIME	TURNAROUNDTIME
1	24	6	30
2	3	4	7
3	3	7	10

The Average Turnaround time is – 15.666667 The
Average Waiting time is ----------5.666667

PROGRAM 4
PRIORITY:

AIM: To write a c program to simulate the CPU scheduling priorityalgorithm.

DESCRIPTION:

To calculate the average waiting time in the priority algorithm, sort the burst times according to their priorities and then calculate the average waiting time of the processes. The waiting time of each process is obtained by summing up the burst times of all the previous processes.

ALGORITHM:

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Sort the ready queue according to the priority number.
Step 5: Set the waiting of the first process as _0' and its burst time as its turnaround time
Step 6: Arrange the processes based on process priority
Step 7: For each process in the Ready Q calculate Step 8:
for each process in the Ready Q calculate

a) Waiting time(n)= waiting time (n-1) + Burst time (n-1)
b) Turnaround time (n)= waiting time(n)+Burst time(n)

Step 9: Calculate

c) Average waiting time = Total waiting Time / Number of process
d) Average Turnaround time = Total Turnaround Time / Number of process Print the results in an order.
Step10: Stop

SOURCE CODE:

```
#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

VIVA QUESTIONS

1) Define the following
   a) Turnaround time b) Waiting time c) Burst time d) Arrival time
2) What is meant by process scheduling?
3) What are the various states of process?
4) What is the difference between preemptive and non-preemptive scheduling
5) What is meant by time slice?
6) What is round robin scheduling?

PROGRAM 5

FIFO PAGE REPLACEMENT ALGORITHMS

AIM: To implement FIFO page replacement technique.

DESCRIPTION:

Page replacement algorithms are an important part of virtual memory management and it helps the OS to decide which memory page can be moved out making space for the currently needed page. However, the ultimate objective of all page replacement algorithms is to reduce the number of page faults.

FIFO-This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

LRU-In this algorithm page will be replaced which is least recently used

OPTIMAL- In this algorithm, pages are replaced which would not be used for the longest duration of time in the future. This algorithm will give us less page fault when compared to other page replacement algorithms.

ALGORITHM:

1. Start the process
2. Read number of pages n
3. Read number of pages no
4. Read page numbers into an array a[i]
5. Initialize avail[i]=0 .to check page hit
6. Replace the page with circular queue, while re-placing check page availability in the frame Place avail[i]=1 if page is placed in theframe Count page faults
7. Print the results.
8. Stop the process.

FIRST IN FIRST OUT

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h> int fr[3];
void main()
{
void display();
int i,j,page[12]={2,3,2,1,5,2,4,5,3,2,5,2};
int
flag1=0,flag2=0,pf=0,frsize=3,top=0;
clrscr();
for(i=0;i<3;i++)
{
fr[i]=-1;
}
for(j=0;j<12;j++)
{
flag1=0; flag2=0; for(i=0;i<12;i++)
{
if(fr[i]==page[j])
{
flag1=1; flag2=1; break;
}
}
if(flag1==0)
{
for(i=0;i<frsize;i++)
{
if(fr[i]==-1)
{
fr[i]=page[j]; flag2=1; break;
}
}
}
if(flag2==0)
{
fr[top]=page[j];
top++;
pf++;
if(top>=frsize)
top=0;
}
display();
}
```

```
printf("Number of page faults : %d ",pf+frsize);
 getch();
}
void display()
{
int     i;     printf("\n");
for(i=0;i<3;i++)
printf("%d\t",fr[i]);
}
```

OUTPUT:

```
2 -1 -1
2  3 -1
2  3 -1
2  3  1
5  3  1
5  2  1
5  2  4
5  2  4
3  2  4
3  2  4
3  5  4
3  5  2
```

Number of page faults: 9

PROGRAM 6
LEAST RECENTLY USED

AIM: To implement LRU page replacement technique.

ALGORITHM:

1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. Declare counter and stack
6. Select the least recently used page by counter value
7. Stack them according the selection.
8. Display the values
9. Stop the process

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
int fr[3];
void main()
{
void display();
int p[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];
int index,k,l,flag1=0,flag2=0,pf=0,frsize=3;
 clrscr();
for(i=0;i<3;i++)
{
fr[i]=-1;
}
for(j=0;j<12;j++)
{
flag1=0,flag2=0;
for(i=0;i<3;i++)
{
if(fr[i]==p[j])
{
flag1=1;
flag2=1; break;
}
}
if(flag1==0)
```

```
{
for(i=0;i<3;i++)
{
if(fr[i]==-1)
{
fr[i]=p[j];       flag2=1;
break;
}
}
}
if(flag2==0)
{
for(i=0;i<3;i++)
fs[i]=0;
for(k=j-1,l=1;l<=frsize-1;l++,k--)
{
for(i=0;i<3;i++)
{
if(fr[i]==p[k]) fs[i]=1;
}}
for(i=0;i<3;i++)
{
if(fs[i]==0)
index=i;
}
fr[index]=p[j];
pf++;
}
display();
}
printf("\n no of page faults :%d",pf+frsize);
getch();
}
void display()
{
int i; printf("\n");
for(i=0;i<3;i++)
printf("\t%d",fr[i]);
}
```

OUTPUT:

2 -1 -1
2 3 -1
2 3 -1
2 3 1
2 5 1
2 5 1
2 5 4
2 5 4
3 5 4
3 5 2
3 5 2
3 5 2

No of page faults: 7

PROGRAM 7
OPTIMAL

AIM: To implement optimal page replacement technique.

ALGORTHIM:

1. Start Program
2. Read Number Of Pages And Frames
3.Read Each Page Value
4. Search For Page In The Frames
5.If Not Available Allocate Free Frame
6. If No Frames Is Free Repalce The Page With The Page That Is Leastly Used
7.Print Page Number Of Page Faults
8.Stop process.

SOURCE CODE:

```
/* Program to simulate optimal page replacement */
#include<stdio.h>
#include<conio.h>
int fr[3], n, m;
void
display();
void main()
{
 int i,j,page[20],fs[10];
 int
 max,found=0,lg[3],index,k,l,flag1=0,flag2=0,pf=0;
 float pr;
 clrscr();
 printf("Enter length of the reference string: ");
 scanf("%d",&n);
 printf("Enter the reference string: ");
 for(i=0;i<n;i++)
 scanf("%d",&page[i]);
 printf("Enter no of frames: ");
 scanf("%d",&m);
 for(i=0;i<m;i++)
 fr[i]=-1; pf=m;
```

```
for(j=0;j<n;j++)
 {
 flag1=0;          flag2=0;
 for(i=0;i<m;i++)
 {
 if(fr[i]==page[j])
 {
 flag1=1;  flag2=1;
 break;
 }
 }
 if(flag1==0)
 {
 for(i=0;i<m;i++)
 {
 if(fr[i]==-1)
 {
fr[i]=page[j];  flag2=1;
break;
 }
 }
 }
 if(flag2==0)
 {
 for(i=0;i<m;i++)
 lg[i]=0;
 for(i=0;i<m;i++)
 {
 for(k=j+1;k<=n;k++)
 {
 if(fr[i]==page[k])
 {
 lg[i]=k-j;
 break;
 }
 }
 }
 found=0;
 for(i=0;i<m;i++)
 {
 if(lg[i]==0)
 {
index=i;
found = 1;
```

```
 break;
 }
 }
if(found==0)
 {
 max=lg[0];    index=0;
 for(i=0;i<m;i++)
 {
 if(max<lg[i])
{
 max=lg[i];
 index=i;
}
 }
 }
 fr[index]=page[j];
 pf++;
}
display();
}
printf("Number of page faults : %d\n", pf);
pr=(float)pf/n*100;
printf("Page fault rate = %f \n", pr); getch();
}
void display()
{
 int  i;  for(i=0;i<m;i++)
 printf("%d\t",fr[i]);
 printf("\n");
}
```

OUTPUT:

```
Enter length of the reference string: 12
Enter the reference string: 1 2 3 4 1 2 5 1 2 3 4 5
Enter no of frames: 3
1 -1 -1
1 2 -1
1 2 3
1 2 4
1 2 4
1 2 4
1 2 5
1 2 5
1 2 5
3 2 5
4 2 5
4 2 5
Number of page faults : 7 Page fault rate = 58.333332
```

VIVA QUESTIONS

1) What is meant by page fault?
2) What is meant by paging?
3) What is page hit and page fault rate?
4) List the various page replacement algorithm
5) Which one is the best replacement algorithm?

PROGRAM 8

FILE ORGANIZATION TECHNIQUES

A) SINGLE LEVEL DIRECTORY:

AIM: Program to simulate Single level directory file organization technique.

DESCRIPTION:

The directory structure is the organization of files into a hierarchy of folders. In a single-level directory system, all the files are placed in one directory. There is a root directory which has all files. It has a simple architecture and there are no sub directories. Advantage of single level directory system is that it is easy to find a file in the directory.

SOURCE CODE :

```
#include<stdio.h>
struct
{
char    dname[10],fname[10][10];
int fcnt;
}dir;

void main()
{
int   i,ch;   char
f[30]; clrscr();
dir.fcnt = 0;
printf("\nEnter name of directory -- ");
scanf("%s", dir.dname);
while(1)
{
printf("\n\n1. Create File\t2. Delete File\t3. Search File \n
4. Display Files\t5. Exit\nEnter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\nEnter the name of the file -- ");
scanf("%s",dir.fname[dir.fcnt]);
dir.fcnt++; break;
case 2: printf("\nEnter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is deleted ",f); strcpy(dir.fname[i],dir.fname[dir.fcnt-1]); break;
}
```

```
        }
        if(i==dir.fcnt)
        printf("File %s not found",f);
                                    else
                                        dir.fcnt--;
                                        break;
            case 3:                     printf("\nEnter the name of the file -- ");
                                        scanf("%s",f);
                                        for(i=0;i<dir.fcnt;i++)
                                        {
                                        if(strcmp(f, dir.fname[i])==0)
                                        {
                                        printf("File %s is found ", f);
                                        break;
                                        }
                                        }
                                        if(i==dir.fcnt)
                                        printf("File %s not found",f);
                                        break;
            case 4:                     if(dir.fcnt==0)
                                        printf("\nDirectory Empty");
                                        else
                                        {
                                        printf("\nThe Files are -- ");
                                        for(i=0;i<dir.fcnt;i++)
                                        printf("\t%s",dir.fname[i]);
                                        }
                                        break;
            default: exit(0);
            }
}
getch();}
```

OUTPUT:

Enter name of directory -- CSE
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- A
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- B
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- C
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 4

The Files are -- A B C
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 3

Enter the name of the file – ABC File
ABC not found
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 2

Enter the name of the file – B
File B is deleted
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 5

PROGRAM 9
TWO LEVEL DIRECTORY

AIM: Program to simulate two level file organization technique

Description:
In the two-level directory system, each user has own user file directory (UFD). The system maintains a master block that has one entry for each user. This master block contains the addresses of the directory of the users. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. When a user refers to a particular file, only his own UFDis searched.

SOURCE CODE :

```
#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\n1. Create Directory\t2. Create File\t3. Delete File");
                printf("\n4. Search File\t\t5. Display\t6. Exit\t Enter your choice --");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: printf("\nEnter name of directory -- ");
                                scanf("%s",             dir[dcnt].dname);
                                dir[dcnt].fcnt=0;
                                dcnt++;
                                printf("Directory created"); break;
                        case 2: 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",dir[i].fname[dir[i].fcnt]);
```

```
                dir[i].fcnt++;
                 printf("File created");
                        }
                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++)
                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();
}
```

OUTPUT

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit
Enter your choice -- 1
Enter name of directory -- DIR1 Directory created

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 1
Enter name of directory -- DIR2 Directory created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 2
Enter name of the directory – DIR1
Enter name of the file -- A1
File created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit
Enter your choice -- 2
Enter name of the directory – DIR1

Enter name of the file -- A2
File created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6.
Exit Enter your choice – 6

VIVA QUESTIONS

1. Define directory?
2. List the different types of directory structures?
3. What is the advantage of hierarchical directory structure?
4. Which of the directory structures is efficient? Why?
5. What is acyclic graph directory?

PROGRAM 10

FILE ALLOCATION STRATEGIES

SEQUENTIAL:

AIM: To write a C program for implementing sequential file allocation method

DESCRIPTION:

The most common form of file structure is the sequential file in this type of file, a fixed format is used for records. All records (of the system) have the same length, consisting of the same number of fixed length fields in a particular order because the length and position of each field are known, only the values of fields need to be stored, thefield name and length for each field are attributes of the file structure.

ALGORITHM:

Step 1: Start the program.
Step 2: Get the number of files.
Step 3: Get the memory requirement of each file.
Step 4: Allocate the required locations to each in sequential order a).
Randomly select a location from availablelocation s1= random(100);

a) Check whether the required locations are free from the selected location.

```
 if(b[s1].flag==0){
for                    (j=s1;j<s1+p[i];j++){
if((b[j].flag)==0)count++;
      }
if(count==p[i]) break;
}
```

b) Allocate and set flag=1 to the allocated locations. for(s=s1;s<(s1+p[i]);s++)

```
{
k[i][j]=s; j=j+1; b[s].bno=s;
b[s].flag=1;
}
```

Step 5: Print the results file no, length, Blocks allocated. Step 6: Stop the program

SOURCE CODE :

```
#include<stdio.h>
main()
{
int f[50],i,st,j,len,c,k;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
X:
printf("\n Enter the starting block & length of file");
scanf("%d%d",&st,&len);
for(j=st;j<(st+len);j++)
if(f[j]==0)
{
f[j]=1
;
printf("\n%d->%d",j,f[j]);
}
else
{
printf("Block already allocated");
break;
}
if(j==(st+len))
printf("\n the file is allocated to disk");
printf("\n if u want to enter more files?(y-1/n-0)");
scanf("%d",&c);
if(c==1)
goto X;
else
exit();
getch();
}
```

OUTPUT:

Enter the starting block & length of file 4 10
4->1
5->1
6->1
7->1
8->1
9->1
10->1
11->1
12->1
13->1
The file is allocated to disk.

www.srichaitanyaengg.com
E-mail : director8a.sctc@gmail.com

SRI CHAITANYA TECHNICAL CAMPUS

COLLEGE OF ENGINEERING & TECHNOLOGY
COLLEGE OF BUSINESS MANAGEMENT

(Approved by AICTE, NEW DELHI & Affiliated to JNTU, Hyderabad)

Sheriguda (V), Ibrahimpatnam (M), R.R. Dist. - 501 510 - A.P.

Ph : 08414 - 223222, 223223 Fax : 08414 - 222678