CGB1232 – OPERATING SYSTEMS LABORATORY

COURSE OBJECTIVES:

To understand the services provided by an Operating System

COURSE OUTCOMES:

Upon completion of the course, the students will be able to :

CO NUMBER	COURSE OUTCOMES
CO1	Write Programs to evaluate various system calls
CO2	Analyze and evaluate the performance of various process management algorithms
CO3	Analyze and evaluate the performance of various memory management algorithms
CO4	Analyze and evaluate the performance of various I/O Management algorithms

LIST OF EXPERIMENTS

1. Linux Commands.
2. Process Management using System Calls.
3. Simulate the following CPU scheduling algorithms a) FIFO b) SJF c) Priority d) Round Robin
4. Implement dinning philosopher problem using semaphore.
5. Simulate Bankers Algorithm for Dead Lock Avoidance.
6. Simulate Paging Technique of memory management.
7. Simulate all page replacement algorithms a) FIFO b) LRU c) LFU
8. Simulate the following disk scheduling algorithms a. FCFS b. SSTF c. SCAN
9. Install any Guest operating system like Linux using VMware

TOTAL: 30 hours

TABLE OF CONTENTS

EXP.NO	DATE	NAMEOF THE EXPERIMENT	PAGE. NO	Signature
1		Linux Commands		
2		Process Management using System Calls: Fork, Exit, Getpid, Exit, Wait, Close, Stat		
3		Simulate the following CPU scheduling algorithms a) FIFO b) SJF c) Priority d) Round Robin		
4		Implement dinning philosopher problem using semaphore		
5		Simulate Bankers Algorithm for Dead Lock Avoidance		
6		Simulate Paging Technique of memory management		
7		Simulate all page replacement algorithms a) FIFO b) LRU c) LFU		
8		Simulate the following disk scheduling algorithms a. FCFS b. SSTF c. SCAN		
9		Install any Guest operating system like Linux using VMware		

EX.NO:1 **BASICS OF UNIX COMMANDS**

AIM:

To study and execute Unix commands.

GENERAL COMMANDS

Command	Function
Date	Used to display the current system date and time.
date +%D	Displays date only
date +%T	Displays time only
date +% Y	Displays the year part of date
date +% H	Displays the hour part of time
Cal	Calendar of the current month
Calyear	Displays calendar for all months of the specified year
calmonth year	Displays calendar for the specified month of the year
Who	Login details of all users such as their IP, Terminal No, User name,
who am i	Used to display the login details of the user
Tty	Used to display the terminal name
Uname	Displays the Operating System
uname –r	Shows version number of the OS (kernel).
uname –n	Displays domain name of the server
echo "txt"	Displays the given text on the screen
echo $HOME	Displays the user's home directory
Bc	Basic calculator. Press **Ctrl+d**to quit
Lpfile	Allows the user to spool a job along with others in a printqueue.
man cmdname	Manual for the given command. Press **q** to exit
History	To display the commands used by the user since log on.
Exit	Exit from a process. If shell is the only process then logs out

DIRECTORY COMMANDS

Command	Function
Pwd	Path of the present working directory
Mkdirdir	A directory is created in the given name under the current Directory
mkdirdir1 dir2	A number of sub-directories can be created under one stroke
cd subdir	Change Directory. If the subdirstarts with / then path startsfrom **root** (absolute) otherwise from current working directory.
Cd	To switch to the home directory.
cd /	To switch to the root directory.
cd..	To move back to the parent directory
Rmdirsubdir	Removes an empty sub-directory.

FILE COMMANDS

Command	Function
cat >filename	To create a file with some contents. To end typing press **Ctrl+d**. The >symbol means redirecting output to a file. (<for input)
cat filename	Displays the file contents.
cat >>filename	Used to append contents to a file
cpsrc des	Copy files to given location. If already exists, it will be Overwritten
cp –i src des	Warns the user prior to overwriting the destination file
cp –r src des	Copies the entire directory, all its sub-directories and files.
mv old new	To rename an existing file or directory. –i option can also be Used
mv f1 f2 f3 dir	To move a group of files to a directory.
mv –v old new	Display name of each file as it is moved.
Rmfile	Used to delete a file or group of files. –i option can also be used
rm *	To delete all the files in the directory.
rm –r *	Deletes all files and sub-directories
rm –f *	To forcibly remove even write-protected files
Ls	Lists all files and subdirectories (blue colored) in sorted manner.
Lsname	To check whether a file or directory exists.
lsname*****	Short-hand notation to list out filenames of a specific pattern.
ls –a	Lists all files including hidden files (files beginning with **.**)
ls –x dirname	To have specific listing of a directory.
ls –R	Recursive listing of all files in the subdirectories
ls –l	Long listing showing file access rights (read/write/execute-**rwx** for user/group/others-**ugo**).
cmpfile1 file2	Used to compare two files. Displays nothing if files are identical.
Wcfile	It produces a statistics of lines (**l**), words(**w**), and characters(**c**).
chmodperm file	Changes permission for the specified file. (r=4, w=2, x=1) chmod 740 file sets all rights for user, read only for groupsand no rights for others

OUTPUT

GENERAL COMMANDS

[student ~]date

Sat May 16 06:10:34 UTC 2020

[student~]date +%D

05/16/20

[student~]date +%T

10:13:11

[student~]date +%Y

2020

[student~]date +%H

10

[student~]cal

May 2020

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

[student~]cal 2020

2020

January

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

February

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

March

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

April

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

May

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

June

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

July

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

August

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

September

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

October

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3

November

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7

December

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5

4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

[student~]cal 2020

July 2020

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

[student~]who

studentpts/1 May 16 10:05 (172.16.1.14)

[student~]who am i

studentpts/1 May 16 10:05 (172.16.1.14)

[student~]tty

/dev/pts/1

[student~]uname

Linux

[student~]echo "hello"

hello

[student~]echo $HOME

/home/student

DIRECTORY COMMANDS

[student~]$ pwd

/home/student

[student~]mkdir san [student~]mkdir s1

s2[student~]ls

s1 s2 san

[student~]cd s1

[student s1]$ cd /

[student /]$ cd . .

[student /]$ rmdir s1

[student ~]$ ls

s2 san

FILE COMMANDS

[student ~]$ cat>test

hi welcome operating systems lab

[student ~]$ cat test

hi welcome operating systems lab

[student ~]$ cat>>test fourth semester[student ~]$ cat test

hi welcome operating systems lab fourth semester

[student ~]$ cat>test1

[student ~]$ cp test test1[student ~]$ cat

test1

hi welcome operating systems lab fourth semester

[student ~]$ cp -i test test1 cp: overwrite `test1'? y

[student ~]$ cp -r test test1

[student ~]$ ls

s s2 san swap.sh temp.sh test TEST test1**[student ~]$ mv san**

san1

[student ~]$ ls

s s2 san1 swap.sh temp.sh test TEST test1**[student ~]$ mv test**

test1 san1

[student ~]$ mv -v san1 sannew

`san1' -> `sannew'

[student ~]$ ls

s s2 sannew swap.sh temp.sh TEST **[student**

~]$ cmp test test1cmp: test: No such file or

directory

RESULT

Thus the study and execution of Unix commands has been completed successfully.

EX.NO.2: IMPLEMENTATION OF FORK, EXEC, GETPID, EXIT, WAIT, AND CLOSE SYSTEM CALLS.

AIM:

To write a program for implementing process management using the following system calls of UNIX operating system: fork, exec, getpid, exit, wait, close.

ALGORITHM:

1. Start the program.
2. Read the input from the command line.
3. Use fork() system call to create process, getppid() system call used to get theparent process ID and getpid() system call used to get the current process ID
4. execvp() system call used to execute that command given on that command lineargument
5. execlp() system call used to execute specified command.
6. Open the directory at specified in command line input.
7. Display the directory contents.
8. Stop the program.

PROGRAM:

```
#include<stdio.h> main(int
arc,char*ar[])
{
        int pid; char s[100]; pid=fork();if(pid<0)
                printf("error");else
        if(pid>0)
        {
                wait(NULL);
                printf("\n Parent  Process:\n"); printf("\n\tParent Process
                id:%d\t\n",getpid());execlp("cat","cat",ar[1],(char*)0);
                error("can't execute cat %s,",ar[1]);
        }
```

```
        else
        {
                printf("\nChild  process:");
                printf("\n\tChildprocess parent id:\t %d",getppid());
                printf(s,"\n\tChild process id :\t%d",getpid()); write(1,s,strlen(s));
                printf("  ");
                printf("  ");
                printf(" "); execvp(ar[2],&ar[2]);
                error("can't execute %s",ar[2]);
        }
}
```

OUTPUT:

```
[root@localhost ~]# ./a.out tst date Child process:
Child process id :
3137 Sat Apr 10 02:45:32 IST 2010
Parent Process:
Parent Process id:3136 sd
dsaASD[root@localhost ~]# cat tst sddsaASD
```

RESULT:

Thus the program for process management was written and successfully executed

EX.NO.3A: IMPLEMENTATION OF FCFS SCHEDULING ALGORITHM

AIM

To write a C program to implement First Come First Serve schedulingalgorithm.

ALGORITHM:

Step 1: Start the program.

Step 2: Get the input process and their burst time.

Step 3: Sort the processes based on order in which it requests CPU.

Step 4: Compute the waiting time and turnaround time for each process.

Step 5: Calculate the average waiting time and average turnaround time.

Step 6: Print the details about all the processes.

Step 7: Stop the program.

PROGRAM

PROGRAM:

```
#include<stdio.h>

Void main()
{
        int bt[50],wt[80],at[80],wat[30],ft[80],tat[80]; int i,n;
        float awt,att,sum=0,sum1=0;char
        p[10][5];
        printf("\nenter the number of process ");
        scanf("%d",&n);
        printf("\nEnter the process name and burst-time:");
        for(i=0;i<n;i++)scanf("%s%d",p[i],&bt[i]); printf("\nEnter the arrival-
        time:");for(i=0;i<n;i++)
                scanf("%d",&at[i]);
        wt[0]=0;
        for(i=1;i<=n;i++)
                wt[i]=wt[i-1]+bt[i-1];
        ft[0]=bt[0]; for(i=1;i<=n;i++)
                ft[i]=ft[i-1]+bt[i];
        printf("\n\n\t\t\tGANTT CHART\n");
        printf("\n \n");
        for(i=0;i<n;i++)
                printf("|\t%s\t",p[i]);
        printf("|\t\n");
        printf("\n \n");
        printf("\n");
        for(i=0;i<n;i++)
```

```
            printf("%d\t\t",wt[i]);
        printf("%d",wt[n]+bt[n]);
        printf("\n           \n");
        printf("\n");
        for(i=0;i<n;i++)
            wat[i]=wt[i]-at[i];
        for(i=0;i<n;i++)
            tat[i]=wat[i]-at[i];
        printf("\n FIRST COME FIRST SERVE\n");
        printf("\n Process Burst-time Arrival-time Waiting-time Finish-time Turnaround-time\n");
        for(i=0;i<n;i++)
            printf("\n\n      %d%s    \t    %d\t\t    %d    \t\t    %d\t\t    %d    \t\t
            %d",i+1,p[i],bt[i],at[i],wat[i],ft[i],tat[i]);
        for(i=0;i<n;i++)
            sum=sum+wat[i];awt=sum/n;
        for(i=0;i<n;i++)
            sum1=sum1+bt[i]+wt[i];
        att=sum1/n;
        printf("\n\nAverage waiting time:%f",awt);
        printf("\n\nAverage turnaround time:%f",att);
}
```

OUTPUT:

enter the number of process 3

Enter the process name and burst-time:p1 2

p2 3

p3 4

Enter the arrival-time:0 1 2

GANTT CHART

	p1		p2		p3	

0 2 5 9

FIRST COME FIRST SERVE

Process	Burst-time	Arrival-time	Waiting-time	Finish-time	Turnaround-time
p1	2	0	0	2	2
p2	3	1	1	5	4
p3	4	2	3	9	7

Average waiting time:1.333333 Average

turnaround time:5.333333

RESULT:

The FCFS scheduling algorithm has been implemented in C.

EX.NO.3B : IMPLEMENTATION OF SJF SCHEDULING ALGORITHM

AIM

To write a C program to implement shortest job first (non-pre-emptive) scheduling algorithm.

ALGORITHM:

Step 1: Start the program.

Step 2: Get the input process and their burst time.

Step 3: Sort the processes based on burst time.

Step 4: Compute the waiting time and turnaround time for each process.

Step 5: Calculate the average waiting time and average turnaround time.

Step 6: Print the details about all the processes.

Step 7: Stop the program.

PROGRAM:

```
#include<stdio.h>void
main()
{
    int i,j,n,bt[30],at[30],st[30],ft[30],wat[30],wt[30],temp,temp1,tot,tt[30];float awt, att;
    int p[15];
    wat[1]=0;
    printf("ENTER THE NO.OF PROCESS");
    scanf("%d",&n);
    printf("\nENTER THE PROCESS NUMBER,BURST TIME AND ARRIVALTIME");
    for(i=1;i<=n;i++)
    {
        scanf("%d\t %d\t %d",&p[i],&bt[i],&at[i]);
    }
    printf("\nPROCESS\tBURSTTIME\tARRIVALTIME");
    for(i=1;i<=n;i++)
    {
        printf("\np%d\t%d\t\t%d",p[i],bt[i],at[i]);

    }
    for(i=1;i<=n;i++)
    {
        for(j=i+1;j<=n;j++)
        {
            if(bt[i]>bt[j])
```

{

```
                    temp=bt[i];
                    bt[i]=bt[j];
                    bt[j]=temp;
                    temp1=p[i];
                    p[i]=p[j];
                    p[j]=temp1;
                }
            }
            if(i==1)
            {
            }
            else
            {
            st[1]=0;
                ft[1]=bt[1]; wt[1]=0;
                st[i]=ft[i-1];
                ft[i]=st[i]+bt[i];
                wt[i]=st[i];
            }
}
printf("\n\n\t\t\tGANTT CHART\n");
printf("\n\n");
for(i=1;i<=n;i++)
        printf("|\tp%d\t",p[i]); printf("|\t\n");
printf("\n\n");
printf("\n");
for(i=1;i<=n;i++)
        printf("%d  \t\t",wt[i]);
printf("%d",wt[n]+bt[n]);
printf("\n          \n");
for(i=2;i<=n;i++)
        wat[i]=wt[i]-at[i];
for(i=1;i<=n;i++)
         tt[i]=wat[i]+bt[i]-at[i];
printf("\nPROCESS\tBURSTTIME\tARRIVALTIME\tWAITINGTIME\tT  URNAROUNDTI  ME\n");
for(i=1;i<=n;i++)
{
                printf("\np%d %5d %15d %15d %15d",p[i],bt[i],at[i],wat[i],tt[i]);
}
```

```
        for(i=1,tot=0;i<=n;i++)tot+=wt[i];
        awt=(float)tot/n;
        printf("\n\n\n AVERAGE WAITING TIME=%f",awt);for(i=1,tot=0;i<=n;i++)
                tot+=tt[i];
        att=(float)tot/n;
        printf("\n\n AVERAGE TURNAROUND TIME=%f",att);
}
```

OUTPUT:

enter the no.of process3

enter the process number,burst time and arrival time1 8 1

2 5 1

3 3 1

PROCESS	BURSTTIME	ARRIVALTIME	WAITINGTIME	TURNAROUNDTIME
p3	3	1	0	2
p2	5	1	2	6
p1	8	1	7	14

AVERAGE WAITING TIME=3.666667 AVERAGE

TURNAROUND TIME=7.333333

RESULT:

The SJF scheduling algorithm has been implemented in C.

EX.NO.3C: IMPLEMENTATION OF ROUND ROBIN SCHEDULING ALGORITHM

AIM:

To write a C program to implement Round Robin scheduling algorithm.

ALGORITHM:

Step 1: Start the program.

Step 2: Get the input process and their burst time.Step

3: Sort the processes based on priority.

Step 4: Compute the waiting time and turnaround time for each process.

Step 5: Calculate the average waiting time and average turnaround time.

Step 6: Print the details about all the processes.

Step 7: Stop the program.

PROGRAM:

```
#include<stdio.h>
voidmain()
{
    int ct=0,y[30],j=0,bt[10],cwt=0; int
    tq,i,max=0,n,wt[10],t[10],at[10],tt[10],b[10]; float
    a=0.0,s=0.0;
    char p[10][10];
    printf("\n enter the no of process:");
    scanf("%d",&n);
    printf("\nenter the time quantum");scanf("%d",&tq);
    printf("\nenter the process name,bursttime,arrival time");
    for(i=0;i<n;i++)
    {
        scanf("%s",p[i]);
        scanf("%d",&bt[i]); scanf("%d",&at[i]);
        wt[i]=t[i]=0;b[i]=bt[i];
    }
    printf("\n\t\tGANTT CHART");
    printf("\n          \n");
    for(i=0;i<n;i++)
    {
        if(max<bt[i])
            max=bt[i];
```

```
}
```

```
while(max!=0)
{
        for(i=0;i<n;i++)
        {
                if(bt[i]>0)
                {
                        if(ct==0)
                                wt[i]=wt[i]+cwt;
                         else
                                wt[i]=wt[i]+(cwt-t[i]);
                }
                if(bt[i]==0)
                cwt=cwt+0;
                else if(bt[i]==max)
                {
                        if(bt[i]>tq)
                        {
                                cwt=cwt+tq;
```

```
                        bt[i]=bt[i]-tq;
                        max=max-tq;
                }
                else
                {
                        cwt=cwt+bt[i];
                        bt[i]=0; max=0;

                }
                printf("|\t%s",p[i]); y[j]=cwt;
                j++;
        }
else if(bt[i]<tq)
{
        cwt=cwt+bt[i]; bt[i]=0;
        printf("|\t%s",p[i]);
        y[j]=cwt;
        j++;
}
else if(bt[i]>tq)
{
        cwt=cwt+tq;
        bt[i]=bt[i]-tq;
        printf("|\t%s",p[i]);
         y[j]=cwt;
        j++;
}
else if(bt[i]==tq)
{
        cwt=cwt+bt[i];
```

```
                printf("|\t%s",p[i]); bt[i]=0;y[j]=cwt;
                j++;
            }
            t[i]=cwt;
            }
        ct=ct+1;
        }
        for(i=0;i<n;i++)
        {
            wt[i]=wt[i]-at[i];
            a=a+wt[i];
            tt[i]=wt[i]+b[i]-at[i];
            s=s+tt[i];
        }
        a=a/n; s=s/n;
        printf("\n            ");
        printf("\n0");
        for(i=0;i<j;i++)
            printf("\t%d",y[i]);
        printf("\n");
        printf("\n            "); printf("\n\t\t ROUND ROBIN\n");
        printf("\n        Process      Burst-time      Arrival-time      Waiting-time      Turnaround-
        time\n");
        for(i=0;i<n;i++)
            printf("\n\n %d%s \t %d\t\t %d \t\t %d\t\t %d", i+1, p[i], b[i], at[i],wt[i],  tt[i]);
        printf("\n\nAvg waiting time=%f",a);
        printf("\n\nAvgturn  around  time=%f",s);
}
```

OUTPUT:

enter the no of process:3 enter

the time quantum2

enter the process name, bursttime, arrival time

p1 2 0

p2 3 1

p3 4 2

GANTT CHART

| | p1| | p2| | p3| | p2| | p3 |

0 2 4 6 7 9

ROUND ROBIN

Process	Burst-time	Arrival-time	Waiting-time	Turnaround-time
p1	2	0	0	2
p2	3	1	3	5
p3	4	2	3	5

Avg Waiting Time=2.000000 Avg

Turnaround Time=4.000000

RESULT

The Round Robin scheduling algorithm has been implemented in C.

EX.NO.3D: IMPLEMENTATION OF PRIORITY SCHEDULING ALGORITHM

AIM

To write a C program to implement Priority Scheduling algorithm.

ALGORITHM:

Step 1: Start the program.

Step 2: Get the input process and their burst time.

Step 3: Sort the processes based on priority.

Step 4: Compute the waiting time and turnaround time for each process.

Step 5: Calculate the average waiting time and average turnaround time.

Step 6: Print the details about all the processes.

Step 7: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
 void main()
{
        int bt[30],pr[30],np;  intwt[30],tat[30],wat[30],at[30],ft[30];int  i,j,x,z,t;
        float sum1=0,sum=0,awt,att;char
        p[5][9],y[9];
        printf("\nenter the number of process");
        scanf("%d",&np);
        printf("\nEnter the process,burst-time and priority:");
         for(i=0;i<np;i++)
                 scanf("%s%d%d",p[i],&bt[i],&pr[i]);
        printf("\nEnter the arrival-time:"); for(i=0;i<np;i++)
                scanf("%d",&at[i]);
         for(i=0;i<np;i++)
                for(j=i+1;j<np;j++)
                {
                        if(pr[i]>pr[j])
                        {
                                x=pr[j]; pr[j]=pr[i];
                                pr[i]=x;
                                strcpy(y,p[j]);
                                strcpy(p[j],p[i]);
                                strcpy(p[i],y);
```

```
z=bt[j]; b t[j]=bt[i];
bt[i]=z;
```

```
                    }
                } wt[0]=0;
  for(i=1;i<=np;i++)
            wt[i]=wt[i-1]+bt[i-1];
ft[0]=bt[0];  for(i=1;i<np;i++)
             ft[i]=ft[i-1]+bt[i];
printf("\n\n\t\tGANTT CHART\n");printf("\n
                    \n");
for(i=0;i<np;i++)
            printf("|\t%s\t",p[i]);
printf("|\t\n");
printf("\n            \n");
printf("\n");
for(i=0;i<=np;i++)
            printf("%d\t\t",wt[i]);
printf("\n            \n");
printf("\n");
 for(i=0;i<np;i++)
            wat[i]=wt[i]-at[i];
 for(i=0;i<np;i++)
            tat[i]=wat[i]-at[i]; printf("\nPRIORITY
SCHEDULING:\n");
printf("\nProcess Priority Burst-time Arrival-time Waiting-time Turnaround-time");
for(i=0;i<np;i++)
            printf("\n\n%d%s\t%d\t\t%d\t\t%d\t%d\t\t%d",i+1,p[i],pr[i],bt[i],a  t[i],wt[i],tat[i]);
 for(i=0;i<np;i++)
            sum=sum+wat[i];
awt=sum/np; for(i=0;i<np;i++)
            sum1=sum1+tat[i];
 att=sum1/np;
printf("\n\nAverage waiting time:%f",awt); printf("\n\nAverageturn aroundtime is:%f",att);
}
```

OUTPUT:

```
Enter the number of process3
Enter the process, burst-time and priority:p1 3 3
p2 4 2
p3 5 1
Enter the arrival-time: 0 1 2
```

GANTT CHART

| p3 | p2 | p1 |

0 5 9 12

PRIORITY SCHEDULING:

Process	Priority	Burst-time	Arrival-time	Waiting-time	Turnaround-time
p3	1	5	0	0	0
p2	2	4	1	5	3
p1	3	3	2	9	5

Average waiting time: 3.666667 Average turnaround time is: 2.666667

RESULT

The Priority scheduling algorithm has been implemented in C.

EX.NO.4 DINING-PHILOSOPHERS PROBLEM.

AIM:

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

ALGORITHM:

Step 1: Start

Step 2: Define the number of philosophers

Step 3: Declare one thread per philosopher

Step 4: Declare one semaphore (represent chopsticks) per philosopher

Step 5: When a philosopher is hungry See if chopsticks on both sides are free Acquire both chopsticks Eat Restore the chopstick If chopsticks aren‟t free

Step 6: Wait till they are available

Step 7: Stop

PROGRAM

```
int tph, philname[20], status[20], howhung, hu[20], cho;main()
{
        int i;
        clrscr();
        printf("\n\nDINING PHILOSOPHER PROBLEM");
        printf("\nEnter the total no. of philosophers: ");scanf("%d",&tph);
        for(i=0;i<tph;i++)
        {
                philname[i] = (i+1);
                status[i]=1;
        }
        printf("How many are hungry : ");
         scanf("%d", &howhung);
        if(howhung==tph)
        {
                printf("\nAll are hungry..\nDead lock stage will occur");printf("\nExiting..");
        }       for(i=0;i<howhung;i++)
        else    {
        {
                        printf("Enter philosopher %d position: ",(i+1));scanf("%d", &hu[i]);
                        status[hu[i]]=2;
                }
```

```
do
{
printf("1.One can eat at a time\t2.Two can eat at atime\t3.Exit\nEnter your choice:");
scanf("%d", &cho);switch(cho)
{
        case 1:  one();
                 break;
        case 2:  two();
                 break;
           case 3: exit(0);
  default: printf("\nInvalid option..");
```

```
                }
                }while(1);
        }
}
one()
{
        int pos=0, x, i;
        printf("\nAllow one philosopher to eat at any time\n");for(i=0;i<howhung;
         i++, pos++)
        {
                printf("\nP %d is granted to eat", philname[hu[pos]]);for(x=pos;x<howhung;x++)
                        printf("\nP %d is waiting", philname[hu[x]]);
        }
}
two()
{
        int i, j, s=0, t, r, x;
        printf("\n Allow two philosophers to eat at same time\n");for(i=0;i<howhung;i++)
        {
                for(j=i+1;j<howhung;j++)
                {
                        if(abs(hu[i]-hu[j])>=1&&  abs(hu[i]-hu[j])!=4)
                        {
                                printf("\n\ncombination  %d \n", (s+1));t=hu[i];
                                r=hu[j];
                                 s++;
                                printf("\nP %d and P %d are granted to eat",philname[hu[i]],philname[hu[j]]);
```

```
                        for(x=0;x<howhung;x++)
                        {
                                if((hu[x]!=t)&&(hu[x]!=r))
                                    printf("\nP %d is waiting", philname[hu[x]]);
                        }
                    }
                }
            }
}
```

OUTPUT

DINING PHILOSOPHER PROBLEM

Enter the total no. of philosophers: 5How

many are hungry : 3

Enter philosopher 1 position: 2

Enter philosopher 2 position: 4

Enter philosopher 3 position: 5

1. One can eat at a time 2.Two can eat at a time 3.Exit

Enter your choice: 1

Allow one philosopher to eat at any time P 3 is granted to eatP 3 is waiting P

5 is waiting P 0 is waiting

P 5 is granted to eat P 5 is waiting P 0 is

waiting

P 0 is granted to eat P 0 is waiting

1.One can eat at a time 2.Two can eat at a time 3.Exit Enter your choice:

2 Allow two philosophers to eat at same time

combination 1

P 3 and P 5 are granted to eat P 0 is waiting

combination 2

P 3 and P 0 are granted to eat P 5 is waiting

combination 3

P 5 and P 0 are granted to eat P 3 is waiting

1.One can eat at a time 2.Two can eat at a time 3.Exit

Enter your choice: 3

RESULT:

Thus the program to implement the dining Philosopher was executed and verified.

EX.NO: 5 **DEADLOCK AVOIDANCE**

AIM:

To Simulate Algorithm for Deadlock avoidance

ALGORITHM:

Step 1: Start the Program

Step 2: Get the values of resources and processes.

Step 3: Get the avail value.

Step 4: After allocation find the need value.

Step 5: Check whether its possible to allocate. If possible it is safe state

Step 6: If the new request comes then check that the system is in safety or not if weallow the request.

Step 7: Stop the execution

PROGRAM:

```
 #include<stdio.h>
void main()
{
        int pno,rno,i,j,prc,count,t,total;
        count=0;
        clrscr();
         printf("\n Enter number  of process:");
         scanf("%d",&pno);
         printf("\n Enter  number  of resources:");
        scanf("%d",&rno);
         for(i=1;i< =pno;i++)
         {
                  flag[i]=0;
        }
         printf("\n  Enter  total  numbers  of  each  resources:");for(i=1;i<= rno;i++)
                 scanf("%d",&tres[i]);
         printf("\n  Enter  Max  resources  for  each  process:");


         for(i=1;i<=  pno;i++)
         {
                  printf("\n for process %d:",i);
                  for(j=1;j<= rno;j++)
                           scanf("%d",&max[i][j]);
```

```
}
printf("\n Enter allocated resources for each process:");for(i=1;i<=
```

```
pno;i++)
 {
        printf("\n for process %d:",i);
       for(j=1;j<= rno;j++)
              scanf("%d",&allocated[i][j]);
}
printf("\n available resources:\n");for(j=1;j<= rno;j++)
{
        avail[j]=0;
       total=0;
        for(i=1;i<= pno;i++)
       {
              total+=allocated[i][j];
        }
        avail[j]=tres[j]-total;
        work[j]=avail[j];
       printf("      %d \t",work[j]);
 }
 do
 {
        for(i=1;i<= pno;i++)
        {
              for(j=1;j<= rno;j++)
              {
                     need[i][j]=max[i][j]-allocated[i][j];
              }
        }
        printf("\n Allocated matrix              Max       need");
        for(i=1;i<= pno;i++)
        {
              printf("\n"); for(j=1;j<=
               rno;j++)
              {

               }
printf("|");  for(j=1;j<= rno;j++)
              {
```

```
printf("%4d",allocated[i][j]);



printf("%4d",max[i][j]);
```

```
                    printf("|"); for(j=1;j<=
                     rno;j++)
                    {


                }
            prc=0;
        printf("%4
        d",need[i][
        j]);
}
```

```
    for(i=1;i<= pno;i++)
  {
        if(flag[i]==0)
        {
            prc=i;
             for(j=1;j<= rno;j++)
            {
                if(work[j]< need[i][j])
                {
                    prc=0;
                    break;
                }
            }
        }
         if(prc!=0)
         break;
   }

   if(prc!=0)
  {
        printf("\n Process %d completed",i);count++;
         printf("\n Available matrix:");for(j=1;j<=
         rno;j++)
         {
            work[j]+=allocated[prc][j];
            allocated[prc][j]=0;
            max[prc][j]=0; flag[prc]=1;
             printf("    %d",work[j]);
         }
   }
  }while(count!=pno&&prc!=0);
if(count==pno)
  printf("\nThe system is in a safe state!!");
 else
  printf("\nThe system is in an unsafe state!!");
getch();
}
```

OUTPUT

Enter number of process:5 Enter

number of resources:3

Enter total numbers of each resources:10 5 7Enter Max

resources for each process:

for process 1: 7 5 3

for process 2: 3 2 2

for process 3: 9 0 2

for process 4: 2 2 2

for process 5: 4 3 3

Enter allocated resources for each process:for process 1:

0 1 0

for process 2: 3 0 2

for process 3: 3 0 2

for process 4: 2 1 1

for process 5: 0 0 2

available resources:

2 3 0

Allocated matrix			Max			need		
0	1	0	7	5	3	7	4	3
3	0	2	3	2	2	0	2	0
3	0	2	9	0	2	6	0	0
2	1	1	2	2	2	0	1	1
0	0	2	4	3	3	4	3	1

Process 2 completed

Available matrix: 5 3 2 Allocated matrix

Max need

0	1	0	7	5	3	7	4	3
0	0	0	0	0	0	0	0	0
3	0	2	9	0	2	6	0	0
2	1	1	2	2	2	0	1	1
0	0	2	4	3	3	4	3	1

Process 4 completed

Available matrix: 7 4 3 Allocated matrix

Max need

0 1 0| 7 5 3| 7 4 3

0 0 0| 0 0 0| 0 0 0

3 0 2| 9 0 2| 6 0 0

0 0 0| 0 0 0| 0 0 0

0 0 2| 4 3 3| 4 3 1

Process 1 completed

Available matrix: 7 5 3 Allocated matrix Max need

0 0 0| 0 0 0| 0 0 0

0 0 0| 0 0 0| 0 0 0

3 0 2| 9 0 2| 6 0 0

0 0 0| 0 0 0| 0 0 0

0 0 2| 4 3 3| 4 3 1

Process 3 completed

Available matrix: 10 5 5 Allocated

matrix Max need

0 0 0| 0 0 0| 0 0 0

0 0 0| 0 0 0| 0 0 0

0 0 0| 0 0 0| 0 0 0

0 0 0| 0 0 0| 0 0 0

0 0 2| 4 3 3| 4 3 1

Process 5 completed

Available matrix: 10 5 7

The system is in a safe state!!

RESULT:

Thus the program to implement the deadlock avoidance was executed and verified.

EX.NO:5B DEADLOCK DETECTION ALGORITHM

AIM:

To Simulate Algorithm for Deadlock detection

ALGORITHM:

Step 1: Start the Program

Step 2: Get the values of resources and processes.

Step 3: Get the avail value..

Step 4: After allocation find the need value.

Step 5: Check whether its possible to allocate.

Step 6: If it is possible then the system is in safe state.

Step 7: Stop the execution

PROGRAM

```
#include<stdio.h>
#include<conio.h> int
max[100][100]; i nt
alloc[100][100]; int
need[100][100]; int
avail[100];
int n,r;
void input();
void show();
void cal(); int
main()
{
        int i,j;
        printf("********** Deadlock Detection Algo **********\n"); input();show();
        cal();
        getch();
        return 0;
```

```
}

void input()
{
        int i,j;
        printf("Enter the no of Processes\t");scanf("%d",&n);
        printf("Enter the no of resource instances\t");scanf("%d",&r);
        printf("Enter the Max Matrix\n");for(i=0;i<n;i++)
        {
                for(j=0;j<r;j++)
                {
                        scanf("%d",&max[i][j]);
                }
        }
        printf("Enter the Allocation Matrix\n");
        for(i=0;i<n;i++)
        {
                for(j=0;j<r;j++)
                {
                        scanf("%d",&alloc[i][j]);
                }
        }
        printf("Enter the available Resources\n");
        for(j=0;j<r;j++)
        {
                scanf("%d",&avail[j]);
        }
}
void show()
{
        int i,j;
        printf("Process\t Allocation\t Max\t Available\t");
        for(i=0;i<n;i++)
        {
                printf("\nP%d\t ",i+1);
                for(j=0;j<r;j++)
                {
                        printf("%d ",alloc[i][j]);
                }
```

```
printf("\t");
```

```
                for(j=0;j<r;j++)
                {
                        printf("%d ",max[i][j]);
                }
                printf("\t");
                if(i==0)
                {
                        for(j=0;j<r;j++) printf("%d
                        ",avail[j]);
                }
        }
}
void cal()
{
        int finish[100],temp,need[100][100],flag=1,k,c1=0; int dead[100];int safe[100]; int
        i,j;
        for(i=0;i<n;i++)
        {
                finish[i]=0;
        }
        //find need matrix
```

```
for(i=0;i<n;i++)
{
        for(j=0;j<r;j++)
        {
                need[i][j]=max[i][j]-alloc[i][j];
        }
}
while(flag)
{
        flag=0;
        for(i=0;i<n;i++)
        {
                int c=0;
                for(j=0;j<r;j++)
                {
                        if((finish[i]==0)&&(need[i][j]<=avail[j]))
                        {
                                c++;
                                if(c==r)
                                {
                                        for(k=0;k<r;k++)
                                        {
                                                avail[k]+=alloc[i][j];  finish[i]=1;flag=1;
                                        }
                                        //printf("\nP%d",i);
                                         if(finish[i]==1)
                                        {
                                                i=n;
                                        }
                                }
                        }
```

```
                        }
                }
        }
        j=0;
        flag=0;
        for(i=0;i<n;i++)
        {
                if(finish[i]==0)
                {
                        dead[j]=i;
                         j++;
                        flag=1;
                }
        }
        if(flag==1)
        {
                printf("\n\nSystem is in Deadlock and the Deadlock process are\n");for(i=0;i<n;i++)
                {
                }

        }
        else
        {       printf("P%d\t",dead[i]);


        }
}
```

OUTPUT:

Enter the no. Of processes 3

Enter the no of resources instances 3Enter the

max matrix

3 6 8

4 3 3

3 4 4

Enter the allocation matrix3 3 3

2 0 3

1 2 4

Enter the available resources1 2 0

Process	allocation	max	available
P1	3 3 3	3 6 8	1 2 0
P2	2 0 3	4 3 3	
P3	1 2 4	3 4 4	

System is in deadlock and deadlock process areP1 P2 P3

RESULT:

Thus the program to implement the deadlock detection was executed successfully.

EX.NO:6 IMPLEMENTATION OF PAGING TECHNIQUE OF MEMORY MANAGEMENT

AIM:

To write a C program to implement paging concept for memory management.

ALGORIHTM:

Step 1: Start the program.

Step 2: Enter the logical memory address.

Step 3: Enter the page table which has offset and page frame.

Step 4: The corresponding physical address can be calculate by, PA = [pageframe* No. of pagesize] + Page offset.

Step 5: Print the physical address for the corresponding logical address.

Step 6:Terminate the program.

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();
}
```

OUTPUT

```
Enter the memory size – 1000Enter
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

Memory is Full

Enter Logical Address to find Physical Address Enter process no. and page number and offset--- 2

3

60

The Physical Address is ---- 760

RESULT:

Thus C program for implementing paging concept for memory management has been executed successfully.

EX.NO:7A IMPLEMENTATION OF THE FIFO PAGE REPLACEMENT ALGORITHMS

AIM:

To write a C program to implement FIFO page replacement algorithm.

ALGORITHM:

1. Start the process
2. Declare the size with respect to page length
3. Check the need of replacement from the page to memory
4. Check the need of replacement from old page to new page in memory
5. Format queue to hold all pages
6. Insert the page require memory into the queue
7. Check for bad replacement and page fault
8. Get the number of processes to be inserted
9. Display the values
10. Stop the process

PROGRAM:

```
#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();
 }
```

OUTPUT

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

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

RESULT:

Thus a C program to implement FIFO page replacement has been executed successfully.

EX.NO:7B IMPLEMENTATION OF LRU PAGE REPLACEMENT ALGORITHM

AIM:

To write C program to implement LRU page replacement algorithm

.

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

PROGRAM:

```
#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();
}
```

OUTPUT

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

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

RESULT:

Thus LRU page replacement algorithm was implemented in C program and was executed successfully.

Ex.NO: 7.C **OPTIMAL(LFU) PAGE REPLACEMENT ALGORITHM**

AIM: To implement Optimal Page replacement algorithm. (The page which is not used for longest time)

ALGORITHM:

Optimal algorithm
Here we select the page that will not be used for the longest period of time.

OPTIMAL:
Step 1: Create a array
Step 2: When the page fault occurs replace page that will not be used for the longest period of time

/*OPTIMAL(LFU) page replacement algorithm*/

Program:

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1; int
recent[10],optcal[50],count=0;
int optvictim();
void main()
{
  clrscr();
  printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHN");
  printf("\n.....");
  printf("\nEnter the no.of frames");
  scanf("%d",&nof);
  printf("Enter the no.of reference string"); scanf("%d",&nor);
  printf("Enter the reference string");
  for(i=0;i<nor;i++)
    scanf("%d",&ref[i]);
  clrscr();
  printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHM");
  printf("\n.....");
  printf("\nThe given string");
  printf("\n..... \n");
  for(i=0;i<nor;i++)
    printf("%4d",ref[i]);
  for(i=0;i<nof;i++)
  {
```

```
        frm[i]=-1;
        optcal[i]=0;
    }
    for(i=0;i<10;i++)
        recent[i]=0;
    printf("\n");
    for(i=0;i<nor;i++)
    {
      flag=0;
      printf("\n\tref no %d ->\t",ref[i]);
      for(j=0;j<nof;j++)
      {
            if(frm[j]==ref[i])
            {
              flag=1; break;
            }
      }
      if(flag==0)
      {
            count++;
            if(count<=nof)
                victim++;
            else
                victim=optvictim(i);
            pf++; frm[victim]=ref[i];
            for(j=0;j<nof;j++)
              printf("%4d",frm[j]);
      }
    }
    printf("\n Number of page faults: %d",pf);
    getch();
}
int optvictim(int index)
{
    int i,j,temp,notfound;
    for(i=0;i<nof;i++)
    {
      notfound=1;
      for(j=index;j<nor;j++)
            if(frm[i]==ref[j])
            {
                notfound=0;
                optcal[i]=j;
                break;
            }
      if(notfound==1)
              return i;
    }
```

```
  temp=optcal[0];
  for(i=1;i<nof;i++)
     if(temp<optcal[i])
          temp=optcal[i];
  for(i=0;i<nof;i++)
     if(frm[temp]==frm[i])
          return i;
 return 0;
}
```

OUTPUT:

```
                    OPTIMAL PAGE REPLACEMENT ALGORITHM

Enter no.of Frames ..... 3
 Enter no.of reference string  6

 Enter reference string.. 6
5 4 2 3 1


            OPTIMAL PAGE REPLACEMENT ALGORITHM
        The given reference string:
          ……..  6   5   4   2   3   1

          Reference NO 6->      6  -1    -1
          Reference NO 5->      6   5    -1
          Reference NO 4->      6   5     4
          Reference NO 2->      2   5     4
          Reference NO 3->      2   3     4
          Reference NO 1->      2   3     1

          No.of page faults...6
```

RESULT:

Thus Optimal page replacement algorithm was implemented in C program and was executed successfully.

EX.NO: 8 DISK SCHEDULING ALGORITHMS

AIM:

Write a C program to simulate disk scheduling algorithms

a) FCFS b) SSTF c) SCAN

Algorithm FCFS

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.
2. Let us one by one take the tracks in default order and calculate the absolute distance of the track from the head.
3. Increment the total seek count with this distance.
4. Currently serviced track position now becomes the new head position.
5. Go to step 2 until all tracks in request array have not been serviced.

PROGRAM

```
#include<stdio.h>main()
{
    int t[20], n, I, j, tohm[20], tot=0;float
    avhm;
    clrscr();
    printf("enter the no.of tracks");scanf("%d",&n);
    printf("enter the tracks to be traversed");
    for(i=2;i<n+2;i++)
        scanf("%d",&t*i+);
    for(i=1;i<n+1;i++)
    {
        tohm[i]=t[i+1]-t[i];
        if(tohm[i]<0)
            tohm[i]=tohm[i]*(-1);
    }
    for(i=1;i<n+1;i++)
        tot+=tohm[i];
    avhm=(float)tot/n;
    printf("Tracks traversed\tDifference between tracks\n"); for(i=1;i<n+1;i++)printf("%d\t\t\t%d\n",t*i+,tohm*i+);
     printf("\nAverage header movements:%f",avhm);getch();
}
```

OUTPUT

Enter no.of tracks:9

Enter track position:55 58 60 70 18 90 150 160 184

Tracks traversed	Difference between tracks
55	45
58	3
60	2
70	10
18	52
90	72
150	60
160	10
184	24

Average header movements:30.888889

SSTF DISK SCHEDULING ALGORITHM

1. Let Request array represents an array storing indexes of tracks that have been requested. ‘head’ is the position of disk head.
2. Find the positive distance of all tracks in the request array from head.
3. Find a track from requested array which has not been accessed/serviced yet and has minimum distance from head.
4. Increment the total seek count with this distance.
5. Currently serviced track position now becomes the new head position.
6. Go to step 2 until all tracks in request array have not been serviced.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
int queue[100],t[100],head,seek=0,n,i,j,temp;float avg;
// clrscr();
printf("*** SSTF Disk Scheduling Algorithm ***\n");printf("Enter
the size of Queue\t");
scanf("%d",&n); printf("Enter the
Queue\t");for(i=0;i<n;i++)
{
scanf("%d",&queue[i]);
}
printf("Enter the initial head position\t");
scanf("%d",&head);
for(i=1;i<n;i++) t[i]=abs(head-queue[i]);
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
if(t[i]>t[j])
{
temp=t[i]; t[i]=t[j];
t[j]=temp; temp=queue[i];
queue[i]=queue[j];
queue[j]=temp;
```

```
}
```

```
}
}
for(i=1;i<n-1;i++)
{
seek=seek+abs(head-queue[i]);
head=queue[i];
}
printf("\nTotal Seek Time is%d\t",seek);
avg=seek/(float)n;
printf("\nAverage Seek Time is %f\t",avg);return 0;
} OUTPUT:

*** SSTF Disk Scheduling Algorithm ***

Enter the size of Queue 5 Enter the
Queue 10 17 2 15 4
Enter the initial head position 3
Total Seek Time is14

  Average Seek Time is 2.800000
```

SCAN DISK SCHEDULING ALGORITHM

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.
2. Let direction represents whether the head is moving towards left or right.
3. In the direction in which head is moving service all tracks one by one.
4. Calculate the absolute distance of the track from the head.
5. Increment the total seek count with this distance.
6. Currently serviced track position now becomes the new head position.
7. Go to step 3 until we reach at one of the ends of the disk.
8. If we reach at the end of the disk reverse the direction and go to step 2 until all tracks in request array have not been serviced.

PROGRAM

```
#include<stdio.h>main()
{
        int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;clrscr();
        printf("enter the no of tracks to be traveresed");scanf("%d'",&n);
        printf("enter the position  of head");scanf("%d",&h);
        t[0]=0;t[1]=h;
        printf("enter the tracks");
        for(i=2;i<n+2;i++)
                scanf("%d",&t[i]);
        for(i=0;i<n+2;i++)
        {
                for(j=0;j<(n+2)-i-1;j++)
                {
                        if(t[j]>t[j+1])
                        {
                                temp=t[j];
                                t[j]=t[j+1];
                                t[j+1]=temp;
                        }
                }
        }
        for(i=0;i<n+2;i++)
```

```
if(t[i]==h)
        j=i;k=i;
```

```
        p=0;
        while(t[j]!=0)
        {
                atr[p]=t[j];j--;
                p++;
        }
        atr[p]=t[j];
         for(p=k+1;p<n+2;p++,k++)
                atr[p]=t[k+1];
        for(j=0;j<n+1;j++)
        {
                if(atr[j]>atr[j+1])
                        d[j]=atr[j]-atr[j+1];
                else
                        d[j]=atr[j+1]-atr[j];
                sum+=d[j];
        }
        printf("\nAverage header movements:%f",(float)sum/n);getch();
}
```

OUTPUT

```
Enter no.of tracks:9
Enter track position:55  58  60  70  18  90  150  160  184

Tracks traversed        Difference between tracks
150     50
160     10
184     24
90      94
70      20
60      10
58      2
55      3
18      37

                Average header movements:  27.77
```

RESULT:

Thus the program to implement disk Scheduling algorithms has been executed successful.

EX.NO:9 IMPLEMENTATION OF THE FIFO PAGE REPLACEMENT ALGORITHMS

AIM:

Installing Linux on Windows in a virtual machine is straightforward. Here's how to install Linux on VMware Workstation, step by step.

Use a PC That Supports Virtualization

A virtual machine is a software environment that replicates the conditions of a hardware environment: a personal computer. The environment is based on the hardware of your physical PC and limited only by the components within. For instance, you couldn't have a virtual four core CPU on a processor with two cores.

However, while virtualization can be achieved on many systems, the results will be far superior on computers equipped with a CPU that supports it.

Several VM tools make it easy to install Linux operating systems (OS). VMware produces the most accomplished virtual machine applications. Let's find out how to install Linux in Windows with VMware Workstation Player.

Install VMware Workstation Player

To start, head to the VMware website and download the latest version of their Workstation Player tool. We're using VMware Workstation 15 Player, which is around 150MB to download.

Download: VMware Workstation 15 Player (Free)

VMware Workstation Player is free and available for non-commercial, personal, and home use. Students and non-profit organizations can also benefit from the free version. In terms of functionality, VMware Workstation Player includes everything you could need for the standard virtual machine tasks.

However, VMware offers a wide selection of virtualization solutions aimed at businesses of all levels. You can find out more about their solutions on the website's product page.

Once VMware Workstation Player has downloaded, launch the installer and follow the installation wizard. You'll see the option to install an Enhanced Keyboard Driver---while you won't need this initially, it's worth having.

Proceed through the installation wizard, and restart Windows when prompted.

Choose Your Preferred Linux OS

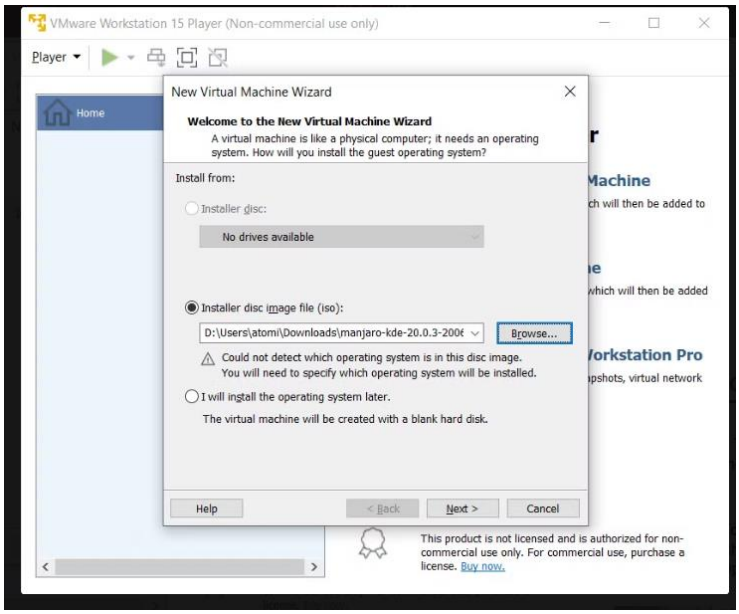
You probably know which Linux OS you want to try. Some Linux distros are particularly suited to running in a VM, but others are not. All 32-bit and 64-bit distros work in a virtual machine. However, you cannot run Linux distros for ARM architecture (such as the Raspberry Pi) in VMware.

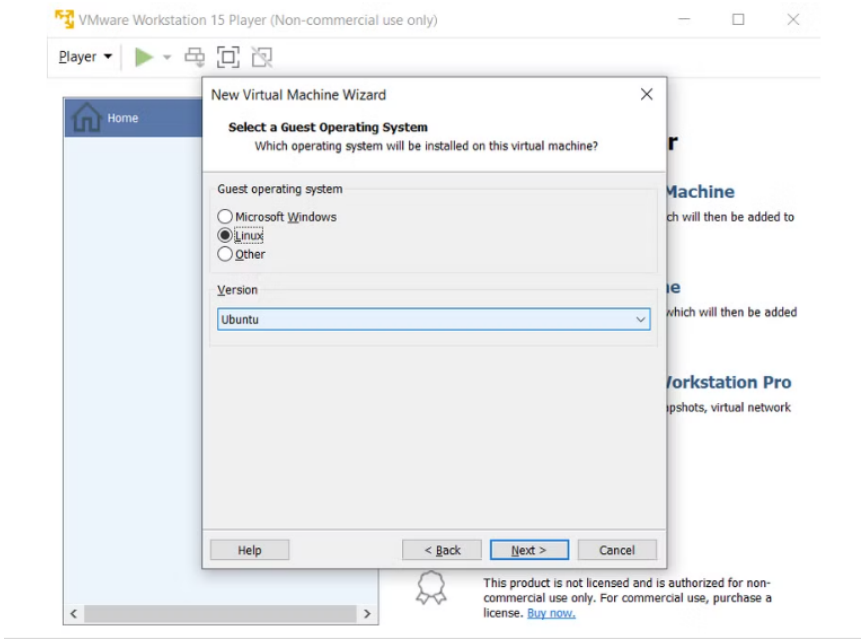
If you want to emulate an ARM environment in Windows, try QEMU.

Create Your Linux Virtual Machine

While your Linux ISO downloads, it's a good time to start configuring your VM. Start by launching VMware Workstation Player. When you're ready to create a VM:

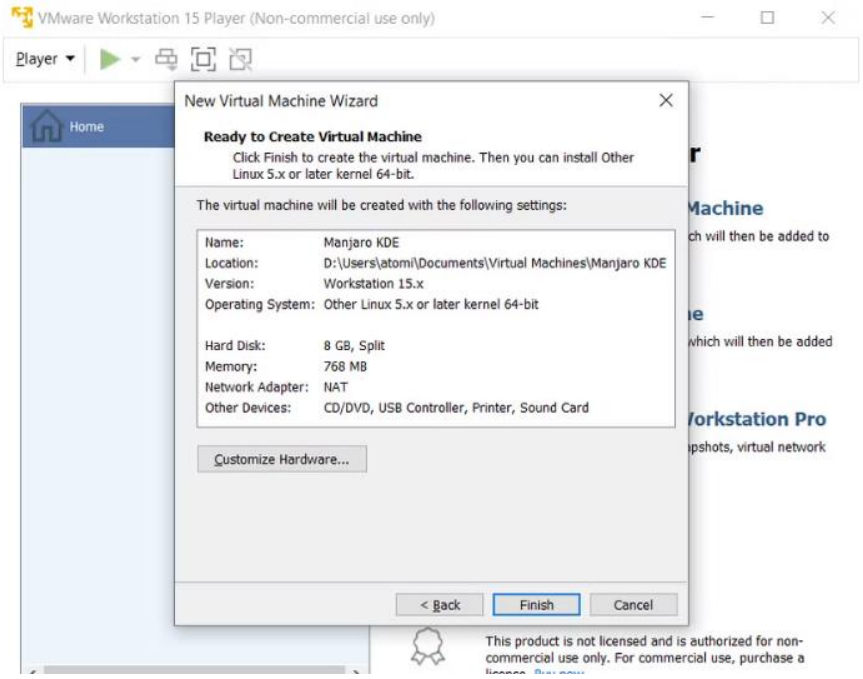
1. Click **Create a New Virtual Machine**
2. Select the default option, **Installer disc image file (iso)**
3. Click **Browse** to find the ISO file
4. With "guest" OS selected, click **Next**
5. Select **Linux** as the Guest operating system type
6. Under **Version**, scroll through the list and select the OS
7. Click **Next** to proceed and if necessary, input a **Virtual machine name**
8. Confirm the storage **Location** and change if needed





With the operating system selected and configured, it's time to build the virtual machine.

1) Under **Specify Disk Capacity** adjust **Maximum disk size** if required (the default should be enough)
2) Select **Split virtual disk into multiple files** as this makes moving the VM to a new PC easy
3) Click **Next** then confirm the details on the next screen
4) If anything seems wrong click **Back**, otherwise click **Finish**

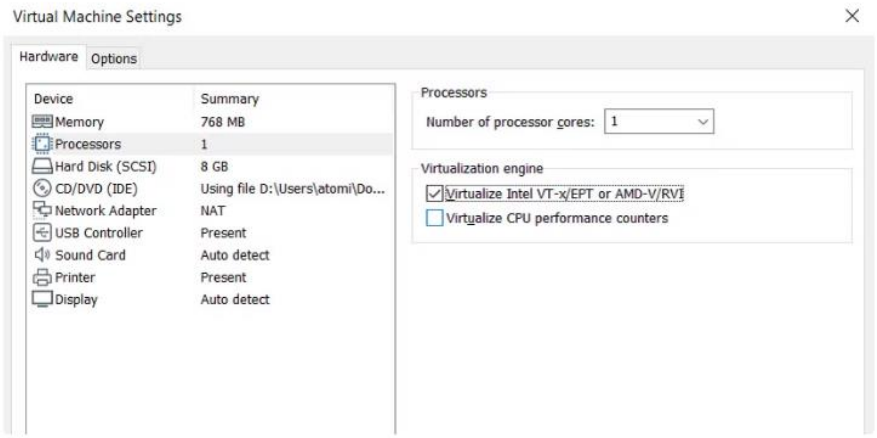


Your Linux virtual machine will be added to VMware Workstation Player.

Customize Your Virtual Hardware

In some cases, you might need to customize the virtual machine before installing Linux. Alternatively, you might install the OS and find there is something missing.

To fix this, right-click your virtual machine in VMware Workstation Player and select **Settings**.



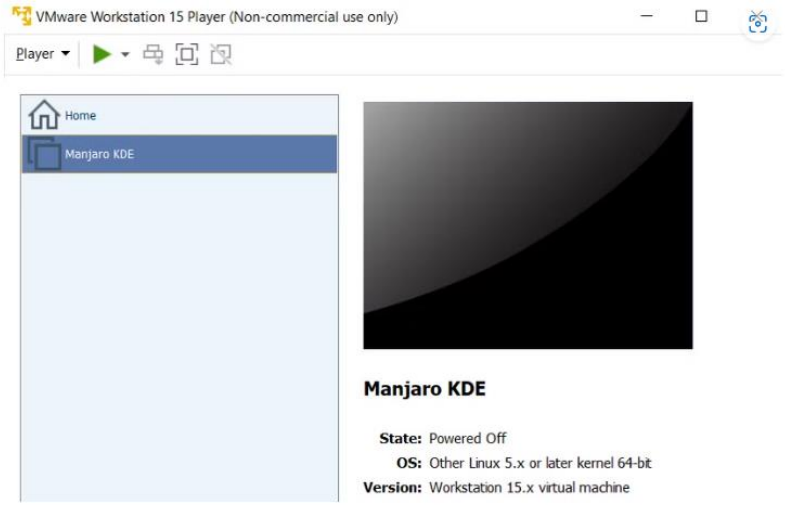
Here, you can tweak the virtual machine's hardware in other ways beyond the HDD. You have options for the **Memory**, **Processors**, **Network Adaptor** configuration, and much more.

It's worth taking a look at the **Processors** screen. In the right-hand pane, you'll spot a reference to a **Virtualization engine**. By default, this works automatically, but for troubleshooting set Intel VT-x or AMD-V, depending on your CPU.

You can address performance issues in the **Memory** screen. Here you'll spot an illustration of the suggested RAM size, as well as recommended options for your virtual machine. It's a good idea to stick to these recommendations. Going too small will prove a problem, while setting the RAM too high will impact on your PC's performance, slowing everything from standard system tasks to running the VM software!

Finally, spare a moment to check the **Display** settings. Default settings should be fine but if there is an issue with the display you can toggle 3D acceleration. Multiple monitors can be used and custom resolution set, but note that some modes will clash with some desktops.

Click **OK** to confirm changes, then select the virtual machine and click the **Play** button to begin.



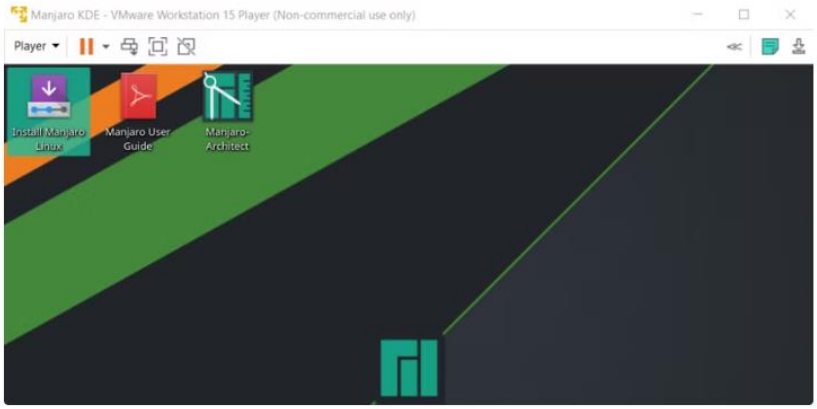
Download and Install VMware Tools

On the first boot of your virtual machine, you'll be prompted to **Download and Install** VMware Tools for Linux. Agree to this, then wait as it is downloaded.

VMware Tools will enhance the performance of the virtual machine while enabling shared folders between host and guest machines.

How to Install Linux in VMware

When the ISO boots in the virtual machine, it will boot into the live environment. This is a temporary Linux that exists only on the boot media and in the system memory. To ensure the environment persists, use the **Install** option on the desktop.



From this point, the installation will continue as if you're installing an OS on a physical machine. Progress through the installation wizard, creating a user account and setting other options when prompted.

Once the installation is complete, you'll be able to log into the Linux virtual machine and start using the guest OS. It's that simple!

How to Run Linux in a Virtual Machine

Now you can launch the Linux virtual machine at any time using the **Play** button in VMware Workstation Player.

Looking for some software to install?

Often, Linux ships with a number of preinstalled applications but if you want something else, check the best Linux apps.

By the way, if you just want to get into the Linux terminal, things are far simpler than installing VMware. Check out how to access the bash shell on Windows.

Install Any Linux Distro in a Virtual Machine on Windows!

If you want easy access to Linux, the best option is to install it in a virtual machine in Windows. VMware Workstation Player provides the best tools for doing just that.

Installing Linux in VMware is simple. Let's run through the steps again:

1. Download the free VMware Workstation Player
2. Install, and restart Windows
3. Create and configure your virtual machine
4. Install Linux in the virtual machine
5. Restart the virtual machine and use Linux

RESULT:

Thus , we can choose one OS from hundreds of Linux distros, which we can install in a VMware-based virtual machine.

VIVA QUESTIONS AND ANSWERS

Ex.No:1

1.What is the command to list all files and directories in a directory?

The command to list all files and directories in a directory is ls.

2.How can you create a new directory using a Linux command?

The command to create a new directory in Linux is mkdir. For example, to create a directory named "my_directory", you can use the command mkdir my_directory.

3.What is the command to delete a file in Linux?

The command to delete a file in Linux is rm. For example, to delete a file named "my_file.txt", you can use the command rm my_file.txt.

4. How can you copy a file from one directory to another using a Linux command?

The command to copy a file from one directory to another in Linux is cp. For example, to copy a file named "my_file.txt" from the directory "dir1" to the directory "dir2", you can use the command cp dir1/my_file.txt dir2/.

5.What is the command to display the contents of a file in Linux?

The command to display the contents of a file in Linux is cat. For example, to display the contents of a file named "my_file.txt", you can use the command cat my_file.txt.

6.What is the command to rename a file in Linux?

The command to rename a file in Linux is mv. The mv command is also used to move files, but it can be used to rename a file by specifying a new name for the file. The basic syntax for renaming a file using the mv command is: mv old_filename new_filename

7.What is the command to move a file from one location to another in linux?

The command to move a file from one location to another in Linux is mv. The mv command can be used to move files and directories, as well as to rename files. The basic syntax for moving a file using the mv command is: mv source_file destination_directory

8.How can you change the permissions of a file in Linux? Answer: The command to change the permissions of a file in Linux is chmod. For example, to give the owner of a file read, write, and execute permissions, you can use the command chmod u+rwx file.txt.

9.What is the command to search for a file or directory in Linux?

The command to search for a file or directory in Linux is find. For example, to search for a file named "my_file.txt" in the current directory and all its subdirectories, you can use the command find . -name my_file.txt.

10.Can you explain the difference between internal and external commands?

Internal commands refer to commands that are built into the shell or command-line interpreter itself. These commands are typically simple and perform basic operations such as changing directories or displaying the contents of a file. Examples of internal commands in Linux include cd, pwd, echo, and history. External commands, on the other hand, refer to commands that are separate programs or scripts that can be executed by the shell or command-line interpreter. These commands are

typically more complex and perform specialized operations such as file compression, network configuration, or software installation. Examples of external commands in Linux include ls, tar, ping, and grep.

Ex.No:2

1. What is a system call?

A system call is a mechanism used by programs to request a service or resource from the operating system (OS) kernel. It provides an interface between the user-space application and the kernel, which is the core part of the operating system responsible for managing system resources such as memory, processes, and I/O. In general, a system call allows a program to perform privileged operations that it cannot perform directly, such as reading or writing to a file, creating a new process, or allocating memory. When a system call is made, the program transfers control to the kernel, which executes the requested operation on behalf of the program and returns the result to the program.

2. How do you open a file using system calls?

In order to open a file using system calls in a C program, you can use the open() function, which is defined in the header file. Syntax: int open(const char *pathname, int flags);

3. What is the difference between reading and writing a file using system calls?

Reading and writing a file using system calls involve different functions and modes of operation. When you want to read data from a file, you can use the read() system call. The read() function takes three parameters: the file descriptor of the open file, a pointer to a buffer where the data will be stored, and the maximum number of bytes to read. On the other hand, when you want to write data to a file, you can use the write() system call. The write() function takes three parameters: the file descriptor of the open file, a pointer to a buffer containing the data to write, and the number of bytes to write.

4. How do you read a file using system calls?

To read a file using system calls, you can use the read() function. The read() function reads data from an open file descriptor and stores it in a buffer.

5. How do you write to a file using system calls?

To write to a file using system calls, you can use the write() function. The write() function writes data to an open file descriptor.
6. What is the purpose of the open() function in file manipulation?

The open() function in file manipulation is used to open a file and obtain a file descriptor. A file descriptor is an integer that uniquely identifies an open file within a process. The open() function takes two arguments: a filename and a set of flags that control how the file should be opened. The flags can specify whether the file should be opened for reading, writing, or both; whether the file should be created if it does not exist; and whether the file should be truncated if it already exists.

7.What is the purpose of the read() function in file manipulation?

The read() function in file manipulation is used to read data from an open file descriptor into a buffer.

8.What is the purpose of the write() function in file manipulation?

The write() function in file manipulation is used to write data to an open file descriptor from a buffer.

9.What is the purpose of the close() function in file manipulation?

The close() function in file manipulation is used to close an open file descriptor. When a file is opened using open() or a similar function, a file descriptor is returned that can be used to read from or write to the file. When the file is no longer needed, the file descriptor should be closed using close(). Closing a file descriptor releases the resources associated with the open file, including any memory buffers, system resources, or locks that were used. It also ensures that any pending I/O operations on the file are completed.

10.How do you create a new file using system calls?

To create a new file using system calls, you can use the open() function with the O_CREAT flag. The open() function is used to open or create a file, and returns a file descriptor that can be used for subsequent I/O operations.

11.How do you delete a file using system calls?

To delete a file using system calls, you can use the unlink() function in C. The unlink() function is used to remove a file from the file system.

12.How do you rename a file using system calls?

To rename a file using system calls, you can use the rename() function in C. The rename() function is used to change the name or location of a file in the file system.

13.What is the purpose of the chmod() function in file manipulation?

The chmod() function is used to change the permissions of a file in the file system. It is a system call in Unix-like operating systems and is typically used in C or other programming languages that allow direct access to system calls. The chmod() function takes two arguments: the name of the file whose permissions you want to change, and the new permissions you want to set. The new permissions are specified using a three-digit octal value, where each digit corresponds to the permissions for the owner, group, and others, respectively.

14.How do you get information about a file using system calls?

You can get information about a file using system calls such as stat() or fstat() in Unix-like operating systems. These system calls provide detailed information about a file, including its size, permissions, owner, and last access and modification times.

15.What is the purpose of the stat() function in file manipulation?

The stat() function is used to get information about a file, such as its size, type, permissions, owner, and modification time. It takes the name of the file as an argument and returns a structure containing information about the file. The stat() function is commonly used in file manipulation programs to check the status of a file before performing operations on it. The information returned by the stat() function can be used to determine whether the file is a regular file, a directory, a symbolic link, or some other type of file.

16.How do you check if a file exists using system calls?

To check if a file exists using system calls in C, you can use the access() function. The access() function checks whether the file specified by the file path exists and whether the user has permission to access it. It returns 0 if the file exists and the user has permission to access it, and -1 otherwise.

Ex.No:3

1.What are various scheduling queues?
Job queue, Ready queue, Device queue

2.What is a job queue?
When a process enters the system it is placed in the job queue.

3.What is a ready queue?
The processes that are residing in the main memory and are ready and waiting to execute are kept on a list called the ready queue.

4.What is a device queue?
A list of processes waiting for a particular I/O device is called device queue.

5.What is a long term scheduler & short term schedulers?
Long term schedulers are the job schedulers that select processes from the job queue and load them into memory for execution. The short term schedulers are the CPU schedulers that select a process form the ready queue and allocate the CPU to one of them.

6.What is context switching?
Transferring the control from one process to other process requires saving the state of the old process and loading the saved state for new process. This task is known as context switching.

7.What are the disadvantages of context switching?
Time taken for switching from one process to other is pure over head. Because the system does no useful work while switching. So one of the solutions is to go for threading when ever possible.

8.What are co-operating processes?
The processes which share system resources as data among each other. Also the processes can communicate with each other via interprocess communication facility generally used in distributed systems. The best example is chat program used on the www.

9.Which module gives control of the CPU to the process selected by the short-term scheduler?

Dispatcher module gives the control of the CPU to the process which was selected by the short-term scheduler.

10.What is Throughput?

The interval from the time of submission of a process to the time of completion is termed as throughput.

11.Which algorithm is defined in Time quantum?

Round Robin Scheduling Algorithm.

12.Which CPU Scheduling Algorithm is best and why?

SRTF algorithm is best . Because it gives minimum average waiting time.

13. Recall the different CPU Scheduling algorithms.

FCFS,RR,SJF,Priority,Multilevel Queue Scheduling ,Multilevel Feedback Queue Scheduling

14. What is the expansion of SRTF?

Shortest Remaining Time first.

Ex.No:4

1. What is Mutual Exclusion?

If a process is executing in its critical section, then no other processes can be executed in their critical section. This condition is called Mutual Exclusion. Critical section of the process is shared between multiple processes. If this section is executed by more than one or all of them concurrently then the outcome of this is not as per desired outcome. For this reason the critical section of the process should not be executed concurrently.

2.What is a race condition?

A situation, where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called a race condition.

3.Define a critical section.

The part of the program, where the shared memory is accessed is called as the critical region or critical section.

4.What is meant by mutual exclusion?

Mutual exclusion is a way of making sure that, if one process is using a shared variable or file, the other processes will be excluded from using the same variable or file.

5.What is a semaphore?

A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait and signal. These operations were originally termed P (for wait; from the Dutch proberen, to test) and V (for signal; from verhogen, to increment).

6.What is a monitor?

A monitor is a concurrency construct that contains both the data and procedures needed to perform allocation of a particular, serially reusable shared resources or group of serially reusable shared resources.

6. What are the solution for Critical section problem?
   Mutual exclusion,Bounded waiting,Progress

7. Recall the different types of semaphore.
   Binary Semaphore,Counting Semaphore.

8. What is a dining philosophers problem?

The Dining Philosopher Problem is a classic synchronization and concurrency problem that deals with resource sharing, deadlock, and starvation in systems where multiple processes require limited resources.The Dining Philosopher Problem involves 'n' philosophers sitting around a circular table. Each philosopher alternates between two states: thinking and eating. To eat, a philosopher needs two chopsticks, one on their left and one on their right. However, the number of chopsticks is equal to the number of philosophers, and each chopstick is shared between two neighboring philosophers.

9. List the constraints and conditions for the Dining Philosophers problem.

- Every Philosopher needs two forks to eat.
- Every Philosopher may pick up the forks on the left or right but only one fork at once.
- Philosophers only eat when they have two forks. We have to design such a protocol i.e. pre and post protocol which ensures that a philosopher only eats if he or she has two forks.
- Each fork is either clean or dirty.

10. What is the main challenge in the dining philosophers problem?
The main challenge in the Dining philosophers problem is prevention of the deadlocks.

Ex.No:5

1.What is meant by deadlock?

A process requests resources; if the resources are not available at that time, the process enters a wait state. Waiting processes may never again change state, because the resources they have requested are held by other waiting processes. This situation is called a deadlock.

2.Explain the necessary conditions for deadlock.

A deadlock situation can arise if the following four conditions hold simultaneously in a system:
a. Mutual exclusion
b. Hold and wait
c. No preemption
d. Circular wait

3.What is a resource-allocation graph? (Jan 08 CU)

Deadlock can be described more precisely in terms of a directed graph called a system resource-allocation graph. This graph consists of a set of vertices V and a set of edges E. The set of vertices V is partitioned into two different types of nodes P={P1, P2, . . . , Pn
}, the set consisting of all the active processes in the system, and R={R1, R2, . . . , Rm}, the set of all resource types in the system.

4.How can we handle deadlock?

Deadlock can be handled in one of the following ways:
a. Protocols can be used to prevent or avoid deadlocks, ensuring that the system will never enter a deadlock state.
b. The system can be allowed to enter a deadlock state which can be detected and recovered.
c. The problem can be ignored and pretending that deadlocks never occurs in the system.

5.What is meant by deadlock prevention?

Deadlock prevention is a set of methods for ensuring that at least one of the necessary conditions cannot hold.

6.Write notes on deadlock avoidance.

Deadlock avoidance; require that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime.

7.When is a system in a safe state?

A state is safe if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock. More formally, a system is in a safe state if there exists a safe sequence. A sequence of processes <P1, P2, . . . , , Pn> is a safe sequence for the current allocation state if, for each Pi the resources that Pi can still request can be satisfied by the currently available resources plus the resources held by all the Pj, with j<i.

8.What should the system do when a deadlock occurs?

When a deadlock situation occurs, the system must provide the following:
a. An algorithm that examines the state of the system to determine whether a deadlock has occurred.
b. An algorithm to recover from deadlock

9.How can the system recover from deadlock?

There are two options for breaking a deadlock. They are:
a. Abort one or more processes to break the circular wait.
b. Preempt some resources from one or more of the deadlocked processes.

10.What are the ways of terminating a process during deadlocks?

To eliminate deadlocks by aborting a process, the following two methods can be used:
a. Abort all deadlocked processes.
b. Abort one process at a time until the deadlock cycle is eliminated.

11.List the issues that needed to be addresses, when preemption is used to deal with deadlocks.

If preemption is required to deal with deadlock, then three issues need to be addressed:
a. Selecting a victim
b. Rollback
c. Starvation

12.What are the ways of terminating a process during deadlocks?
To eliminate deadlocks by aborting a process, the following two methods can be used.

Ex.No:6

1.Differentiate between logical address space and physical address space.

An address generated by the CPU is commonly referred to as a logical address whereas an address seen by the memory unit – that is, the one loaded into the memory-address register of the memory – is commonly referred to as a physical address. The set of all logical addresses generated by a program is a logical-address space; the set of all physical addresses corresponding to these logical addresses is a physical address space.

2.What is a memory-management unit?

The run-time mapping from virtual to physical address is done by a hardware device called the memory-management unit.

3.Write short notes on dynamic loading.

To obtain better memory-space utilization, dynamic loading can be used. A routine is not loaded until it is called. The advantage of dynamic loading is that an unused routine is never loaded.

.

4.What is internal fragmentation?

Memory that is internal to a partition but is not being used results in internal fragmentation.

5.Explain paging technique.

Paging is a memory-management scheme that permits the physical-address space of a process to be noncontiguous. Paging avoids considerable problem of fitting the varying-sized memory chunks onto the backing store, from which most of the previous memory-management schemes suffered.

6.Define frames.

Physical memory is broken into fixed-sized blocks called frames.

7.Define pages.
Logical memory is broken into blocks of the same size called pages.

8.Define frame table.

The operating system is managing physical memory, it must be ware of the allocation details of physical memory; which frames are allocated, which frames are available, how many total frames there are, and so on. This information is generally kept in a data structure called a frame table

9.Explain the term page-table base register.

The page table is kept in main memory, and a page-table base register (PTBR) points to the page table.

10.Explain the term Translation look–aside buffer.

The TLB is associative, high-speed memory. Each entry in the TLB consists of two parts: a key (or tag) and a value. When the associative memory presented with an item, it is compared with all keys simultaneously. If the item is found, the corresponding value field is returned. The search is fast; the hardware is expensive. Typically, the number of entries in a TLB is small, often numbering between 64, and 1024.

11.What is a hit ratio?

The percentage of times that a particular page number is found in the TLB is called the hit ratio.

12.What are the different types of page table?

- Multilevel or Hierarchical Page Table
- Hashed Page Table
- Inverted Page Table

Ex.No:7

1.What is a reference string?

An algorithm is evaluated by running it on a particular string of memory references and computing the number of page faults. The string of memory reference is called a reference string.

2.List the various page replacement algorithms.

Page replacement algorithms include:
a. FIFO page replacement
b. Optimal page replacement
c. LRU page replacement

d. LRU approximation page replacement
e.Counting-based page replacement

3.How does FIFO page replacement work?

A FIFO page replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen.

4.State Belady's anomaly.

Belady's anomaly states that for some page replacement algorithms, the page-fault rate may increase as the number of allocated frames increases.

5.Explain lease-recently-used algorithm.

LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses that page that has not been used for the longest period of time.

6.Describe least frequently used page-replacement.

The least frequently used (LFU) page-replacement algorithm requires that the page with the smallest count be replaced.

7.How does most frequently used page-replacement work?

The most frequently used (MFU) page-replacement algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

8.What is a meant by thrashing?
It is a high paging activity. If a process spents more time in paging than in execution it is called as thrashing.

9.Which Page replacement algorithm is best and why?
Optimal Page replacement algorithm is best as it does not suffer from belady's anamoly.

10.Which Page replacement algorithm suffers from belady's anamoly?
FIFO page replacement algorithm suffers from belady's anamoly.

Ex.No:8

1. Define seek time.

The time taken by the head to move to the appropriate cylinder or track is called seek time.

2. Recall latency.
Once the head is at right track, it must wait until the desired block rotates under the read-write head. This delay is latency time.

3.Define rotational latency.
Rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head.

4.What is disk bandwidth?
The disk bandwidth is the total number of bytes transferred, divided by the time between the first request for service and the completion of the last transfer.

5.Define buffering.
A buffer is a memory area that stores data while they are transferred between two devices or between a device and an application. Buffering is done for three reasons a. To cope with a speed mismatch between the producer and consumer of a data stream b. To adapt between devices that have different data transfer sizes c. To support copy semantics for application I/O

6.Define caching.
A cache is a region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original. Caching and buffering are distinct functions, but sometimes a region of memory can be used for both purposes.

7.What are the various disk-scheduling algorithms?
The various disk-scheduling algorithms are
a. First Come First Served Scheduling
b. Shortest Seek Time First Scheduling
c. SCAN Scheduling
d. C-SCAN Scheduling
f. LOOK scheduling

8.What is low-level formatting?
Before a disk can store data, it must be divided into sectors that the disk controller can read and write. This process is called low-level formatting or physical formatting. Low-level formatting fills the disk with a special data structure for each sector. The data structure for a sector consists of a header, a data area, and a trailer.

9.What is the use of boot block?
For a computer to start running when powered up or rebooted it needs to have an initial program to run. This bootstrap program tends to be simple. It finds the operating system on the disk loads that kernel into memory and jumps to an initial address to begin the operating system execution. The full bootstrap program is stored in a partition called the boot blocks, at fixed location on the disk. A disk that has boot partition is called boot disk or system disk.

10.What is sector sparing?
Low-level formatting also sets aside spare sectors not visible to the operating system. The controller can be told to replace each bad sector logically with one of the spare sectors. This scheme is known as sector sparing or forwarding.

Ex.No:9

1. What is virtualization?

Virtualization is technology that lets you create useful IT services using resources that are traditionally bound to hardware. It allows you to use a physical machine's full capacity by distributing its capabilities among many users or environments.

2. List down the different types of virtualization.
   - Desktop Virtualization.
   - Application Virtualization.
   - Server Virtualization.
   - Storage Virtualization.
   - Network Virtualization.

3.List the advantages of virtualization.

- Uses Hardware Efficiently.

- Available at all Times.
- Recovery is Easy.
- Quick and Easy Setup.
- Cloud Migration is Easier.

4.List the disadvantages of virtualization.

- High Initial Investment.
- Data Can be at Risk.
- Quick Scalability is a Challenge.
- Performance Witnesses a Dip.

5.What is OS virtualization?

In this virtualization, a user installs the virtualization software in the operating system of his system like any other program and utilize this application to operate and generate various virtual machines. Here, the virtualization software allows direct access to any of the created virtual machine to the user. As the host OS can provide hardware devices with the mandatory support, operating system virtualization may affect compatibility issues of hardware even when the hardware driver is not allocated to the virtualization software.

6.List down the operating system based services offered for OS virtualization.

- Backup and Recovery.
- Security Management.
- Integration to Directory Services.

7.List the characteristics of Virtualization.

- Increased Security
- Managed Execution
- Sharing
- Aggregation
- Emulation
- Isolation
- Portability

9. What is a VMware?

VMware is a leading provider of virtualization software that plays a crucial role in many enterprises' IT infrastructures. Virtualization software creates an abstraction layer over computer hardware, allowing the hardware elements of a single computer—such as processors, memory, and storage—to be divided into multiple virtual computers called virtual machines (VMs)

10. List the key features of VMware.

- Virtual Machine Creation
- Network Virtualization
- Scalability