ST. ANNE'S COLLEGE OF ENGINEERING AND TECHNOLOGY
(Accredited by NAAC, Approved by AICTE, New Delhi. Affiliated to Anna University, Chennai)
ANGUCHETTYPALAYAM, PANRUTI – 607 106.
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING(AIML)

Subject Code/Subject Name : AL3452/ OPERATING SYSTEMS LAB
Branch : AIML
Year/Semester : II / IV

Syllabus

1. Installation of windows operating system
2. Illustrate UNIX commands and Shell Programming
3. Process Management using System Calls: Fork, Exit, Getpid, Wait, Close
4. Write C programs to implement the various CPU Scheduling Algorithms
5. Illustrate the inter process communication strategy
6. Implement mutual exclusion by Semaphore
7. Write C programs to avoid Deadlock using Banker's Algorithm
8. Write a C program to Implement Deadlock Detection Algorithm
9. Write C program to implement Threading
10. Implement the paging Technique using C program
11. Write C programs to implement the following Memory Allocation Methods
    a. First Fit b. Worst Fit c. Best Fit
12. Write C programs to implement the various Page Replacement Algorithms
13. Write C programs to Implement the various File Organization Techniques
14. Implement the following File Allocation Strategies using C programs
    a. Sequential b. Indexed c. Linked
15. Write C programs for the implementation of various disk scheduling algorithms
16. Install any guest operating system like Linux using VMware.

By

R.MANICKAVASAGAN,AP/CSE

TABLE OF CONTENTS

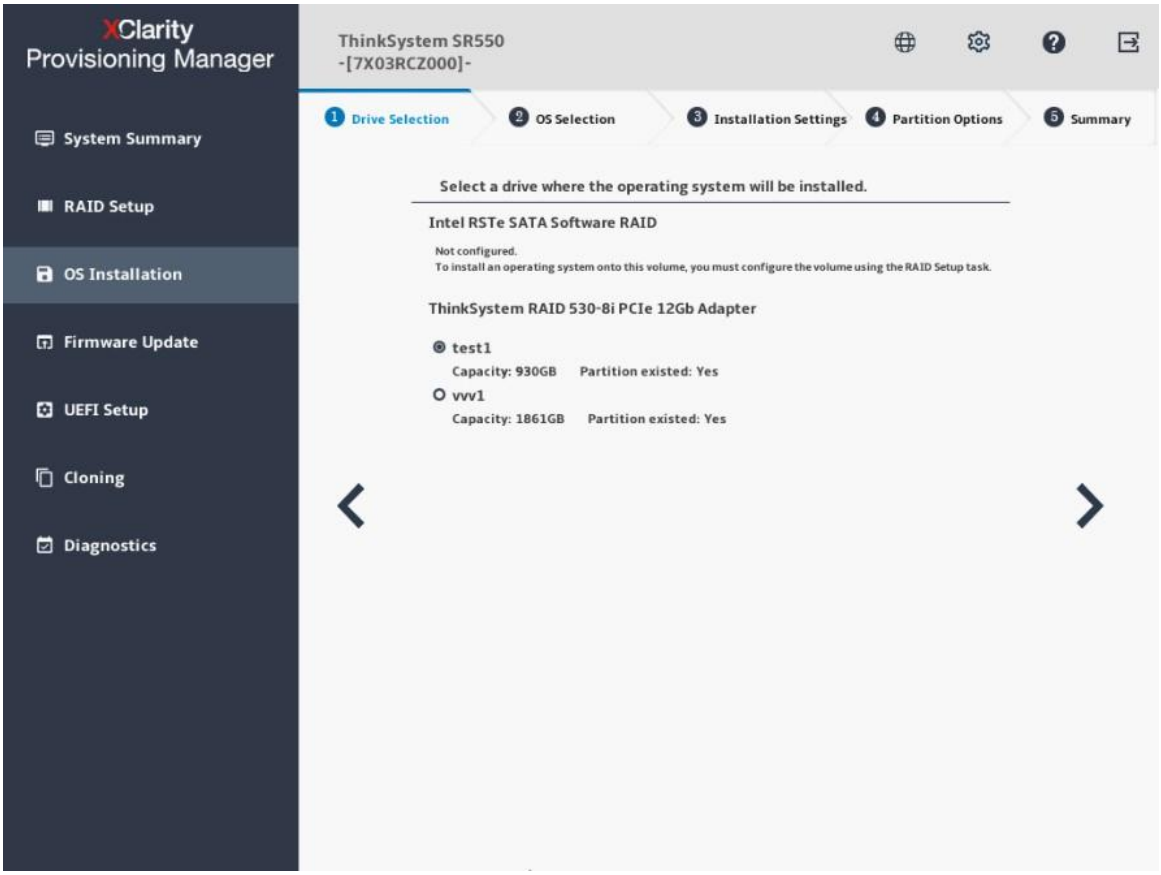
Ex. No: 1 **Installation of windows operating system**

Installing a Windows operating system

The wizard provides a step by step guidance for installing an operating system. Follow the instructions on the screen and the tips listed below to install a Windows operating system.

1. **Drive Selection**

Figure 1. Drive Selection step (for Windows)

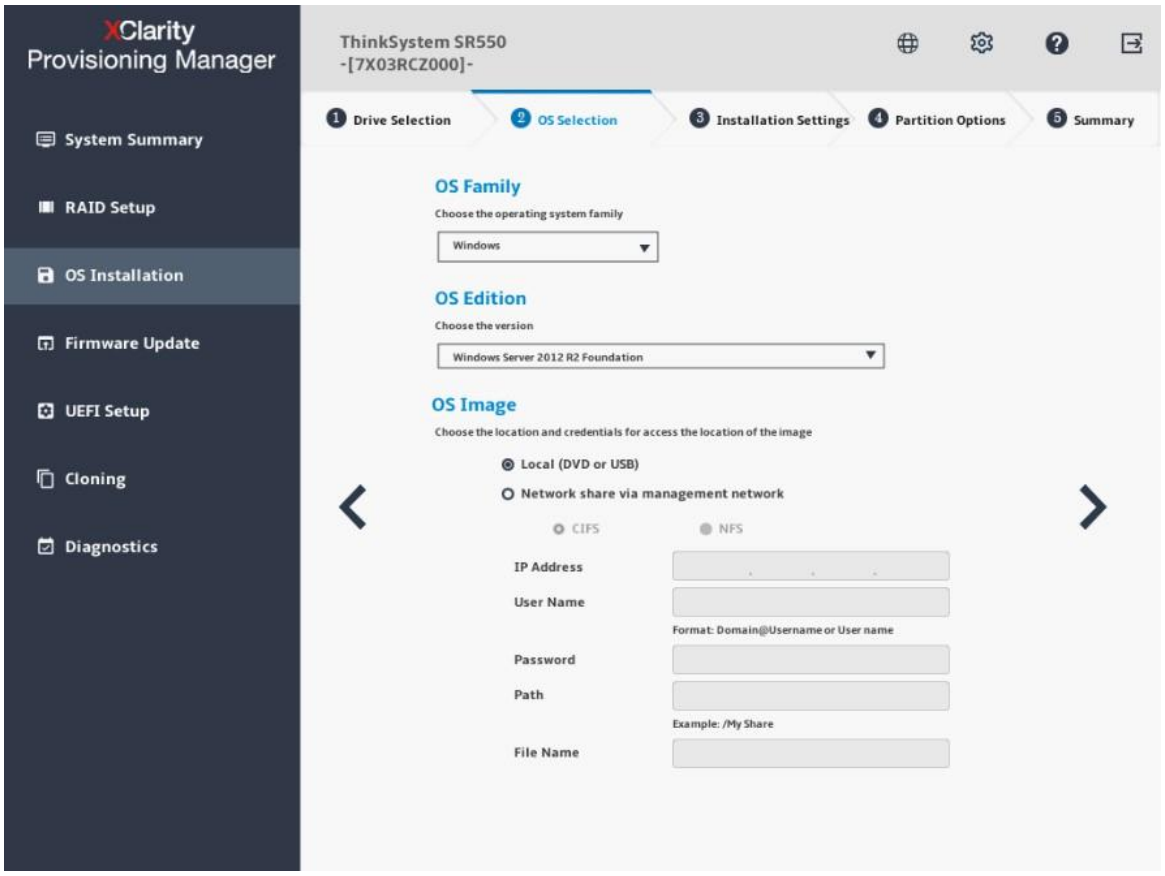


Attention: The selected drive will be formatted during the installation. Back up all data on it before the installation.

Note: The drivers will be installed automatically after the OS installation. It is recommended to restart your server to ensure that all installed drivers take effect.

2. **OS Selection**

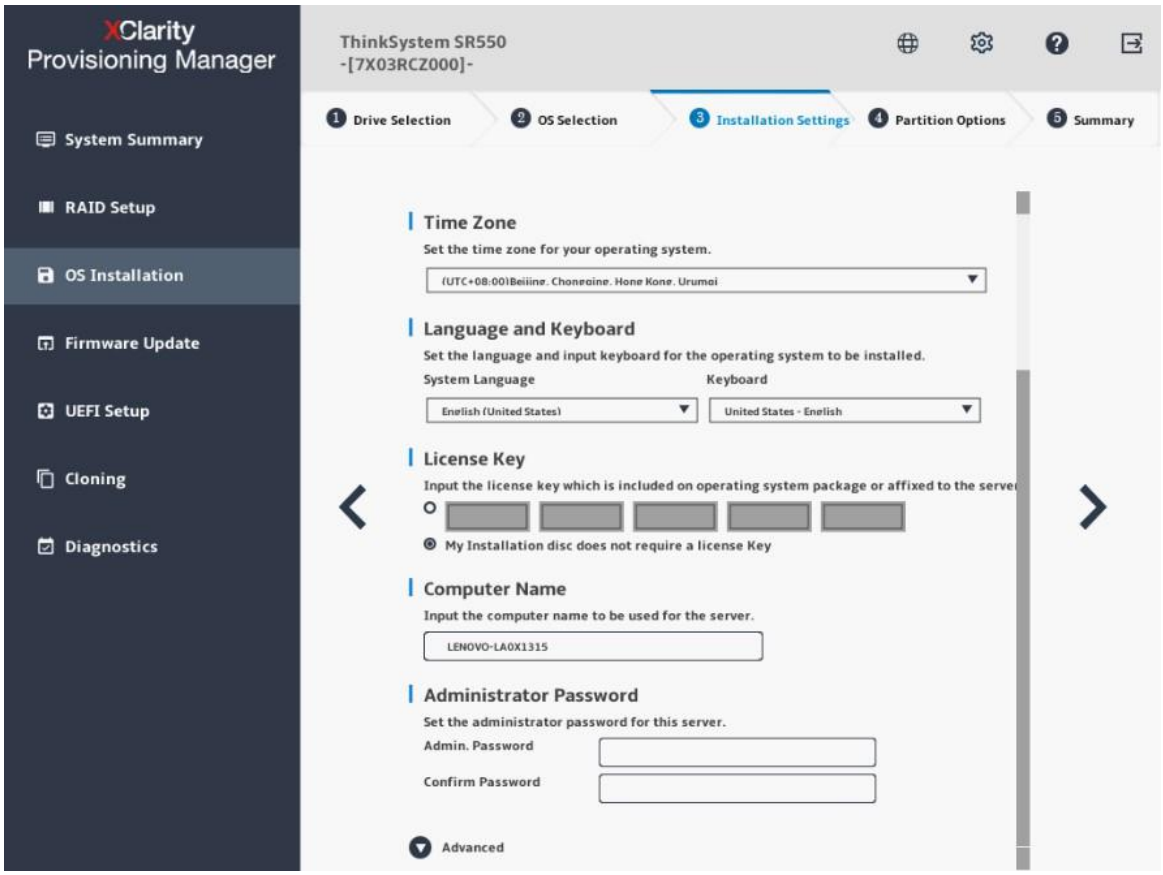
Figure 2. OS Selection step (for Windows)



The IP address is made up of four parts separated by dots. The following table lists the valid value range for each part.

3. **Installation Settings**

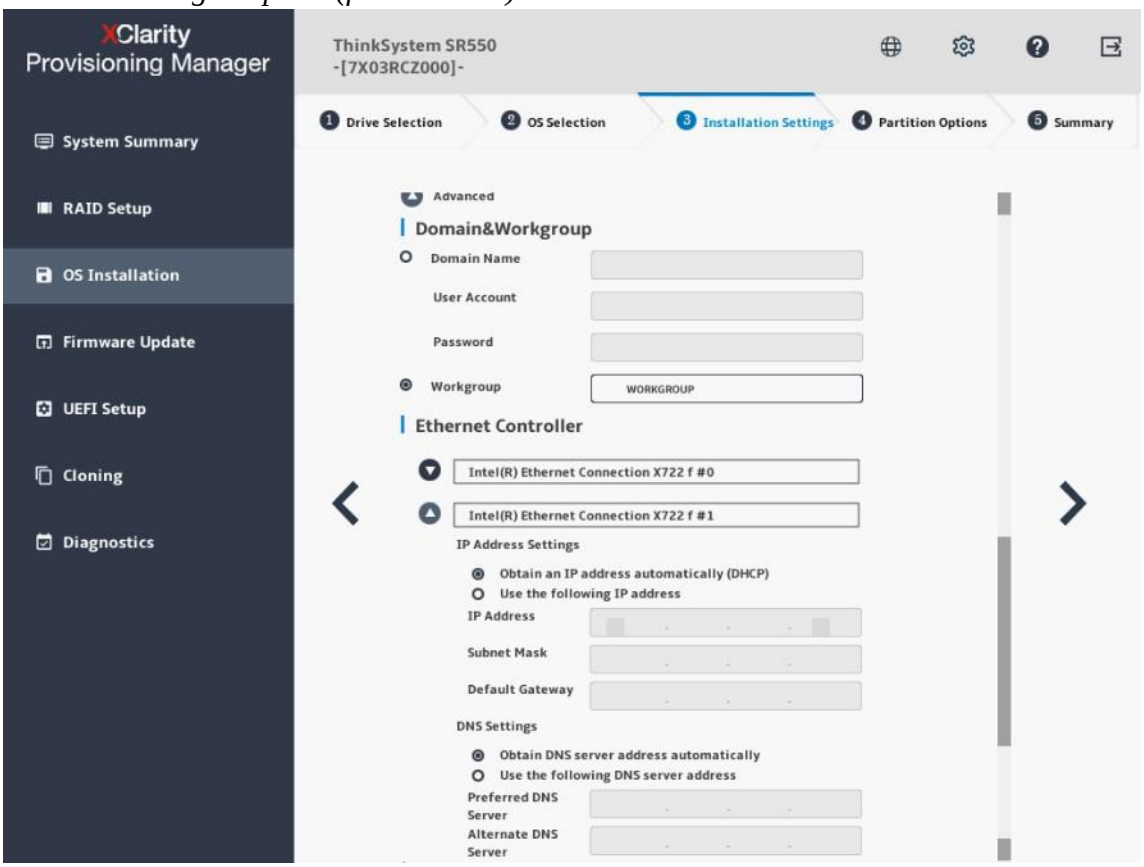
Figure 3. Installation Settings step – 1 (for Windows)



Administrator Password: You can change your administrator password later from the operating system.

If you want to do advanced configurations, expand the list by clicking the arrow icon next to **Advanced**.

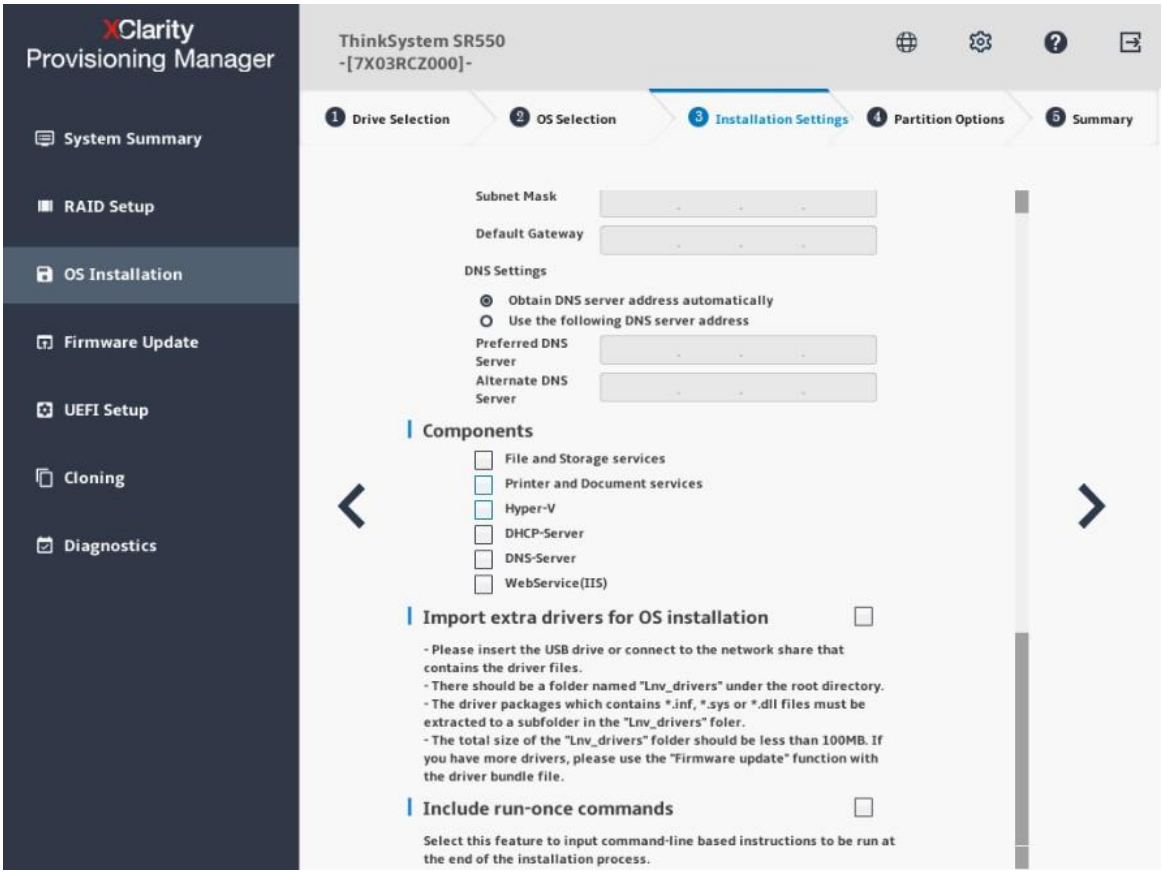
Figure 4. Installation Settings step – 2 (for Windows)



Refer to the following table for the valid values when you type the required address information.

Address	Part 1	Part 2	Part 3	r
ddress	26; 128 – 223	55	55	5
et Mask	55	55	55	5
ult Gateway	55	55	55	5
erred DNS Server	26; 128 – 223	55	55	5
rnate DNS Server	26; 128 – 223	55	55	5

Figure 5. Installation Settings step – 3 (for Windows)

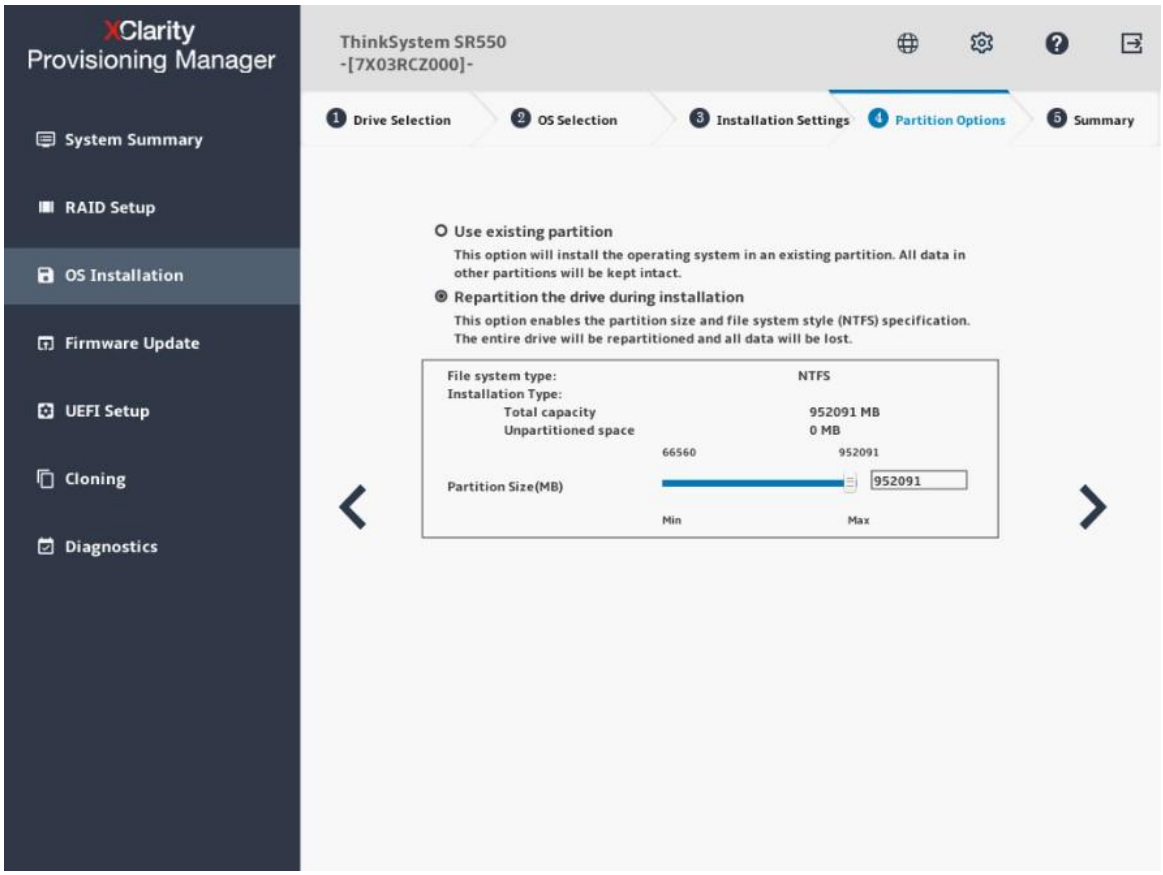


Components: You can select one or more components for installation according to your requirements.

Include run-once commands: If you want to run specified commands at the end of the installation process, select the check box. A command-type area is displayed. Type one command and click **Add**. The command is added to the command list. You can add five commands at most. If you want to remove a certain command, select it and click **Remove**. The commands in the command list will be run one time only and in the order you type them.

4. **Partition Options**

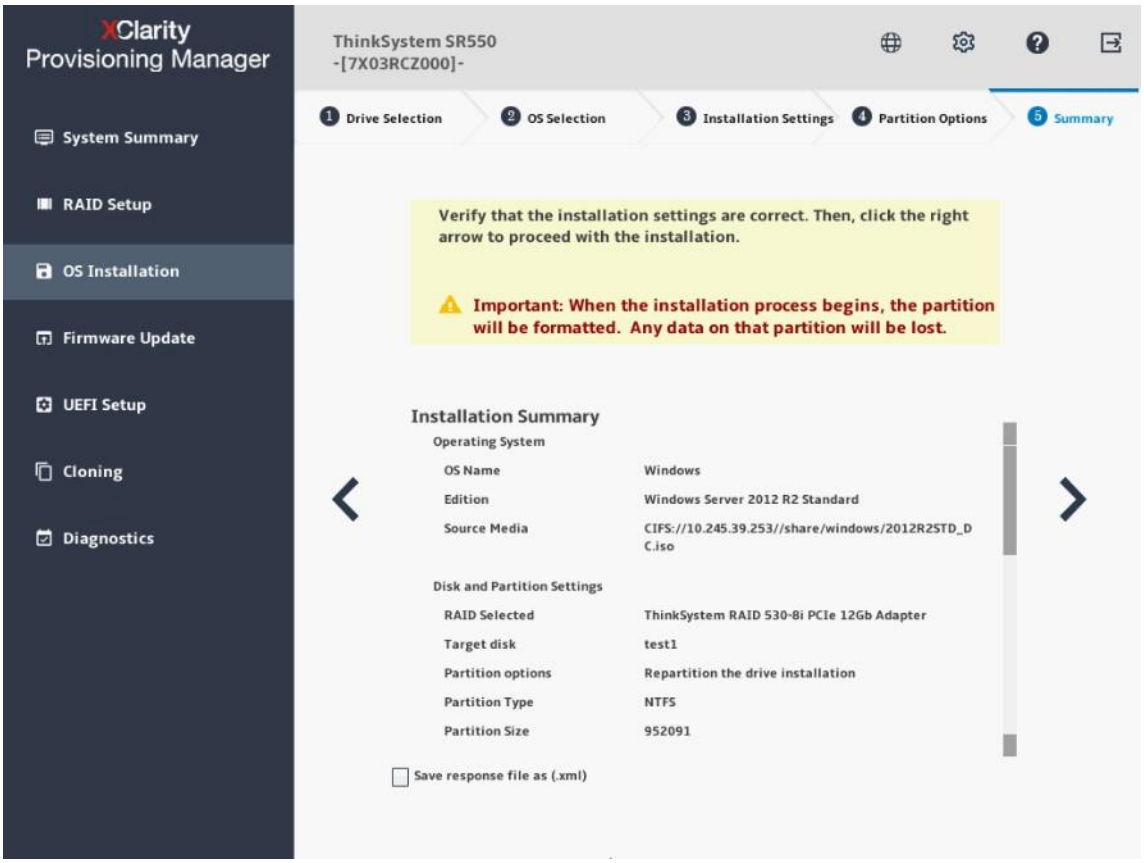
Figure 6. Partition Options step (for Windows)



If no existing partition is detected on the drive, select **Repartition the drive during installation**.

5. **Summary**

Figure 7. Summary step (for Windows)



Ex. No: 2 Illustrate UNIX commands and Shell Programming

Aim :

To write a C program to simulate basic Unix commands like ls,grep,cp.

Simulation of ls command

Algorithm :

1. Include necessary header files for manipulating directory.
2. Declare and initialize required objects.
3. Read the directory name form the user
4. Open the directory using opendir() system call and report error if the directory is not available
5. Read the entry available in the directory
6. Display the directory entry (ie., name of the file or sub directory.
7. Repeat the step 6 and 7 until all the entries were read.

```
/* 1. Simulation of ls command */
#include<dirent.h>
#include<stdio.h>
main()
{
char dirname[10];
DIR *p;
struct dirent *d;
printf("Enter directory name ");
scanf("%s",dirname);
p=opendir(dirname);
if(p==NULL)
{
perror("Cannot find dir.");
exit(-1);
}
while(d=readdir(p))
printf("%s\n",d->d_name);
}
```

OUTPUT:

```
enter directory name iii
.
..f2
f1
```

2. Simulation of grep command.

Algorithm :

1. Include necessary header files
2. Make necessary declarations
3. Read the file name from the user and open the file in the read only mode.
4. Read the pattern from the user.

5. Read a line of string from the file and search the pattern in that line.
6. If pattern is available, print the line.
7. Repeat the step 4 to 6 till the end of the file.

```
/* 2. Simulation of grep command */
#include<stdio.h>
#include<string.h>
main()
{
char fn[10],pat[10],temp[200];
FILE *fp;
printf("\n Enter file name : ");
scanf("%s",fn);
printf("Enter the pattern to be searched : ");
scanf("%s",pat);
fp=fopen(fn,"r");
while(!feof(fp))
{
fgets(temp,1000,fp);
if(strstr(temp,pat))
printf("%s",temp);
}
fclose(fp);
}
```

OUTPUT:

```
enter file name: file4
enter the pattern to be searched: roll
roll no percentage grade
```

3. Simulation of cp command

Algorithm:

1. Include required header file
2. Make necessary declarations
3. Read the source and destination file names from the user.
4. Using read() system call, read the content of the file to the buffer.
5. Uing write() system call, write the buffer content to the destination file.
6. Close the opened files.

```
/* using - open ,read and write system calls
file copy operation*/
#include<stdio.h>
#include<fcntl.h>
main()
{
char buf[1000],fn1[10],fn2[10];
int fd1,fd2,n;
printf("Enter source file name ");
```

```
scanf("%s",fn1);
printf("Enter destination file name ");
scanf("%s",fn2);
fd1=open(fn1,O_RDONLY);
n=read(fd1,buf,1000);
fd2=open(fn2,O_CREAT|0666);
n=write(fd2,buf,n);//write file
close(fd1);
close(fd1);
}
```

Source file:vrscet.txt

To learn operating system

OUTPUT:

```
enter source file name
vrscet.txt
enter destination file name
cse
[it2-20@localhost ~]$ cat cse
```

To learn operating system

RESULT: Thus the program to simulate UNIX commands was executed successfully.

i) BASIC ARITHMETIC OPERATION USIG SHELL PROGRAMMIG

AIM: To write a shell program to solve arithmetic operation.

ALGORITHM :

Step 1 :Include the necessary header files.
Step 2 : get the input
Step 3 : perform the arithmetic calculation.
Step 4 : print the result.
Step 5 : stop the execution.

PROGRAMCODING:

```
echo  "enter  a value"
read   a
echo  "enter b value"
read   b
c=`expr $a  + $b`
echo  "sum:"$c
c=`expr $a  -  $b`
echo  "sub:"$c
c=`expr $a  \*   $b`
echo  "mul:"$c
c=`expr $a /   $b`
echo  "div:"$c
```

OUTPUT:

```
[2mecse25@rhes3linux ~]$ sh pg2.sh
Enter  theavale
10
Enter  b value
50
sum: 60
sub: -40
mul: 500
div: 0
```

RESULT: Thus the program to solve arithmetic operation was executed successfully.

ii) NUMBER CHECKIG USIG SHELL PROGRAM

AIM: To write a shell program to check whether the number is odd or even.

ALGORITHM :

Step 1 :Include the necessary header files.
Step 2 : get the input
Step 3 : perform the division by2.
Step 4 : print the result.
Step 5 :stop the execution.

PROGRAMCODING:

```
num="1 2 3 4 5 6 7 8"
for n  in  $num
do
q=`expr $n % 2`
if  [  $q  -eq  0  ]
then
echo  "even no"
continue
fi
echo  "odd no"
done
```

OUTPUT:

```
[2mecse25@rhes3linux ~]$ sh pg2.sh
odd no
even no
odd no
even no
odd no
even no
odd no
even no
```

RESULT: Thus the program to check whether the number is odd or even was executed successfully

iii) MULTIPLICATION TABLE USIG SHELL PROGRAM

AIM: To write a shell program to implement multiplication table.

ALGORITHM :

Step 1 :Include the necessary header files.
Step 2 : get the input
Step 3 : perform the multiplication .
Step 4 : print the result.
Step 5 :stop the execution.

PROGRAMCODING:

```
echo  "which table you want"
read   n
for((i=1;i<10;i++))
do
echo   $i   "*" $n  "="   `expr $i  \*   $n`
done
```

OUTPUT:

```
[2mecse25@rhes3linux ~]$ sh pg2.sh
which table you want
 5
 1 * 5 =5
 2 * 5 =10
 3 * 5 =15
 4 * 5 =20
 5 * 5 =25
 6 * 5 =30
 7 * 5 =35
 8 * 5 =40
 9 * 5 =45
```

RESULT: Thus the program to implement multiplication table was executed successfully.

iv) USING WHILE LOOPIN SHELL PROGRAM

AIM: To write a shell program to print the number from 1 to 10 using while loop.

ALGORITHM

Step 1 :Include the necessary header files.
Step 2 : Define the buffer size as 1024.
Step 3 : Get the filename which has been created already.
Step 4 : Open the file in read mode.
Step 5 :.Read the contents of the file and store it in the buffer.
Step 6 : Print the contents stored in the buffer.
Step 7 : Close the file.

PROGRAMCODING:

```
a=1
while  [ $a-lt 11 ]
do
echo "$a"
a=`expr$a +1`
done
```

OUTPUT:

```
[2mecse25@rhes3linux ~]$ sh pg2.sh
1
2
3
4
5
6
7
8
9
10
```

RESULT: Thus the program to print the number from 1 to 10 using while loop was executed successfully.

v) USINGIFSTATEMENT IN SHELL PROGRAMING

AIM: To write a shell program to use simple if statement.

ALGORITHM:

Step 1 :Include the necessary header files.
Step 2 : Define the buffer size as 1024.
Step 3 : Get the filename which has been created already.
Step 4 : Open the file in read mode.
Step 5 :.Read the contents of the file and store it in the buffer.
Step 6 : Print the contents stored in the buffer.
Step 7 : Close the file.

PROGRAMCODING:

```
for var1  in  1  2  3
do
for var2  in  0  5
do
if  [  $var1  -eq  2  -a  $var2  -eq  0 ]
then
continue
else
echo  "$var1  $var2"
fi
done
```

OUTPUT:

```
[2mecse25@rhes3linux ~]$ sh pg2.sh
1 0
1 5
2 5
3 0
3 5
```

RESULT: Thus the program to use simple if statement was executed successfully

vi) SIMPLE FUNCTION IN SHELL PROGRAMING

AIM: To write a shell program to add a two number using function.

ALGORITHM :

Step 1 :Include the necessary header files.
Step 2 : Define the buffer size as 1024.
Step 3 : Get the filename which has been created already.
Step 4 : Open the file in read mode.
Step 5 :.Read the contents of the file and store it in the buffer.
Step 6 : Print the contents stored in the buffer.
Step 7 : Close the file.

PROGRAMCODING:

```
add()
{
c=`expr$1 +$2`
echo"addition = $c"
}
add   5  10
```

OUTPUT:

```
[2mecse25@rhes3linux ~] sh pg2.sh
addition = 15
```

RESULT: Thus the program to add a two number using function was executed successfully

vii SWITCH STATEMENT IN SHELL PROGRAMING

AIM: To write a shell program to add a two number using function.

ALGORITHM :

Step 1 :Include the necessary header files.
Step 2 : Define the buffer size as 1024.
Step 3 : Get the filename which has been created already.
Step 4 : Open the file in read mode.
Step 5 :.Read the contents of the file and store it in the buffer.
Step 6 : Print the contents stored in the buffer.
Step 7 : Close the file.

PROGRAMCODING

```
ch='y'
while  [  $ch  = 'y']
do
echo  "enter your choice"
echo  "1  no of user loged on" echo  "2  print calender"
echo  "3  print date" read  d
case $d  in
1) who  | wc - l;;
2) cal  20;;
3) date;;
*) break;;
esac
echo  "do you wish to continue(y/n)"
read  ch
done
```

OUTPUT:

```
[2mecse25@rhes3linux ~]$ sh case2.sh
Enter you rchoice
1 no of user loged on
2 print calender
3 print date
Thu Apr4 11:18:20IST2013 do you wish to continue(y/n)
n
```

RESULT: Thus the program to add a two number using function was executed successfully

Ex. No: 3 Process Management using System Calls: Fork, Exit, Getpid, Wait, Close

Program : fork()

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
main(void)
{
        pid\_t pid = 0;

        pid = fork();
        if (pid == 0) {
                printf("I am the child.\\n");
        }
        if (pid > 0) {
                printf("I am the parent, the child is %d.\\n", pid);
        }
        if (pid < 0) {
                perror("In fork():");
        }

        exit(0);
}
```

Running this, we get:

```
kris@linux:/tmp/kris> make probe1
cc     probe1.c  -o probe1
kris@linux:/tmp/kris> ./probe1
I am the child.
```

I am the parent, the child is 16959.

Program : wait()

```
#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>


#include <sys/types.h>

#include <sys/wait.h>


main(void) {

	pid\_t pid = 0;

	int   status;


	pid = fork();

	if (pid == 0) {

		printf("I am the child.\\n");

		sleep(10);

		printf("I am the child, 10 seconds later.\\n");

	}

	if (pid > 0) {

		printf("I am the parent, the child is %d.\\n", pid);

		pid = wait(&status);

		printf("End of process %d: ", pid);

		if (WIFEXITED(status)) {

			printf("The process ended with exit(%d).\\n", WEXITSTATUS(status));

		}

		if (WIFSIGNALED(status)) {

			printf("The process ended with kill -%d.\\n", WTERMSIG(status));

		}
```

```
        }
        if (pid < 0) {
                perror("In fork():");
        }

        exit(0);
}
```

And the runtime protocol:

```
kris@linux:/tmp/kris> make probe2
cc     probe2.c  -o probe2
kris@linux:/tmp/kris> ./probe2
I am the child.
I am the parent, the child is 17399.
I am the child, 10 seconds later.
End of process 17399: The process ended with exit(0).
```

Program : exec()

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

#include <sys/types.h>
#include <sys/wait.h>

main(void) {
```

```
        pid\_t pid = 0;
        int   status;


        pid = fork();
        if (pid == 0) {
                printf("I am the child.\\n");
                execl("/bin/ls", "ls", "-l", "/tmp/kris", (char \*) 0);
                perror("In exec(): ");
        }
        if (pid > 0) {
                printf("I am the parent, and the child is %d.\\n", pid);
                pid = wait(&status);
                printf("End of process %d: ", pid);
                if (WIFEXITED(status)) {
                        printf("The process ended with exit(%d).\\n", WEXITSTATUS(status));
                }
                if (WIFSIGNALED(status)) {
                        printf("The process ended with kill -%d.\\n", WTERMSIG(status));
                }
        }
        if (pid < 0) {
                perror("In fork():");
        }


        exit(0);
}
```

The runtime protocol:

```
kris@linux:/tmp/kris> make probe3
```

```
cc     probe3.c   -o probe3


kris@linux:/tmp/kris> ./probe3
I am the child.
I am the parent, the child is 17690.
total 36
-rwxr-xr-x 1 kris users 6984 2007-01-05 13:29 probe1
-rw-r--r-- 1 kris users  303 2007-01-05 13:36 probe1.c
-rwxr-xr-x 1 kris users 7489 2007-01-05 13:37 probe2
-rw-r--r-- 1 kris users  719 2007-01-05 13:40 probe2.c
-rwxr-xr-x 1 kris users 7513 2007-01-05 13:42 probe3
-rw-r--r-- 1 kris users  728 2007-01-05 13:42 probe3.c
End of process 17690: The process ended with exit(0).
```

Ex. No: 4 Write C programs to implement the various CPU Scheduling Algorithms

A) ROUND ROBIN SCHEDULING

AIM: To write the program to simulate the Round Robin program.

ALGORITHM:

Step 1:Initialize all the structure elements

Step 2: Receive inputs from the user to fill process id, burst time and arrival time.

Step 3: Calculate the waiting time for all the process id.

i) The waiting time for first instance of a process is calculated as:

a[i].wait time=count + a[i].arrive

ii) The waiting time for the rest of the instances of the process is calculated as:

a) Ifthetimequantumisgreaterthantheremainingbursttimethenwaitingtime is calculated as:

a[i].wait time=count + tq

b) Else if the time quantum is greater than the remaining burst time then waiting time is calculated as:

a[i].wait time=count- remaining burst time

Step 4: Calculate the average waiting time and average turn around time

Step 5: Print the results of the step 4.

PROGRAM:

```
#include<stdio.h>
void main()
{
int i,tbt=0,nop,ts=0,flag[20], rem[20];
int from,wt[20],tt[20],b[20], twt=0,ttt=0;
int dur;
float awt,att;
printf("Enter no. of Processes: ");
scanf("%d",&nop);
printf("Enter the time slice: "); scanf("%d",&ts);
printf("Enter the Burst times..\n");
for(i=0;i<nop;i++)
{
wt[i]=tt[i]=0;
printf("P%d\t: ",i+1); scanf("%d",&b[i]);
```

```
rem[i]=b[i];
tbt+=b[i];
flag[i]=0;
}
from=0;
 i=0;
printf("\n\t Gantt Chart");
printf("\n  ProcessID\tFrom Time\tTo Time\n");
while(from<tbt)
{
if(!flag[i])
{
if(rem[i]<=ts)
{ dur=rem[i]; flag[i]=1; tt[i]=dur+from; wt[i]=tt[i]-b[i];
}
else
dur=ts;
printf("%7d%15d%15d\n",i+1, from,from+dur);
rem[i] -= dur;
from += dur;
}
i=(i+1)%nop;
}
for(i=0;i<nop;i++)
{ twt+=wt[i]; ttt+=tt[i];
}
printf("\n\n Process ID \t Waiting Time \t Turn Around Time");
for(i=0;i<nop;i++)
{
printf("\n\t%d\t\t%d\t\t%d",i+1,wt[i],tt[i]);
} awt=(float)twt/(float)nop; att=(float)ttt/(float)nop;
printf("\nTotal Waiting Time:%d",twt); printf("\nTotal Turn Around Time:%d",ttt); printf("\nAverage
Waiting Time:%.2f",awt);
printf("\nAverage Turn Around Time:%.2f\n",att);
}
```

OUTPUT:

```
[fosslab@fosslab ~]$ vi finallrr.c
[fosslab@fosslab ~]$ cc finallrr.c
[fosslab@fosslab ~]$ ./a.out
Enter no. of Processes: 3
Enter the time slice: 3
Enter the Burst times..
P1 : 24
P2 : 5
P3 : 3
 Gantt Chart
 ProcessID From Time To Time
    1          0          3
    2          3          6
    3          6          9
    1          9         12
    2         12         14
    1         14         17
    1         17         20
    1         20         23
    1         23         26
    1         26         29
    1         29         32

Process ID    Waiting Time    Turn Around Time
 1   8   32
 2   9   14
 3   6   9

 Total Waiting Time:23
 Total Turn Around Time:55
 Average Waiting Time:7.67
 Average Turn Around Time:18.33
```

RESULT: Thus the program to simulate the Round Robin program was executed successfully.

EX.NO:4) b) SHORTEST JOB FIRST.

AIM: To write a C program to implement the CPU scheduling algorithm for shortest job first.

ALGORITHM:

Step 1: Get the number of process.

Step 2: Get the id and service time for each process.

Step 3: Initially the waiting time of first short process as 0 and total time of first short is process the service time of that process.

Step 4: Calculate the total time and waiting time of remaining process.

Step5: Waiting time of one process is the total time of the previous process.

Step6: Total time of process is calculated by adding the waiting time and service time of each process.

Step7: Total waiting time calculated by adding the waiting time of each process.

Step 8: Total turn round time calculated by adding all total time of each process.

Step9: Calculate average waiting time by dividing the total waiting time by total number of process.

Step 10: Calculate average turnaround time by dividing the total waiting time by total number of process.

Step 11: Display the result.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
void main()
{
int et[20],at[10],n,i,j,temp,st[10],ft[10],wt[10],ta[10];
int totwt=0,totta=0;
float awt,ata;
char pn[10][10],t[10];
printf("Enter the number of process:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter process name, arrival time & execution time:");
scanf("%s%d%d",pn[i],&at[i],&et[i]);
}
```

```
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
if(et[i]<et[j])
{
temp=at[i];
at[i]=at[j];
at[j]=temp;
temp=et[i];
et[i]=et[j];
et[j]=temp;
strcpy(t,pn[i]);
strcpy(pn[i],pn[j]);
strcpy(pn[j],t);
}
}
for(i=0;i<n;i++)
{
if(i==0)
st[i]=at[i];
else
st[i]=ft[i-1];
wt[i]=st[i]-at[i];
ft[i]=st[i]+et[i];
ta[i]=ft[i]-at[i];
totwt+=wt[i];
totta+=ta[i];
}
awt=(float)totwt/n;
ata=(float)totta/n;
printf("\nPname\tarrivaltime\texecutiontime\twaitingtime\ttatime");
for(i=0;i<n;i++)
printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],wt[i],ta[i]);
printf("\nAverage waiting time is:%f",awt);
printf("\nAverage turnaroundtime is:%f",ata);
 }
```

OUTPUT:

```
[fosslab@fosslab ~]$ vi finalsjf.c
[fosslab@fosslab ~]$ cc finalsjf.c
[fosslab@fosslab ~]$ ./a.out
Enter the number of process: 3
Enter process name, arrival time & execution time: short 4 3
Enter process name, arrival time & execution time: long 5 2
Enter process name, arrival time & execution time: medium 2 5

Pname    arrivaltime  executiontime  waitingtimetatime
 long     5        2        0        2
 short     4       3        3        6
 medium  2        5        8       13

 Average waiting time is:3.666667
 Average turnaroundtime is:7.000000
```

RESULT: Thus the program to implement the CPU scheduling algorithm for shortest job first was executed successfully.

EX.NO: 4) C) FIRST COME FIRST SERVE

AIM:

To write and execute a C program to implement the First Come First Served Scheduling Algorithm.

ALGORITHM:

1. Start.
2. Call Getdata() to get process name, burst time and arrival time from the user.
3. Call fcfs() to perform the First Come First Served Scheduling Algorithm.
4. Stop.

ALGORITHM FOR Getdata() :

1. Start.
2. Get the number of processes from the user.
3. For each process get the process name, burst time and arrival time from the user.
4. Stop.

ALGORITHM FOR FCFS() :

1. Start.
2. Swap the processes according to their arrival times.
3. Call Calculate()
4. Call Gantt_chart().
5. Stop.

ALGORITHM FOR Gantt_chart ():

1. Start.
2. Display the all the process names.
3. Display the waiting time of each process.
4. Stop.

ALGORITHM FOR Calculate ():

1. Start.
2. Initialize the wait time and Turnaround Time of first process as 0.
3. For each process, do the following steps.
    a. Calculate the wait time of each process.
    b. Calculate total wait time.
    c. Calculate total turnaround time.
4. Calculate the average wait time and the average Turnaround Time.
5. Display the average Waiting and Turnaround time.
6. Stop.

PROGRAM:

```
#include<stdio.h>
int main()
{
int n,b[10],t=0,i,w=0,r=0,a=0; float  avg,avg1;
printf("\nEnter number of processes:");
scanf("%d",&n);
printf("\nEnter the burst times : \n");
for(i=1;i<=n;i++) scanf("%d",&b[i]);
printf("\n Gantt chart ");
for(i=1;i<=n;i++)
printf("P%d\t",i);
printf("\n\nProcess BurstTime WaitingTime TurnaroundTime\n");
for(i=1;i<=n;i++)
{
t=t+w;
r=r+b[i];
printf("P%d\t\t%d\t\t%d\t\t%d\t\t\n",i,b[i],w,r);
w=w+b[i];
a=a+r;
}
avg=(float)t/n;
avg1=(float)a/n;
printf("\n Average WaitingTime is %f",avg);
printf("\n Average TurnaroundTime is %f\n",avg1);
return(0);
}
```

OUTPUT:

```
 [fosslab@fosslab ~]$ vi fcfsfinal.c
 [fosslab@fosslab ~]$ cc fcfsfinal.c
 [fosslab@fosslab ~]$ ./a.out
 Enter number of processes:3
 Enter the burst times :
 3
 5
 7
```

Gantt chart P1 P2 P3

Process	BurstTime	WaitingTime	TurnaroundTime
P1	3	0	3
P2	5	3	8
P3	7	8	15

Average WaitingTime is 3.666667

Average TurnaroundTime is 8.666667

RESULT: Thus the program to implement the First Come First Served Scheduling Algorithm was executed successfully.

Ex No: 4) D) PRIORITY SCHEDULING

AIM:

To write and execute a C program to implement the Priority Scheduling Algorithm.

ALGORITHM:

1. Start.
2. Call Getdata () to get process name, burst time and arrival time from the user.
3. Call prior () to perform the Priority Scheduling Algorithm.
4. Stop.

ALGORITHM FOR Getdata ():

1. Start.
2. Get the number of processes from the user.
3. For each process get the process name, burst time and arrival time from the user.
4. Stop.

ALGORITHM FOR Prior() :

1. Start.
2. Store the burst time of each process in a temporary array.
3. Swap the processes according to their priorities (except the first process) and Calculate Tt as sum of Tt and burst time of the current process.
4. Iniitialise S[i] for each process as 'T'
5. Initialize the wait time and Turnaround Time of first process as 0.
6. Set w as w+B[1] and S[1] as F.
7. While w<Tt, do the following:
   a. Set i as 2.
   b. While i<=n, do the following:
      i. If S[i]='T' and A[i]<=t, do the following:
         1) Set waiting time of i^{th} process as w and S[i] as F.
         2) Set w as w+B[i] and t as w.
         3) Set i as 2.
      ii. Else increment i by 1.
8. For each process, do the following steps.
   a. Calculate total wait time, Twt, as the sum of Twt and the difference between the wait time and arrival time of the current process.
   b. Calculate total turnaround time, Ttt, as the difference between sum of Ttt, wait time and burst time and the arrival time of the current process.
9. Calculate the average wait time.
10. Calculate the average Turnaround Time.

11. Display the average Waiting and Turnaround time.
12. Call Gantt_chart().
13. Stop.

ALGORITHM FOR Gantt_chart() :

1. Start.
2. Display the all the process names.
3. Display the waiting time of each process.
4. Stop.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
void main()
{
int et[20],at[10],n,i,j,temp,p[10],st[10],ft[10],wt[10],ta[10];
int totwt=0,totta=0;
float awt,ata;
char pn[10][10],t[10];
printf("Enter the number of process:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter process name,arrivaltime,execution time & priority:");
scanf("%s%d%d%d",pn[i],&at[i],&et[i],&p[i]);
}
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
if(p[i]<p[j])
{
temp=p[i];
p[i]=p[j];
p[j]=temp;
temp=at[i];
at[i]=at[j];
```

```
at[j]=temp;
temp=et[i];
et[i]=et[j];
et[j]=temp;
strcpy(t,pn[i]);
strcpy(pn[i],pn[j]);
strcpy(pn[j],t);
}
}
for(i=0;i<n;i++)
{
if(i==0)
{
st[i]=at[i];
wt[i]=st[i]-at[i];
ft[i]=st[i]+et[i];
ta[i]=ft[i]-at[i];
}
else
{
st[i]=ft[i-1];
wt[i]=st[i]-at[i];
ft[i]=st[i]+et[i];
ta[i]=ft[i]-at[i];
}
totwt+=wt[i];
totta+=ta[i];
}
awt=(float)totwt/n;
ata=(float)totta/n;
printf("\nPname\tarrivaltime\texecutiontime\tpriority\twaitingtime\ttatime");
for(i=0;i<n;i++)
printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],p[i],wt[i],ta[i]);
printf("\n Average waiting time is:%f",awt);
printf("\n Average turnaround time is:%f",ata);
}
```

OUTPUT:

```
[fosslab@fosslab ~]$ vi finalpriority.c
[fosslab@fosslab ~]$ cc finalpriority.c
[fosslab@fosslab ~]$ ./a.out
Enter the number of process:3
Enter process name,arrivaltime,execution time & priority: long  4 3 2
Enter process name,arrivaltime,execution time & priority: short 1 4 1
Enter process name,arrivaltime,execution time & priority: medium 5 5 4

P name  arrivaltime   executiontime   priority  waitingtimetatime
short        1            4            1        0          4
long         4            3            2        1          4
medium       5            5            4        3          8

Average waiting time is:1.333333
Average turnaround time is:5.333333
```

RESULT: Thus the program to implement the Priority Scheduling Algorithm was executed successfully

Ex. No: 5 **Illustrate the inter process communication strategy**

AIM: To write a C program to implement shared memory and inter-process communication.

ALGORITHM:

Step 1: Start the Program

Step 2: Obtain the required data through char and int data types.

Step 3: Enter the filename,index block.

Step 4: Print the filename index loop.

Step 5: Fill is allocated to the unused index blocks

Step 6: This is allocated to the unused linked allocation.

Step 7: Stop the execution

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/ipc.h>
#include<sys/uio.h>
#include<sys/types.h>
main()
{
int pid,pfd[2],n,a,b,c;
if(pipe(pfd)==-1)
{
printf("\nError in pipe connection\n");
exit(1);
}
pid=fork();
if(pid>0)
{
printf("\nParent Process");\
printf("\n\n\tFibonacci Series");
```

```
printf("\nEnter the limit for the series:");
scanf("%d",&n);
close(pfd[0]);
write(pfd[1],&n,sizeof(n));
close(pfd[1]);
exit(0);
}
else
{
close(pfd[1]);
read(pfd[0],&n,sizeof(n));
printf("\nChild Process");
a=0;
b=1;
close(pfd[0]);
printf("\nFibonacci Series is:");
printf("\n\n%d\n%d",a,b);
while(n>2)
{

c=a+b;
printf("\n%d",c);
a=b;
b=c;
n--;
}
}
}
```

OUTPUT:

Parent Process

Fibonacci Series

Enter the limit for the series: 5

Child Process

Fibonacci Series is:

0

1

1

2

3

RESULT: Thus the program to implement shared memory and inter-process communication was executed successfull

Ex. No: 6 Implement mutual exclusion by Semaphore

AIM: To write a C program to implement the Producer & consumer Problem (Semaphore)

ALGORITHM:

Step 1: The Semaphore mutex, full & empty are initialized.

Step 2:In the case of producer process

i) Produce an item in to temporary variable.

ii) If there is empty space in the buffer check the mutex value for enters into the critical section.

iii) If the mutex valueis0, allow the producer to add value in the temporary variable to the buffer.

Step 3:In the case of consumer process

i) It should wait if the buffer is empty

ii) If there is any item in the buffer check formute x value, if the mutex==0,remove item from buffer

iii) Signal the mutex value and reduce the emptyvalueby1.

iv) Consume the item.

Step 4: Print the result

PROGRAM:

```
#include<stdio.h>
int mutex=1,full=0,empty=3,x=0;
main()
{
int n;
void producer();
void consumer();
int wait(int);
int signal(int);
printf("\n 1.producer\n2.consumer\n3.exit\n");
while(1)
{
printf(" \nenter ur choice");
scanf("%d",&n);
switch(n)
{
case 1:if((mutex==1)&&(empty!=0))
producer();
else
printf("buffer is full\n");
break;
case 2:if((mutex==1)&&(full!=0))
consumer();
else
```

```
printf("buffer is empty");
break;
case 3:exit(0);
break;
}
}
}
int wait(int s)
{
return(--s);
}
int signal(int s)
{
return (++s);
}
void producer()
{
mutex=wait(mutex);
full=signal(full);
empty=wait(empty);
x++;
printf("\n producer produces the items %d",x);
mutex=signal(mutex);
}
void consumer()
{
mutex=wait(mutex);
full=wait(full);
empty=signal(empty);
printf("\n consumer consumes the item %d",x);
x--;
mutex=signal(mutex);
}
```

OUTPUT:

```
[fosslab@fosslab ~]$ vi finalsemaphore.c
[fosslab@fosslab ~]$ cc finalsemaphore.c
[fosslab@fosslab ~]$ ./a.out
 1. producer
2. consumer
3. exit
enter ur choice 1
 producer produces the items 1
enter ur choice 2
 consumer consumes the item 1
enter ur choice 1
 producer produces the items 1
```

enter ur choice 1
producer produces the items 2
enter ur choice 2
consumer consumes the item 2
enter ur choice 1
producer produces the items 2
enter ur choice 2
consumer consumes the item 2
enter ur choice 2
consumer consumes the item 1
enter ur choice 2
buffer is empty
enter ur choice 1
producer produces the items 1
enter ur choice 3

RESULT: Thus the program to implement the Producer & consumer Problem (Semaphore)was executed successfully.

Ex. No: 7 Write C programs to avoid Deadlock using Banker's Algorithm

AIM:

To implement deadlock avoidance & detection by using Banker's Algorithm.

ALGORITHM:

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety.
9. or not if we allow the request.
10. stop the program.

PROGRAM:

```
#include<stdio.h>
struct da
{
   int max[10],a1[10],need[10],before[10],after[10];
 }p[10];
 void main()
 {
  int i,j,k,l,r,n,tot[10],av[10],cn=0,cz=0,temp=0,c=0;
printf("\n ENTER THE NO. OF PROCESSES:");
 scanf("%d",&n);
 printf("\n ENTER THE NO. OF RESOURCES:");
 scanf("%d",&r);
 for(i=0;i<n;i++)
 {
    printf("PROCESS %d \n",i+1);
    for(j=0;j<r;j++)
    {
     printf("MAXIMUM VALUE FOR RESOURCE %d:",j+1);
     scanf("%d",&p[i].max[j]);
    }
   for(j=0;j<r;j++)
    {
     printf("ALLOCATED FROM RESOURCE %d:",j+1);
     scanf("%d",&p[i].a1[j]);
     p[i].need[j]=p[i].max[j]-p[i].a1[j];
    }
 }
  for(i=0;i<r;i++)
```

```
 {
   printf("ENTER TOTAL VALUE OF RESOURCE %d:",i+1);
   scanf("%d",&tot[i]);
 }
 for(i=0;i<r;i++)
 {
   for(j=0;j<n;j++)
   temp=temp+p[j].a1[i];
   av[i]=tot[i]-temp;
   temp=0;
 }
 printf("\n\t RESOURCES  ALLOCATED   NEEDED   TOTAL  AVAIL");
 for(i=0;i<n;i++)
 {
   printf("\n P%d \t",i+1);
   for(j=0;j<r;j++)
   printf("%d",p[i].max[j]);
   printf("\t");
   for(j=0;j<r;j++)
   printf("%d",p[i].a1[j]);
   printf("\t");
   for(j=0;j<r;j++)
   printf("%d",p[i].need[j]);
   printf("\t");
   for(j=0;j<r;j++)
   {
    if(i==0)
    printf("%d",tot[j]);
   }
   printf("     ");
   for(j=0;j<r;j++)
  {
      if(i==0)
      printf("%d",av[j]);
  }
}
  printf("\n\n\t AVAIL  BEFORE\T AVAIL AFTER ");
  for(l=0;l<n;l++)
  {
   for(i=0;i<n;i++)
   {
  for(j=0;j<r;j++)
  {
    if(p[i].need[j] >av[j])
    cn++;
    if(p[i].max[j]==0)
    cz++;
```

```
   }
    if(cn==0 && cz!=r)
    {
      for(j=0;j<r;j++)
      {
  p[i].before[j]=av[j]-p[i].need[j];
  p[i].after[j]=p[i].before[j]+p[i].max[j];
  av[j]=p[i].after[j];
  p[i].max[j]=0;
      }
      printf("\n P %d \t",i+1);
       for(j=0;j<r;j++)
       printf("%d",p[i].before[j]);
       printf("\t");
       for(j=0;j<r;j++)
       printf("%d",p[i].after[j]);
   cn=0;
   cz=0;
   c++;
   break;
   }
   else
   {
     cn=0;cz=0;
   }
  }
}
 if(c==n)
 printf("\n THE ABOVE SEQUENCE IS A SAFE SEQUENCE");
    else
    printf("\n DEADLOCK OCCURED");
}
```

OUTPUT:

//TEST CASE 1:

```
 ENTER THE NO. OF PROCESSES:4
ENTER THE NO. OF RESOURCES:3
PROCESS 1
MAXIMUM VALUE FOR RESOURCE 1:3
MAXIMUM VALUE FOR RESOURCE 2:2
MAXIMUM VALUE FOR RESOURCE 3:2
ALLOCATED FROM RESOURCE 1:1
ALLOCATED FROM RESOURCE 2:0
ALLOCATED FROM RESOURCE 3:0

PROCESS 2
MAXIMUM VALUE FOR RESOURCE 1:6
```

MAXIMUM VALUE FOR RESOURCE 2:1
MAXIMUM VALUE FOR RESOURCE 3:3
ALLOCATED FROM RESOURCE 1:5
ALLOCATED FROM RESOURCE 2:1
ALLOCATED FROM RESOURCE 3:1

PROCESS 3
MAXIMUM VALUE FOR RESOURCE 1:3
MAXIMUM VALUE FOR RESOURCE 2:1
MAXIMUM VALUE FOR RESOURCE 3:4
ALLOCATED FROM RESOURCE 1:2
ALLOCATED FROM RESOURCE 2:1
ALLOCATED FROM RESOURCE 3:1

PROCESS 4
MAXIMUM VALUE FOR RESOURCE 1:4
MAXIMUM VALUE FOR RESOURCE 2:2
MAXIMUM VALUE FOR RESOURCE 3:2
ALLOCATED FROM RESOURCE 1:0
ALLOCATED FROM RESOURCE 2:0
ALLOCATED FROM RESOURCE 3:2
ENTER TOTAL VALUE OF RESOURCE 1:9
ENTER TOTAL VALUE OF RESOURCE 2:3
ENTER TOTAL VALUE OF RESOURCE 3:6

	RESOURCES	ALLOCATED	NEEDED	TOTAL	AVAIL
P1	322	100	222	936	112
P2	613	511	102		
P3	314	211	103		
P4	422	002	420		

	AVAIL BEFORE	AVAIL AFTER
P 2	010	623
P 1	401	723
P 3	620	934
P 4	514	936

THE ABOVE SEQUENCE IS A SAFE SEQUENCE
//TEST CASE:2
ENTER THE NO. OF PROCESSES:4
ENTER THE NO. OF RESOURCES:3
PROCESS 1
MAXIMUM VALUE FOR RESOURCE 1:3
MAXIMUM VALUE FOR RESOURCE 2:2
MAXIMUM VALUE FOR RESOURCE 3:2

ALLOCATED FROM RESOURCE 1:1
ALLOCATED FROM RESOURCE 2:0
ALLOCATED FROM RESOURCE 3:1

PROCESS 2
MAXIMUM VALUE FOR RESOURCE 1:6
MAXIMUM VALUE FOR RESOURCE 2:1
MAXIMUM VALUE FOR RESOURCE 3:3
ALLOCATED FROM RESOURCE 1:5
ALLOCATED FROM RESOURCE 2:1
ALLOCATED FROM RESOURCE 3:1

PROCESS 3
MAXIMUM VALUE FOR RESOURCE 1:3
MAXIMUM VALUE FOR RESOURCE 2:1
MAXIMUM VALUE FOR RESOURCE 3:4
ALLOCATED FROM RESOURCE 1:2
ALLOCATED FROM RESOURCE 2:1
ALLOCATED FROM RESOURCE 3:2

PROCESS 4
MAXIMUM VALUE FOR RESOURCE 1:4
MAXIMUM VALUE FOR RESOURCE 2:2
MAXIMUM VALUE FOR RESOURCE 3:2
ALLOCATED FROM RESOURCE 1:0
ALLOCATED FROM RESOURCE 2:0
ALLOCATED FROM RESOURCE 3:2
ENTER TOTAL VALUE OF RESOURCE 1:9
ENTER TOTAL VALUE OF RESOURCE 2:3
ENTER TOTAL VALUE OF RESOURCE 3:6

RESOURCES	ALLOCATED	NEEDED	TOTAL	AVAIL
P1	322	101	221	936	110
P2	613	511	102		
P3	314	212	102		
P4	422	002	420		

AVAIL BEFORE AVAIL AFTER
DEADLOCK OCCURRED

RESULT: Thus the program to implement deadlock avoidance & detection by using Banker's Algorithm was executed successfully.

Ex. No: 8 Write a C program to Implement Deadlock Detection Algorithm

AIM: To write a C program to implement Deadlock Detection Algorithm.

ALGORITHM:

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state
8. Stop the process.

Program:

```
#include<stdio.h>
#include<conio.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void input();
void show();
void cal();
int main()
{
        int i,j;
        printf("********** 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++)
                {
                        printf("P%d\t",dead[i]);
                }
        }
        else
        {
                printf("\nNo Deadlock Occur");
        }
}
```

Output

```
********** Deadlock Detection Algo ************
Enter the no of Processes       3
Enter the no of resource instances      3
Enter the Max Matrix
3 6 8
4 3 3
3 4 4
Enter the Allocation Matrix
3 3 3
2 0 3
1 2 4
Enter the available Resources
1 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 the Deadlock process are
P0      P1      P2
```

Result: Thus the program to implement Deadlock Detection Algorithm was executed successfully.

Ex. No : 9 Write C program to implement Threading

AIM: To write a C program to implement Threading &Synchronization

ALGORITHM:

Step 1: Start the Program
Step 2: Obtain the required data through char and in data types.
Step 3: Enter the filename, index block.
Step 4: Print the filename index loop.
Step 5: File is allocated to the unused index blocks
Step 6: This is allocated to the unused linked allocation.
Step 7: Stop the execution.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
void *functionC();
pthread_mutex_t mutex1=PTHREAD_MUTEX_INITIALIZER;
int counter=0;
void main()
{
int rc1,rc2;
pthread_t thread1, thread2;
if((rc1=pthread_create(&thread1,NULL, &functionC,NULL)))
printf("thread creation failesd:%d\n", rc1);
if((rc2=pthread_create(&thread2,NULL,&functionC,NULL)))
pthread_join(thread1, NULL);
pthread_join(thread2,NULL);
exit(EXIT_SUCCESS);
}
void *functionC()
{
pthread_mutex_lock(&mutex1);
counter++;
printf("counter value %d\n", counter);
pthread_mutex_unlock(&mutex1);

}
```

OUTPUT:

```
[fosslab@fosslab ~]$ vi mutex1.c
[fosslab@fosslab ~]$ cc -lpthread mutex1.c
[fosslab@fosslab ~]$ ./a.out
counter value 1
counter value 2
```

RESULT: Thus the program to implement Threading &Synchronization was executed successfully

Ex. No: 10 Implement the paging Technique using C program

AIM:

To write and execute a C program, to implement the paging technique in memory management.

ALGORITHM:

1. Start.
2. Get the base address for physical memory from the user.
3. Get the size of main memory and frame size from the user.
4. Get the size of the logical memory from the user.
5. Get the frame values from the page table.
6. Display the page table with index, frame number and valid bit.
7. Stop.

Program:

```
#include<stdio.h>
void main()
{
int memsize=15;
int pagesize,nofpage;
int p[100];
int frameno,offset;
int logadd,phyadd;
int i;
int choice=0;
printf("\nYour memsize is %d ",memsize);
printf("\nEnter page size:");
scanf("%d",&pagesize);

nofpage=memsize/pagesize;
```

```
for(i=0;i<nofpage;i++)
{
printf("\nEnter the frame of page%d:",i+1);
scanf("%d",&p[i]);
}
do
{
printf("\nEnter a logical address:");
scanf("%d",&logadd);
frameno=logadd/pagesize;
offset=logadd%pagesize;
phyadd=(p[frameno]*pagesize)+offset;
printf("\nPhysical address is:%d",phyadd);
printf("\nDo you want to continue(1/0)?:");
scanf("%d",&choice);
}while(choice==1);
}
```

OUTPUT:

Your memsize is 15

Enter page size:5

Enter the frame of page1:2

Enter the frame of page2:4

Enter the frame of page3:7

Enter a logical address:3

Physical address is:13

Do you want to continue(1/0)?:1

Enter a logical address:1

Physical address is:11

Do you want to continue(1/0)?:0

RESULT: Thus the program to implement the paging technique in memory management was executed successfully.

Ex. No: 11 Write C programs to implement the following Memory Allocation Methods

a. First Fit

b. Worst Fit

c. Best Fit

AIM: To Write a C program to simulate the following contiguous memory allocation techniques

a) Worst-fit b) Best-fit c) First-fit

PROGRAM

a) WORST-FIT

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - First Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
```

```
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
{
ff[i]=j;
break;
}
}
}
frag[i]=temp;
bf[ff[i]]=1;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

INPUT

Enter the number of blocks: 3

Enter the number of files: 2

Enter the size of the blocks:-

Block 1: 5

Block 2: 2

Block 3: 7

Enter the size of the files:-

File 1: 1

File 2: 4

OUTPUT

File No	File Size	Block No	Block Size	Fragment
1	1	1	5	4
2	4	3	7	3

b) Best-fit

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
static int bf[max],ff[max];
clrscr();
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
```

```
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
if(lowest>temp)
{
ff[i]=j;

lowest=temp;
}
}
```

```
}
frag[i]=lowest;
bf[ff[i]]=1;
lowest=10000;
}
printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
for(i=1;i<=nf && ff[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

INPUT

Enter the number of blocks: 3

Enter the number of files: 2

Enter the size of the blocks:-

Block 1: 5

Block 2: 2

Block 3: 7

Enter the size of the files:-

File 1: 1

File 2: 4

OUTPUT

File No	File Size	Block No	Block Size	Fragment
1	1	2	2	1
2	4	1	5	1

c) First-fit

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - Worst Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
```

```
{


for(j=1;j<=nb;j++)
{
if(bf[j]!=1) //if bf[j] is not allocated
{
temp=b[j]-f[i];
if(temp>=0)
if(highest<temp)
{
ff[i]=j;
highest=temp;
}
}
}
frag[i]=highest;
bf[ff[i]]=1;
highest=0;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

INPUT

```
Enter the number of blocks: 3
```

Enter the number of files: 2

Enter the size of the blocks:-

Block 1: 5

Block 2: 2

Block 3: 7

Enter the size of the files:-

File 1: 1

File 2: 4

OUTPUT

File No	File Size	Block No	Block Size	Fragment
1	1	3	7	6
2	4	1	5	1

RESULT: Thus the program to implement the Memory Allocation Methods First Fit, Worst Fit ,Best Fit was executed successfully.

Ex. No :12 Write C programs to implement the various Page Replacement Algorithms

AIM: To write a C program to implement page replacement FIFO (First In First Out) algorithm

ALGORITHM:

Step 1: Start the Program

Step 2: Obtain the required data through char and in data types.

Step 3:Enter the filename,index block.

Step 4: Print the filename index loop.

Step 5:Fill is allocated to the unused index blocks

Step 6: This is allocated to the unused linked allocation.

Step 7: Stop the execution

PROGRAM:

```
#include<stdio.h>
void main()
{
int a[5],b[20],n,p=0,q=0,m=0,h,k,i,q1=1;
char f='F';
printf("Enter the Number of Pages:");
scanf("%d",&n);
printf("Enter %d Page Numbers:",n);
for(i=0;i<n;i++)
scanf("%d",&b[i]);
for(i=0;i<n;i++)
{if(p==0)
{
if(q>=3)
q=0;
a[q]=b[i];
q++;
if(q1<3)
{
q1=q;
```

```
}
}
printf("\n%d",b[i]);
printf("\t");
for(h=0;h<q1;h++)
printf("%d",a[h]);
if((p==0)&&(q<=3))
{
printf("-->%c",f);
m++;
}
p=0;
for(k=0;k<q1;k++)
{
if(b[i+1]==a[k])
p=1;
}
}
printf("\nNo of faults:%d",m);
}
```

OUTPUT:

Input:

Enter the Number of Pages: 12

Enter 12 Page Numbers:

232152453252

22-->F

323-->F

223

1231-->F

5531-->F

2521-->F

4524-->F

5524

3324-->F

2324

5354-->F

2352-->F

No of faults:9

RESULT: Thus the program to implement page replacement FIFO (First In First Out) algorithm was executed successfully

EX.NO: 12)ii) PAGE REPLACEMENT ALGORITHMS -LRU

AIM: To write a C program to implement page replacement LRU (Least Recently Used) algorithm.

ALGORITHM:

Step 1: Start the Program

Step 2: Obtain the required data through char and int data types.

Step 3:Enter the filename,index block.

Step 4: Print the filename index loop.

Step 5:Fill is allocated to the unused index blocks

Step 6: This is allocated to the unused linked allocation.

Step 7: Stop the execution.

PROGRAM:

```
#include<stdio.h>
void main()
{
int g=0,a[5],b[20],p=0,q=0,m=0,h,k,i,q1=1,j,u,n;
char f='F';
printf("Enter the number of pages:");
scanf("%d",&n);
printf("Enter %d Page Numbers:",n);
for(i=0;i<n;i++)
scanf("%d",&b[i]);
for(i=0;i<n;i++)
{if(p==0)
{
if(q>=3)
q=0;
a[q]=b[i];
q++;
if(q1<3)
{
q1=q;
```

```
g=1;
}}
printf("\n%d",b[i]);
printf("\t");
for(h=0;h<q1;h++)
printf("%d",a[h]);
if((p==0)&&(q<=3))
{
printf("-->%c",f);
m++;
}
p=0;
g=0;
if(q1==3)
{
for(k=0;k<q1;k++)
{
if(b[i+1]==a[k])
p=1;
}
for(j=0;j<q1;j++)
{
u=0;
k=i;
while(k>=(i-1)&&(k>=0))
{
if(b[k]==a[j])
u++;
k--;
}
if(u==0)
```

```
q=j;
}}
else
{
for(k=0;k<q;k++)
{
if(b[i+1]==a[k])
p=1;
}}}
printf("\nNo of faults:%d",m);
 }
```

OUTPUT:

```
No of faults:9[fosslab@fosslab ~]$ vi finaslru.c
[fosslab@fosslab ~]$ cc finaslru.c
[fosslab@fosslab ~]$ ./a.out
Enter the number of pages: 12
Enter 12 Page Numbers:  2 3 2 1 5 2 4 5 3 2 5 2
22-->F
323-->F
223
1231-->F
5251-->F
2251
4254-->F
5254
3354-->F
2352-->F
5352
2352
 No of faults:7
```

RESULT: Thus the program to implement page replacement LRU (Least Recently Used) algorithm was executed successfully.

EX.NO: 12) iii) PAGE REPLACEMENT ALGORITHMS -LFU

AIM:

To implement page replacement algorithms Optimal (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

PROGRAM:

```
/*OPTIMAL(LFU) page replacement algorithm*/
#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 the program to implement page replacement algorithms Optimal (The page which is not used for longest time)was executed successfully

Ex. No: 13. a) **Write C programs to Implement the various File Organization Techniques**

AIM: To write a C program to implement File Organization concept using the technique Single level directory.

ALGORITHM:

Step 1: Start the Program

Step 2: Obtain the required data through char and int data types.

Step 3: Enter the filename, index block.

Step 4: Print the filename index loop.

Step 5:Fill is allocated to the unused index blocks

Step 6: This is allocated to the unused linked allocation.

Step 7: Stop the execution

PROGRAM:

```
#include<stdio.h>
main()
{
int master,s[20];
char f[20][20][20];
char d[20][20];
int i,j;
printf("enter number of directorios:");
scanf("%d",&master);
printf("enter names of directories:");
for(i=0;i<master;i++)
scanf("%s",&d[i]);
printf("enter size of directories:");
for(i=0;i<master;i++)
scanf("%d",&s[i]);
printf("enter the file names :");
for(i=0;i<master;i++)
for(j=0;j<s[i];j++)
```

```
scanf("%s",&f[i][j]);
printf("\n");
printf(" directory\tsize\tfilenames\n");
printf("*************************************************\n");
for(i=0;i<master;i++)
{
printf("%s\t\t%2d\t",d[i],s[i]);
for(j=0;j<s[i];j++)
printf("%s\n\t\t\t",f[i][j]);
printf("\n");
}
printf("\t\n");
}
```

OUTPUT:

```
[fosslab@fosslab ~]$ cc Finalsinglelevel.c
[fosslab@fosslab ~]$ ./a.out

enter number of directorios: 2
enter names of directories: short  long
enter size of directories: 2  3
enter the file names : as ad we er ty
 directorysize filenames
*************************
short    2  as
     ad

long     3 we
     er
     ty
```

RESULT: Thus the program to implement File Organization concept using the technique Single level directory was executed successfully.

EX.NO: 13 b) FILE ORGANIZATION TECHNIQUES: TWO LEVEL DIRECTORY

AIM: To write a C program to implement File Organization concept using the technique two-level directory.

ALGORITHM:

Step 1: Start the Program

Step 2: Obtain the required data through char and in data types.

Step 3: Enter the filename,index block.

Step 4: Print the filename index loop.

Step 5: File is allocated to the unused index blocks

Step 6: This is allocated to the unused linked allocation.

Step 7: Stop the execution

PROGRAM:

```
#include<stdio.h>
struct st
{
char dname[10];
char sdname[10][10];
char fname[10][10][10];
int ds,sds[10];
}dir[10];
void main()
{
int i,j,k,n;
printf("enter number of directories:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter directory %d names:",i+1);
scanf("%s",&dir[i].dname);
printf("enter size of directories:");
```

```
scanf("%d",&dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("enter subdirectory name and size:");
scanf("%s",&dir[i].sdname[j]);
scanf("%d",&dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
{
printf("enter file name:");
scanf("%s",&dir[i].fname[j][k]);
}
}
}
printf("\ndirname\t\tsize\tsubdirname\tsize\tfiles");
printf("\n*****************************************************\n");
for(i=0;i<n;i++)
{
printf("%s\t\t%d",dir[i].dname,dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("\t%s\t\t%d\t",dir[i].sdname[j],dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
printf("%s\t",dir[i].fname[j][k]);
printf("\n\t\t");
}
printf("\n");
}
}
```

OUTPUT:

```
[fosslab@fosslab ~]$ ./a.out
enter number of directories: 2
```

enter directory 1 names: short

enter size of directories: 1

enter subdirectory name and size: tooshort 1

enter file name: a

enter directory 2 names: long

enter size of directories: 1

enter subdirectory name and size: toolong 1

enter file name: aaaaaaa

dirnamesize subdirname size files

****

short 1 tooshort 1 a

long 1 toolong 1 aaaaaaa

RESULT: Thus the program to implement File Organization concept using the technique two-level directory was executed successfully

EX.NO: 13 c) FILE ORGANIZATION TECHNIQUES: HIERARCHICAL

AIM: To write a C program to implement File Organization concept using the technique hierarchical directory.

ALGORITHM:

Step 1: Start the Program

Step 2: Obtain the required data through char and in data types.

Step 3: Enter the filename,index block.

Step 4: Print the filename index loop.

Step 5: File is allocated to the unused index blocks

Step 6: This is allocated to the unused linked allocation.

Step 7: Stop the execution

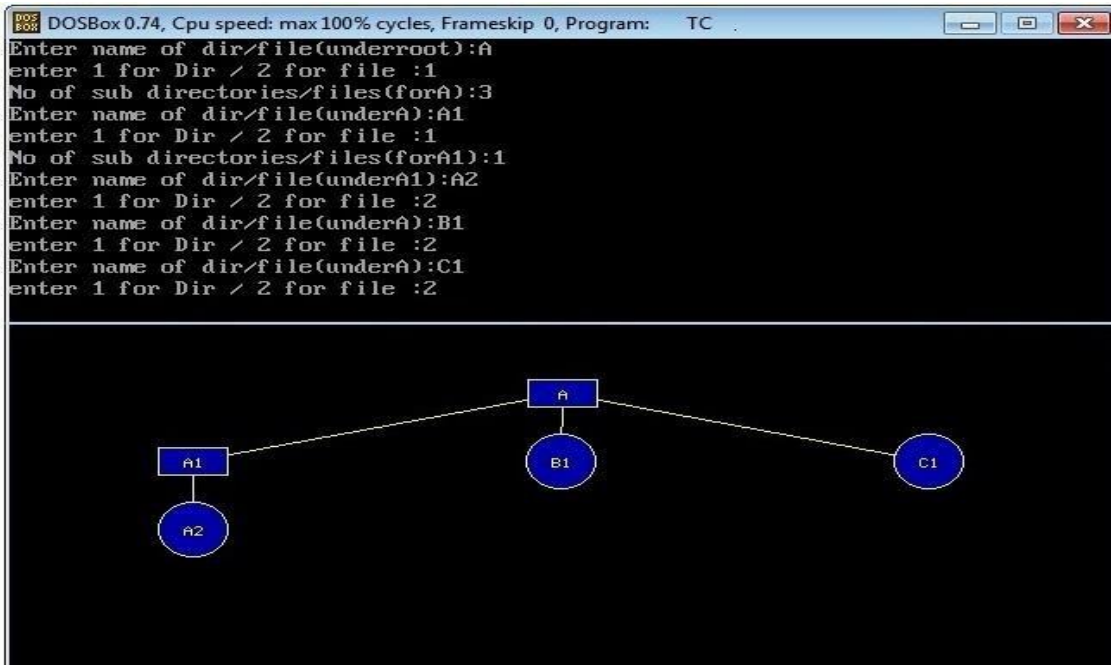
PROGRAM:

```
# include<stdio.h>
# include<graphics.h>
struct tree_element
{
char name[20];
int x,y,ftype,lx,rx,nc,level;
struct tree_element*link[5];
};
typedef struct tree_element node;
void main( )
{
int gd=DETECT,gm;node*root;
root=NULL;
clrscr( );
create(&root,0,"root",0,639,320);
clrscr();
initgraph(&gd,&gm,"c:\\tc\\BGI");
display(root);getch();closegraph();
}
create(node**root,int lev, char * dname, int lx , int rx, int x)
```

```
{
int i,gap;
if(*root==NULL)
{
(*root)=(node*)malloc(sizeof(node));
printf("Enter name of dir/file(under%s):",dname);
fflush(stdin);
gets((*root)->name);
printf("enter 1 for Dir / 2 for file :");
scanf("%d",&(*root)->ftype);
(*root)->level=lev;
(*root)->y=50+lev*50;
(*root)->x=x;(*root)->lx=lx;(*root)->rx=rx;
for(i=0;i<5;i++)
(*root)->link[i]=NULL;
if((*root)->ftype==1)
{
printf("No of sub directories/files(for%s):",(*root)->name);
scanf("%d",&(*root)->nc);
if((*root)->nc==0)gap=rx-lx;
else
gap=(rx-lx)/(*root)->nc;
for(i=0;i<(*root)->nc;i++)
create(&((*root)->link[i]),lev+1,(*root)->name,lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2);
}
else(*root)->nc=0;
}
}
display(node*root)
{
int i;
settextstyle(2,0,4);settextjustify(1,1);setfillstyle(1,BLUE);setcolor(14);
if( root!=NULL)
{
for(i=0;i<root->nc;i++)
{
```

```
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
}
if(root->ftype==1)bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0);
elsefillellipse(root->x,root->y,20,20);
outtextxy(root->x,root->y,root->name);
for(i=0;i<root->nc;i++)
{
display(root->link[i]);
}}}
```

OUTPUT:



RESULT:

Thus the program for implementing hierarchical directory structure has been completed.

EX.NO: 13 d) **FILE ORGANIZATION TECHNIQUES: DAG**

AIM: To write a C program to implement File Organization concept using the technique DAG directory.

ALGORITHM:

Step 1: Start the Program

Step 2: Obtain the required data through char and in data types.

Step 3: Enter the filename,index block.

Step 4: Print the filename index loop.

Step 5: File is allocated to the unused index blocks

Step 6: This is allocated to the unused linked allocation.

Step 7: Stop the execution

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<string.h>
struct tree_element
{
char name[20];
int x,y,ftype,lx,rx,nc,level;
struct tree_element *link[5];
};

typedef struct tree_element node;
typedef struct
{
  char from[20];
  char to[20];
}link;
link L[10];
int nofl;
node * root;

void main()
{
int gd=DETECT,gm;
root=NULL;
clrscr();
```

```
create(&root,0,"root",0,639,320);
read_links();
clrscr();
initgraph(&gd,&gm,"C:\\TC\\BGI");
draw_link_lines();
display(root);
getch();
closegraph();
}

read_links()
{
int i;
printf("how many links");
scanf("%d",&nofl);
for(i=0;i<nofl;i++)
{
printf("File/dir:");
fflush(stdin);
gets(L[i].from);
printf("user name:");
fflush(stdin);
gets(L[i].to);
}
}
draw_link_lines()
{
int i,x1,y1,x2,y2;
for(i=0;i<nofl;i++)
{
search(root,L[i].from,&x1,&y1);
search(root,L[i].to,&x2,&y2);
setcolor(LIGHTGREEN);
setlinestyle(3,0,1);
line(x1,y1,x2,y2);
setcolor(YELLOW);
setlinestyle(0,0,1);
}
}
search(node *root,char *s,int *x,int *y)
{
int i;
if(root!=NULL)
{
if(strcmpi(root->name,s)==0)
{
*x=root->x;
```

```
*y=root->y;
return;
}
else
{
for(i=0;i<root->nc;i++)
search(root->link[i],s,x,y);
}
}
}
create(node **root,int lev,char *dname,int lx,int rx,int x)
{
int i,gap;
if(*root==NULL)
{
(*root)=(node *)malloc(sizeof(node));
printf("enter name of dir/file(under %s):",dname);
fflush(stdin);
gets((*root)->name);
printf("enter 1 for dir/ 2 for file:");
scanf("%d",&(*root)->ftype);
(*root)->level=lev;
(*root)->y=50+lev*50;
(*root)->x=x;
(*root)->lx=lx;
(*root)->rx=rx;
for(i=0;i<5;i++)
(*root)->link[i]=NULL;
if((*root)->ftype==1)
{
printf("no of sub directories /files (for %s):",(*root)->name);
scanf("%d",&(*root)->nc);
if((*root)->nc==0)
gap=rx-lx;
else
gap=(rx-lx)/(*root)->nc;
for(i=0;i<(*root)->nc;i++)
create( & ( (*root)->link[i] ) , lev+1 ,
(*root)->name,lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2);
}
else (*root)->nc=0;
}
}
/* displays the constructed tree in graphics
mode */ display(node *root)
{
int i;
```

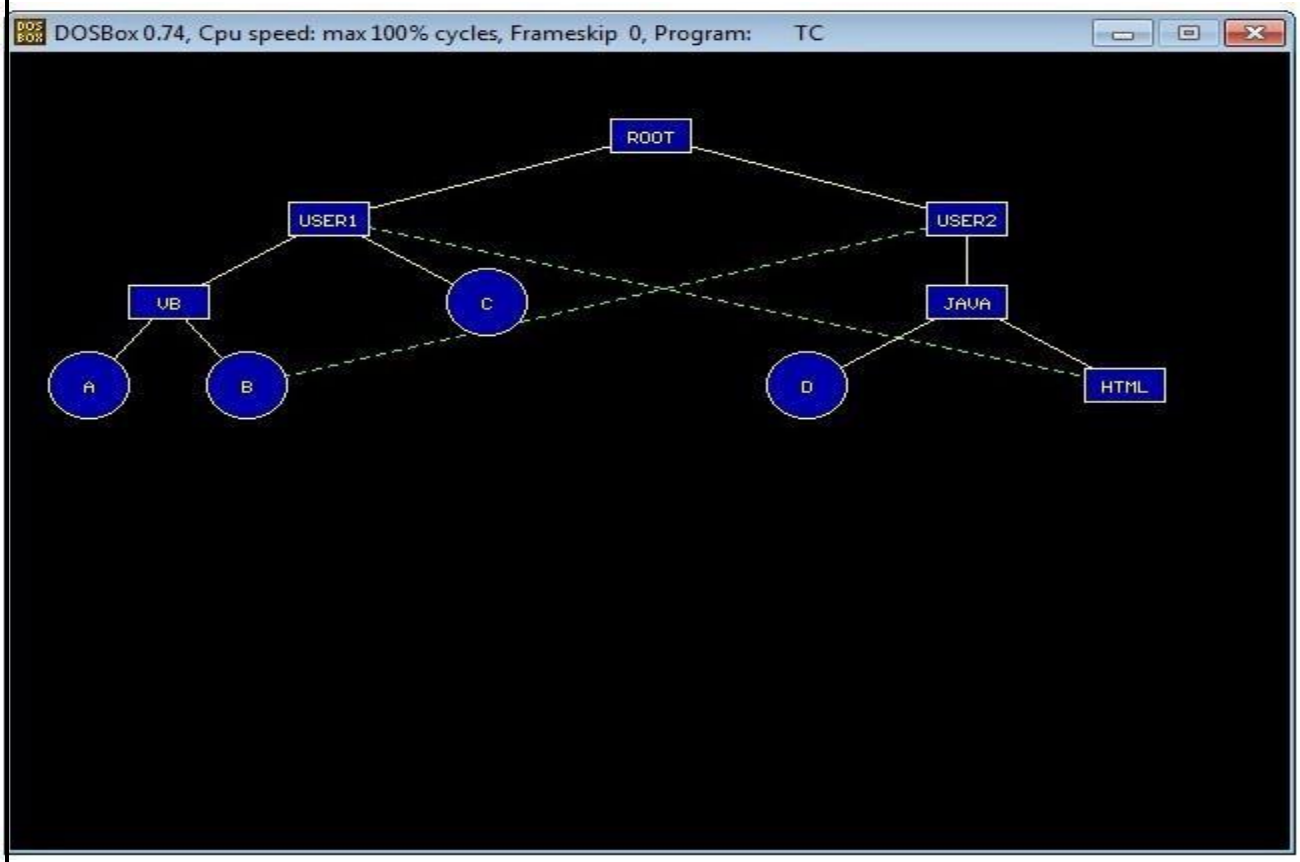
```
settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
setcolor(14); if(root
!=NULL)
{
for(i=0;i<root->nc;i++)
{
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
}
if(root->ftype==1) bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0);
else
fillellipse(root->x,root->y,20,20);
outtextxy(root->x,root->y,root->name); for(i=0;i<root->nc;i++)
{
display(root->link[i]);
}}}
```

OUTPUT:

```
Enter Name of dir/file (under root): ROOT
Enter 1 for Dir / 2 For File : 1
No of subdirectories / files (for ROOT) :2
Enter Name of dir/file (under ROOT): USER 1
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for USER 1): 2
Enter Name of dir/file (under USER1): VB
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for VB): 2
Enter Name of dir/file (under VB): A
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under VB): B
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under USER1): C
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under ROOT): USER2
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for USER2): 1
Enter Name of dir/file (under USER2):JAVA
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for JAVA):2
Enter Name of dir/file (under JAVA):D
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under JAVA):HTML
Enter 1 for Dir /2 for file:1
```

No of subdirectories /files (for HTML):0
How many links:2
File/Dir: B
User Name: USER 2
File/Dir: HTML
User Name: USER1

Output:



RESULT

Thus directed acyclic graph has been executed in C program.

EX. NO : 14) i) Implement the following File Allocation Strategies using C programs

a. Sequential

b. Indexed

c. Linked

AIM:

To write and execute a C program, to implement sequential File Allocation Technique.

ALGORITHM:

1. Start.
2. Get the number size of process from the user.
3. Allocate memory for each process.
4. Display the result.
5. Stop.

PROGRAM:

```
#include<stdio.h>
main()
{
int f[50],i,st,j,len,c,k;
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;
}
```

OUTPUT:

```
[fosslab@fosslab ~]$ vi finalseq.c

[fosslab@fosslab ~]$ cc finalseq.c
```

```
[fosslab@fosslab ~]$ ./a.out
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
  if u want to enter more files?(y-1/n-0)0
```

RESULT: Thus the program to implement sequential File Allocation Technique was executed successfully

EX. NO : 14) ii) FILE ALLOCATION TECHNIQUE-INDEXED ALLOCATION

AIM: To write a C program to implement file Allocation concept using the technique indexed allocation Technique..

ALGORITHM:

Step 1: Start the Program

Step 2: Obtain the required data through char and int data types.

Step 3: Enter the filename, index block.

Step 4: Print the filename index loop.

Step 5: Fill is allocated to the unused index blocks

Step 6: This is allocated to the unused linked allocation.

Step 7: Stop the execution.

PROGRAM:

```
#include<stdio.h>
void main()
{
char a[10];
inti,ib,cib[10];
printf("\n enter the filename:");
scanf("%s",a);
printf("\n indexblock:"); scanf("%d",&ib); for(i=1;i<=5;i++)
{
printf("\n enter thechild of indexblock %d:",i);
scanf("%d",&cib[i]);
}
printf("\n the list of files\t indexblock\n");
printf("%s\t\t %d",a,ib);
printf("\n theabove fileutiltization indexblock of child blocks followin\t");
printf("\n");
for(i=1;i<=5;i++)
{
```

```
printf("%d\t\t",cib[i]);
}
printf("\n");
}
```

OUTPUT:

```
[fosslab@fosslab ~]$ vi finalindex.c
[fosslab@fosslab ~]$ cc finalindex.c
[fosslab@fosslab ~]$ ./a.out
  enter the file name:testing
  index block:19
  enter the child of index block 1: 9

  enter the child of index block 2:16
  enter the child of index block 3:1
  enter the child of index block 4:1
  enter the child of index block 5:25

     the list of files    index block
      testing      19
      the above file utiltization index block of child blocks following
       9    16   1   1   25
```

RESULT: Thus the program to implement file Allocation concept using the technique indexed allocation Technique was executed successfully

EX.NO : 14)iii) FILE ALLOCATION TECHNIQUE-LINKED ALLOCATION

AIM: To write a C program to implement file Allocation concept using the technique Linked List Technique.

ALGORITHM:

Step 1: Start the Program

Step 2: Obtain the required data through char and int data types.

Step 3: Enter the file name, starting block ending block.

Step 4: Print the free block using loop.

Step 5: for loop is created to print the file utilization of linked type of entered type.

Step 6: This is allocated to the unused linked allocation.

Step 7: Stop the execution.

PROGRAM:

```
#include<stdio.h>
void main()
{
char a[10];
inti,sb,eb,fb1[10];
printf("\n enter the filename:");
scanf("%s",a);
printf("\n Enter the starting block:");
scanf("%d",&sb);
printf("Enter the ending Block:");
scanf("%d",&eb);
for(i=0;i<5;i++)
{
printf("Enter the free block %d",i+1);
scanf("%d",&fb1[i]);
}
printf("\nFilename\t Startingblock\t Endingblock\n");
printf("%s\t\t %d\t\t %d",a,sb,eb);
```

```
printf("\n %sFile Utilization of Linked type  of following blocks:",a);
printf("\n %d->",sb);
for(i=0;i<5;i++)
{
printf("%d->",fb1[i]);
}
printf("%d\n",eb);
}
```

OUTPUT:

```
[fosslab@fosslab ~]$ vi finallinked.c
[fosslab@fosslab ~]$ cc finallinked.c
[fosslab@fosslab ~]$ ./a.out
 enter the file name: testing
Enter the starting block: 18
Enter the ending Block:26
Enter the free block 1 1 1
Enter the free block 2 4 5

Enter the free block 3 17
Enter the free block 4 34
Enter the free block 5 15
 File name     Starting block     Ending block
testing      18      26
 testing File Utilization of Linked type of following blocks:

 18->11->45->17->34->15->26
```

RESULT: Thus the program to implement file Allocation concept using the technique Linked List Technique was executed successfully.

Ex. no: 15 Write C programs for the implementation of various disk scheduling algorithm

AIM: To write a C program to implementation of various disk scheduling algorithm (FCFS, SSTF, Scan, C-scan, Look, C-look).

ALGORITHM:

Step 1: Start the Program.

Step 2: Get the values of resources and processes.

Step 3: Enter the Requests Sequence

Step 4: Enter initial head position out

Step 5: Display the all the process

Step 6: Display the total disk size

Step 7: Stop the execution.

Various disk scheduling algorithm:

1) FCFS (First come first serve)

2) SSTF (Short seek time first)

3) Scan/ Elevator Disk scheduling

4) C-scan disk scheduling

5) Look disk scheduling

6) C-look disk scheduling

C-Program of FCFS (First come first serve) Disk scheduling Algorithms

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,n,TotalHeadMoment=0,initial;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);

   // logic for FCFS disk scheduling

   for(i=0;i<n;i++)
   {
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
      initial=RQ[i];
   }

   printf("Total head moment is %d",TotalHeadMoment);
   return 0;
   }
```

Output:-

```
Enter the number of Request
8
```

Enter the Requests Sequence

95 180 34 119 11 123 62 64

Enter initial head position

50

Total head movement is 644

SSTF (Short seek time first)Disk scheduling Algorithms

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,n,TotalHeadMoment=0,initial,count=0;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);

   // logic for sstf disk scheduling

      /* loop will execute until all process is completed*/
   while(count!=n)
   {
      int min=1000,d,index;
      for(i=0;i<n;i++)
      {
        d=abs(RQ[i]-initial);
```

```
        if(min>d)
        {
            min=d;
            index=i;
        }

    }
    TotalHeadMoment=TotalHeadMoment+min;
    initial=RQ[index];
    // 1000 is for max
    // you can use any number
    RQ[index]=1000;
    count++;
  }

  printf("Total head movement is %d",TotalHeadMoment);
  return 0;
}
```

Output:-

```
Enter the number of Request
8
Enter Request Sequence
95 180 34 119 11 123 62 64
Enter initial head Position
50
Total head movement is 236
```

C-program of scan or elevator disk scheduling

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);
   printf("Enter total disk size\n");
   scanf("%d",&size);
   printf("Enter the head movement direction for high 1 and for low 0\n");
   scanf("%d",&move);

   // logic for Scan disk scheduling

      /*logic for sort the request array */
   for(i=0;i<n;i++)
   {
      for(j=0;j<n-i-1;j++)
      {
         if(RQ[j]>RQ[j+1])
         {
            int temp;
```

```
            temp=RQ[j];
            RQ[j]=RQ[j+1];
            RQ[j+1]=temp;
        }

    }
}

int index;
for(i=0;i<n;i++)
{
    if(initial<RQ[i])
    {
        index=i;
        break;
    }
}

// if movement is towards high value
if(move==1)
{
    for(i=index;i<n;i++)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }
    // last movement for max size
    TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
    initial = size-1;
```

```
        for(i=index-1;i>=0;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }
    }
    // if movement is towards low value
    else
    {
        for(i=index-1;i>=0;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }
        //  last movement for min size
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
        initial =0;
        for(i=index;i<n;i++)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];

        }
    }

    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```

Output:-

Enter the number of Request

8

Enter the Requests Sequence

95 180 34 119 11 123 62 64

Enter initial head position

50

Enter total disk size

200

Enter the head movement direction for high 1 and for low 0

1

Total head movement is 337

C program for C-SCAN disk Scheduling algorithm

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);
   printf("Enter total disk size\n");
   scanf("%d",&size);
   printf("Enter the head movement direction for high 1 and for low 0\n");
```

```
scanf("%d",&move);

// logic for C-Scan disk scheduling

   /*logic for sort the request array */
for(i=0;i<n;i++)
{
   for( j=0;j<n-i-1;j++)
   {
      if(RQ[j]>RQ[j+1])
      {
         int temp;
         temp=RQ[j];
         RQ[j]=RQ[j+1];
         RQ[j+1]=temp;
      }

   }
}

int index;
for(i=0;i<n;i++)
{
   if(initial<RQ[i])
   {
      index=i;
      break;
   }
}
```

```
// if movement is towards high value
if(move==1)
{
   for(i=index;i<n;i++)
   {
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
      initial=RQ[i];
   }
   //  last movement for max size
   TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
   /*movement max to min disk */
   TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
   initial=0;
   for( i=0;i<index;i++)
   {
       TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
       initial=RQ[i];

   }
}
// if movement is towards low value
else
{
   for(i=index-1;i>=0;i--)
   {
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
      initial=RQ[i];
   }
```

```
        // last movement for min size
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
        /*movement min to max disk */
        TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
        initial =size-1;
        for(i=n-1;i>=index;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];

        }
    }

    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```

Output:-

```
Enter the number of Request
8
Enter the Requests Sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
Enter total disk size
200
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 382
```

C- program of look disk scheduling in operating system

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);
   printf("Enter total disk size\n");
   scanf("%d",&size);
   printf("Enter the head movement direction for high 1 and for low 0\n");
   scanf("%d",&move);

   // logic for look disk scheduling

      /*logic for sort the request array */
   for(i=0;i<n;i++)
   {
      for(j=0;j<n-i-1;j++)
      {
         if(RQ[j]>RQ[j+1])
         {
            int temp;
            temp=RQ[j];
```

```
            RQ[j]=RQ[j+1];
            RQ[j+1]=temp;
        }
    }
}

int index;
for(i=0;i<n;i++)
{
    if(initial<RQ[i])
    {
        index=i;
        break;
    }
}

// if movement is towards high value
if(move==1)
{
    for(i=index;i<n;i++)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }

    for(i=index-1;i>=0;i--)
    {
         TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
         initial=RQ[i];
```

```
        }
    }
    // if movement is towards low value
    else
    {
        for(i=index-1;i>=0;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }

        for(i=index;i<n;i++)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
             initial=RQ[i];

        }
    }

    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```

Output:-

Enter the number of Request

8

Enter the Requests Sequence

95 180 34 119 11 123 62 64

Enter initial head position

50

Enter the head movement direction for high 1 and for low 0

1

Total head movement is 299

C-Program of C- Look Disk scheduling Algorithms

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);
   printf("Enter total disk size\n");
   scanf("%d",&size);
   printf("Enter the head movement direction for high 1 and for low 0\n");
   scanf("%d",&move);

   // logic for C-look disk scheduling

      /*logic for sort the request array */
   for(i=0;i<n;i++)
   {
```

```
    for( j=0;j<n-i-1;j++)
    {
        if(RQ[j]>RQ[j+1])
        {
            int temp;
            temp=RQ[j];
            RQ[j]=RQ[j+1];
            RQ[j+1]=temp;
        }

    }
}

int index;
for(i=0;i<n;i++)
{
    if(initial<RQ[i])
    {
        index=i;
        break;
    }
}

// if movement is towards high value
if(move==1)
{
    for(i=index;i<n;i++)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
```

```
        initial=RQ[i];
    }

    for( i=0;i<index;i++)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];

    }
}
// if movement is towards low value
else
{
    for(i=index-1;i>=0;i--)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }

    for(i=n-1;i>=index;i--)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];

    }
}

printf("Total head movement is %d",TotalHeadMoment);
return 0;
```

}

Output:-

Enter the number of Request

8

Enter the Requests Sequence

95 180 34 119 11 123 62 64

Enter initial head position

50

Enter the head movement direction for high 1 and for low 0

1

Total head movement is 322

RESULT:

Thus the program to implement the disk scheduling algorithm (FCFS, SSTF, Scan, C-scan, Look, C-look) was executed successfully

Ex.No:16. Install any guest operating system like Linux using VMware.

Process to Install guest OS from ISO images in VMware Workstation:

To install the OS from an ISO image in a virtual machine:

Save the ISO image file in any location accessible to your host. For example:

Windows: C:\Temp or %TEMP%

Linux: /tmp or /usr/tmp

Note: For best performance, place this image on the host computer's hard drive. However, to make the ISO image accessible to multiple users, you can also place the ISO image on a network share drive (Windows) or exported filesystem (Linux). If your OS install spans multiple discs, you need to use an ISO image of each disc and place them all of them in a location accessible to the host.

Create a new virtual machine. Go to File > New > Virtual Machine.

Select Typical to accept Workstation's recommendations for various settings (such as processors, RAM, and disk controller type). Select Custom if you want to select these options yourself.

On the Guest Operating System Installation screen, when prompted where to install from, select Installer disc image file (iso).

Click Browse, and navigate to the location where you saved the ISO image file.

Click Next, and proceed through the new virtual machine wizard.

Before you click Finish, to create the virtual machine, deselect Power on this virtual machine after creation.

Edit the virtual machine settings so that its virtual CD/DVD device is configured to use the ISO image rather than the physical CD/DVD drive:

Select the tab for the virtual machine you just created.

Click Edit virtual machine settings.

On the Hardware tab, select the CD/DVD drive.

On the right side:

Select Connect at power on.

Select Use ISO image file.

Click Browse and navigate to where you saved the ISO image file.

Click OK.

Power on the virtual machine.

The virtual machine boots from the ISO image, which contains the installation software for your guest OS. Follow the installation procedure for your guest OS. For more information, see the Guest Operating System Installation Guide.

Note: If the guest OS asks for the second CD/DVD, repeat step 8 to point the virtual CD/DVD device to the second ISO image.

When you are finished installing the guest OS, you can edit the virtual machine settings so that it is once more set to use the host computer's physical drive. You do not need to leave the drive set to connect at power on.

Prerequisites

Verify that the operating system is supported. See the online VMware Compatibility Guide on the VMware Web site.

See the VMware Guest Operating System Installation Guide for information on the guest operating system that you are installing.

Procedure

If you are installing the guest operating system from an installer disc, configure the virtual machine to use a physical CD-ROM or DVD drive and configure the drive to connect at power on.

Select the virtual machine and **select VM > Settings.**

On the Hardware tab, select CD/DVD drive.

Select Connect at power on.

(Remote virtual machine only) Select the location of the CD-ROM or DVD drive.

Select Use physical drive and select a the drive.

Click OK to save your changes.

If you are installing the guest operating system from an ISO image file, configure the CD/DVD drive in the virtual machine to point to the ISO image file and configure the drive to connect at power on.

Select the virtual machine and select VM > Settings.

On the Hardware tab**, select CD/DVD drive.**

Select Connect at power on.

(Remote virtual machine only) Select the location of the ISO image file.

Select Use ISO image file and browse to the location of the ISO image file.

Click OK to save your changes.

If you are installing the guest operating system from an installer disc, insert the disc in the CD-ROM or DVD drive.

Power on the virtual machine.

Follow the installation instructions provided by the operating system vendor.

If the operating system consists of multiple installer discs and you are prompted to insert the next disc, insert the next disc in the physical drive.

If the operating system consists of multiple ISO image files, select the image file for the next CD.

Select VM > Removable Devices > CD/DVD > Disconnect and disconnect from the current ISO image file.

Select VM > Removable Devices > CD/DVD > Settings and select the next ISO image file.

Select Connected and **click OK**.

Use the standard tools in the operating system to configure its settings.

EX.NO : 17 PROGRAM ILLUSTRATING I/O SYSTEM CALLS

AIM:

To write and execute a C program in Linux, using the I/O system calls namely open(), read(), write() and close().

ALGORITHM:

1. Open the input file, in read-only mode using open().
2. Create (if necessary) and Open an output file, in write-only or truncate mode using open().
3. Read the maximum of 20 bytes from the input file using read().
4. Write the content read to the output file using write().
5. Display the number of bytes read and written successfully.
6. Close both the input and output files using close().
7. Stop.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<fcntl.h>
int main()
  {
        int fp1,fp2,o,i;
        char *c;
        c=(char*)calloc(50,sizeof(char));
        fp1=open("input.txt",O_RDONLY);
        fp2=open("output.txt",O_CREAT|O_WRONLY|O_TRUNC);
        o=read(fp1,c,20);
        i=write(fp2,c,strlen(c));
        printf("\n Number of Bytes read = %d\n",o);
        printf("\n Number of Bytes written = %d\n",i);
        close(fp1);
```

```
        close(fp2);
}
```

INPUT FILE: input.txt

Operating systems

OUTPUT:

```
[student@localhost os]$ cc io.c
[student@localhost os]$ ./a.out
Number of Bytes read = 18
Number of Bytes written = 18
[student@localhost os]$
```

OUTPUT FILE: output.txt

Operating systems

RESULT:

Thus the C program in Linux that illustrated the use of various I/O system calls was executed successfully.

Ex. No: 18 SHARED MEMORY AND INTER PROCESS COMMUNICATION

AIM: To write a C program to implement shared memory and inter-process communication.

ALGORITHM:

Step 1: Start the Program

Step 2: Obtain the required data through char and int datatypes.

Step 3: Enter the filename,index block.

Step 4: Print the filename index loop.

Step 5: Fill is allocated to the unused index blocks

Step 6: This is allocated to the unused linked allocation.

Step 7: Stop the execution

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/ipc.h>
#include<sys/uio.h>
#include<sys/types.h>
main()
{
int pid,pfd[2],n,a,b,c;
if(pipe(pfd)==-1)
{
printf("\nError in pipe connection\n");
exit(1);
}
pid=fork();
if(pid>0)
{
```

```
printf("\nParent Process");\
printf("\n\n\tFibonacci Series");
printf("\nEnter the limit for the series:");
scanf("%d",&n);
close(pfd[0]);
write(pfd[1],&n,sizeof(n));
close(pfd[1]);
exit(0);
}
else
{
close(pfd[1]);
read(pfd[0],&n,sizeof(n));
printf("\nChild Process");
a=0;
b=1;
close(pfd[0]);
printf("\nFibonacci Series is:");
printf("\n\n%d\n%d",a,b);
while(n>2)
{

c=a+b;
printf("\n%d",c);
a=b;
b=c;
n--;
}
}
```

}

OUTPUT:

Parent Process

Fibonacci Series

Enter the limit for the series: 5

Child Process

Fibonacci Series is:

0

1

1

2

3

RESULT: Thus the program to implement shared memory and inter-process communication was executed successfully