Ex: NO 1 **INSTALLATION OF WINDOWS XP OPERATING SYSTEM**

Aim:

To install Windows XP Operating System.

Description

Windows XP is a versatile O.S. which took over various laggings in the earlier O.S. given by Microsoft. One major component which really improves the performance and working is the availability of various device drivers which really make device operation as simple as plug-n-play. If the minimum system requirement is not fulfilled then you will not be able to install this O.S. on to your machine, however, Windows 98 might serve the purpose. Windows XP is in huge demand globally, let us learn how to load it before making us to work on it.

System Requirements

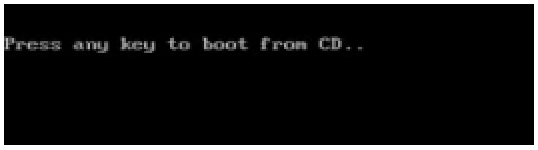
System Requirements	**Minimum**	**Recommended**
Processor	233MHz	300 MHz or higher
Memory	64 MB RAM	128 MB RAM or higher
Video adapter and Monitor	Super VGA (800 x 600) or higher resolution	
Hard drive disk free space	1.5 GB or higher	
Drives	CD-ROM drive or DVD drive	
Input Devices	Keyborad, Microsoft Mouse	
Sound	Speakers. Sound card, Head Phones	

Installation Steps

The following step by step procedure will help you to install Windows XP. The installation procedure is shown with the figure appears on your screen after doing a step.

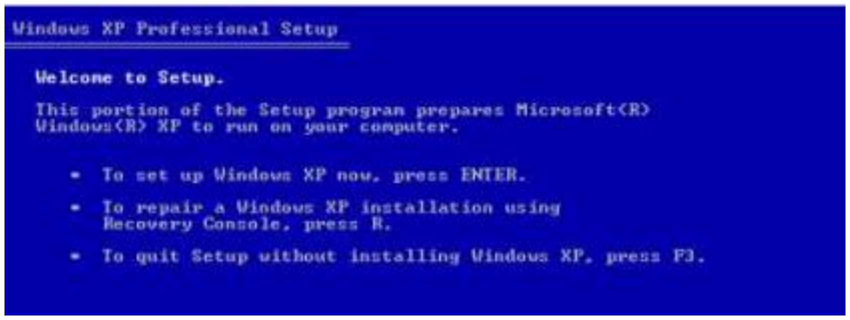
1) Insert the Windows XP CD into your computer and restart.

2) If prompted to start from the CD, press SPACEBAR. If you miss the prompt (it only appears for a few seconds), restart your computer to try again.



```
Press any key to boot from CD..
```

3) Windows XP Setup begins. During this portion of setup, your mouse will not work, so you must use the keyboard. On the Welcome to Setup page, press ENTER.



```
Windows XP Professional Setup

  Welcome to Setup.

  This portion of the Setup program prepares Microsoft(R)
  Windows(R) XP to run on your computer.

      •  To set up Windows XP now, press ENTER.

      •  To repair a Windows XP installation using
         Recovery Console, press R.

      •  To quit Setup without installing Windows XP, press F3.
```

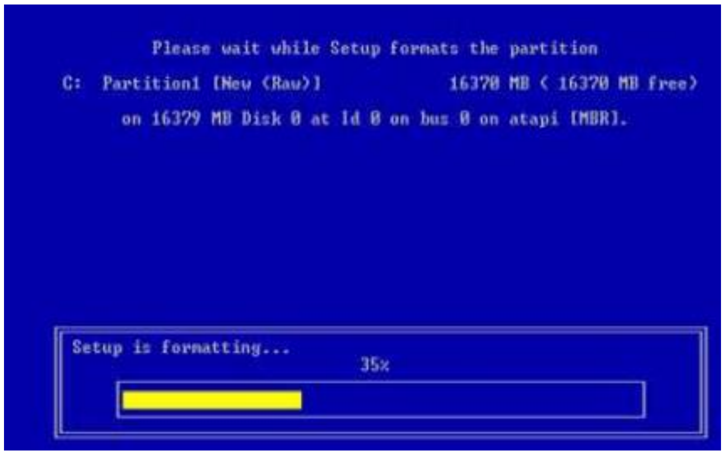
4) On the Windows XP Licensing Agreement page, read the licensing agreement. Press the PAGE DOWN key to scroll to the bottom of the agreement. Then press F8.

5) Next page enables you to select the hard disk drive on which Windows XP will be installed. Once you complete this step, all data on your hard disk drive will be removed and cannot be recovered. It is extremely important that you have a recent backup copy of your files before continuing. When you have a backup copy, press D, and then press L when prompted. This deletes your existing data.

6) Press ENTER to select Unpartitioned space, which appears by default.

7) Press ENTER again to select Format the partition using the NTFS file system, which appears by default.

8) Windows XP erases your hard disk drive using a process called formatting and then copies the setup files. You can leave your computer and return in 20 to 30 minutes



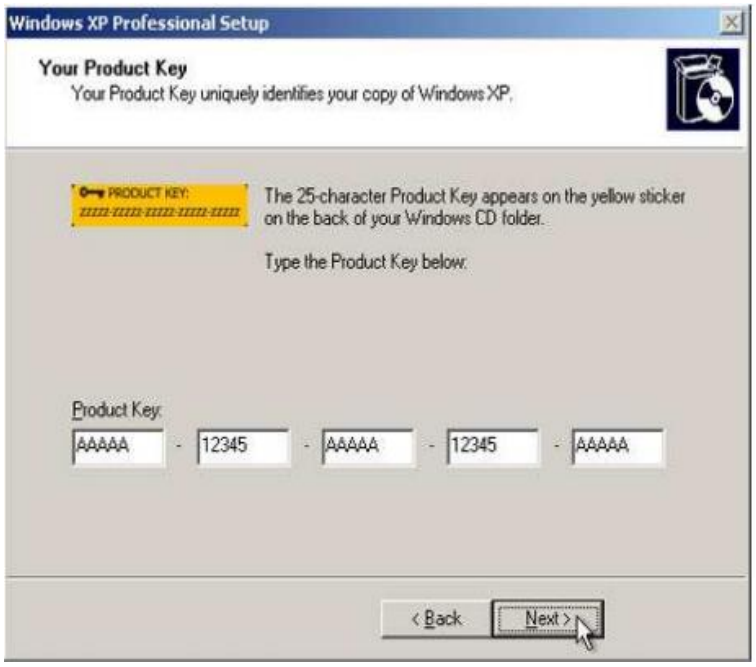
```
          Please wait while Setup formats the partition
  C:  Partition1 [New (Raw)]            16370 MB ( 16370 MB free)
        on 16379 MB Disk 0 at Id 0 on bus 0 on atapi [MBR].

  Setup is formatting...
                                 35%
```

9) Windows XP restarts and then continues with the installation process. From this point forward, you can use your mouse. Eventually, the Regional and Language Options page appears. Click Next to accept the default settings. If you are multilingual or prefer a language other than English, you can change language settings after setup is complete.

10) On the Personalize Your Software page, type your name and your organization name. Some programs use this information to automatically fill in your name when required. Then, click Next.

11) On the Your Product Key page, type your product key as it appears on your Windows XP CD case. The product key is unique for every Windows XP installation. Then, click Next.



Windows XP Professional Setup

Your Product Key

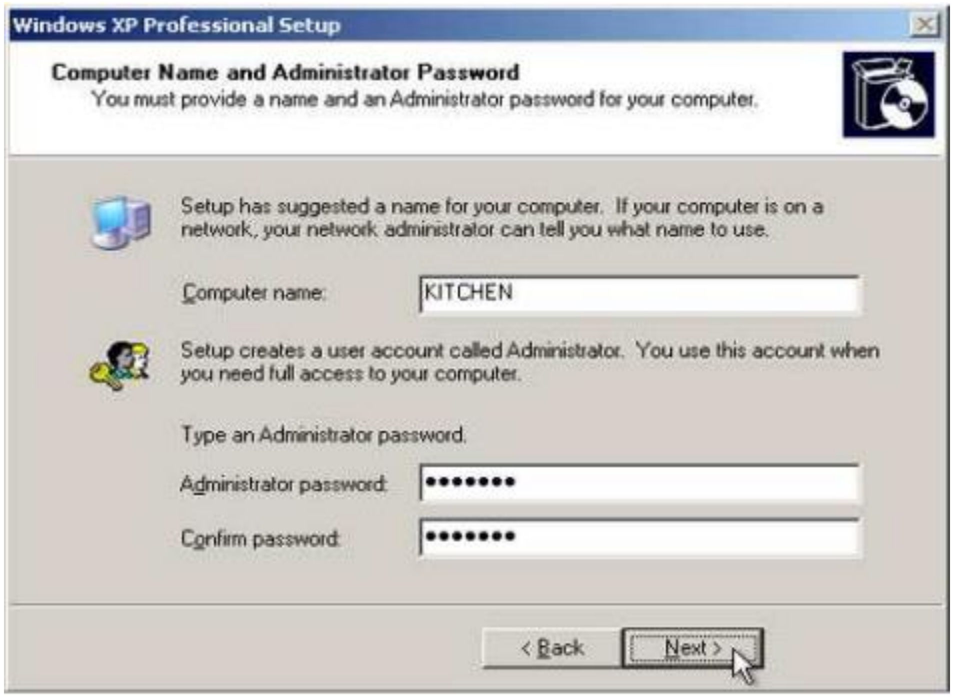
Your Product Key uniquely identifies your copy of Windows XP.

The 25-character Product Key appears on the yellow sticker on the back of your Windows CD folder.

Type the Product Key below:

Product Key: AAAAA - 12345 - AAAAA - 12345 - AAAAA

12) On the Computer Name and Administrator Password page, in the Computer name box, type a name that uniquely identifies your computer in your house, such as FAMILYROOM or TOMS. You cannot use spaces or punctuation. If you connect your computer to a network, you will use this computer name to find shared files and printers. Type a strong password that you can remember in the Administrator password box, and then retype it in the Confirm password box. Write the password down and store it in a secure place. Click Next.


Windows XP Professional Setup

Computer Name and Administrator Password

You must provide a name and an Administrator password for your computer.

Setup has suggested a name for your computer. If your computer is on a network, your network administrator can tell you what name to use.

Computer name: KITCHEN

Setup creates a user account called Administrator. You use this account when you need full access to your computer.

Type an Administrator password.

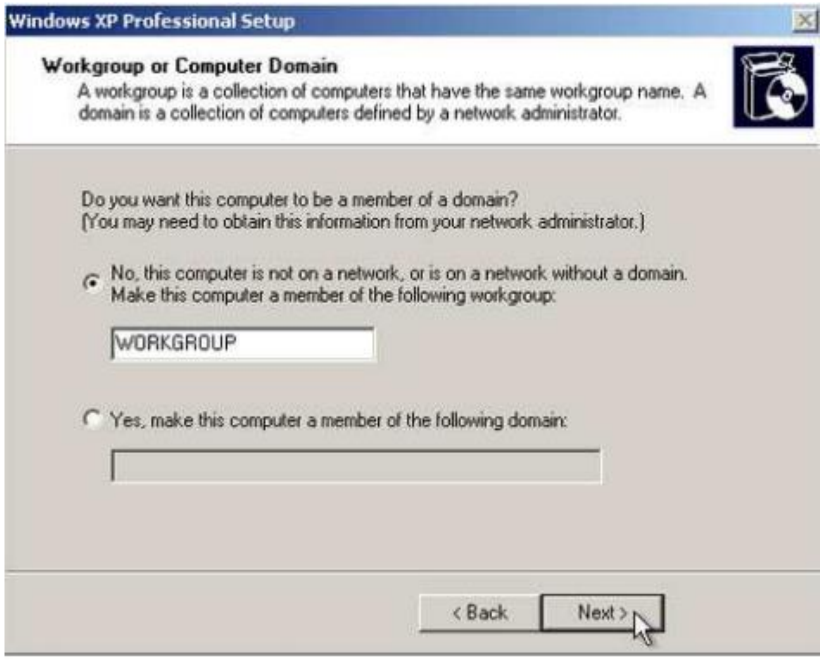
Administrator password: •••••••

Confirm password: •••••••

13) On the Date and Time Settings page, set your computer's clock. Then, click the Time Zone down arrow, and select your time zone. Click Next.

14) Windows XP will spend about a minute configuring your computer. On the Networking Settings page, click Next.

15) On the Workgroup or Computer Domain page, click Next.


Windows XP Professional Setup

Workgroup or Computer Domain

A workgroup is a collection of computers that have the same workgroup name. A domain is a collection of computers defined by a network administrator.

Do you want this computer to be a member of a domain?
(You may need to obtain this information from your network administrator.)

● No, this computer is not on a network, or is on a network without a domain. Make this computer a member of the following workgroup:

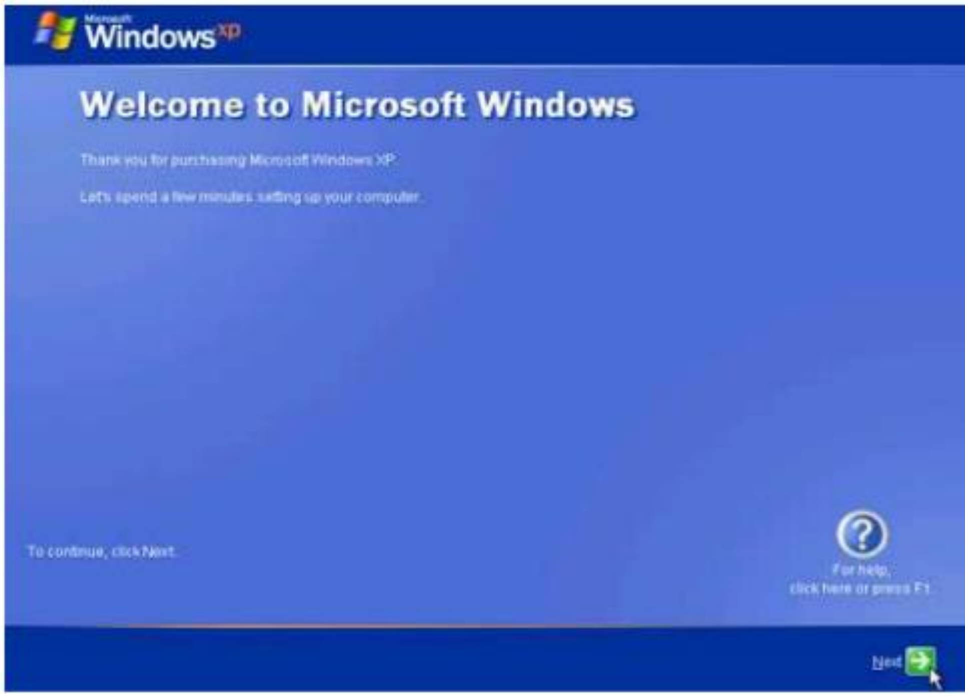
WORKGROUP

○ Yes, make this computer a member of the following domain:

16) Windows XP will spend 20 or 30 minutes configuring your computer and will automatically restart when finished. When the Display Settings dialog appears, click OK.

17) When the Monitor Settings dialog box appears, click OK.

18) The final stage of setup begins. On the Welcome to Microsoft Windows page, click Next.



Welcome to Microsoft Windows

Thank you for purchasing Microsoft Windows XP

Let's spend a few minutes setting up your computer

To continue, click Next

19) On the Help protect your PC page, click Help protect my PC by turning on Automatic Updates now. Then, click Next

20) Windows XP will then check if you are connected to the Internet:

If you are connected to the Internet, select the choice that describes your network connection on the Will this computer connect to the Internet directly, or through a network? page. If you're not sure, accept the default selection, and click Next.

21) If you use dial-up Internet access, or if Windows XP cannot connect to the Internet, you can connect to the Internet after setup is complete. On the How will this computer connect to the Internet? page, click Skip.

22) Windows XP Setup displays the Ready to activate Windows? page. If you are connected to the Internet, click Yes, and then click Next. If you are not yet connected to the Internet, click No, click Next, and then skip to step 24. After setup is complete, Windows XP will automatically remind you to activate and register your copy of Windows XP.

23) On the Ready to register with Microsoft? page, click Yes, and then click Next

24) On the Collecting Registration Information page, complete the form. Then, click Next.

25) On the Who will use this computer? page, type the name of each person who will use the computer. You can use first names only, nicknames, or full names. Then click Next. To add users after setup is complete or to specify a password to keep your account private, read Create and customize user accounts.

26) On the Thank you! page, click Finish.

27) Congratulations! Windows XP setup is complete. You can log on by clicking your name on the logon screen. If you've installed Windows XP on a new computer or new hard disk drive, you can now use the File and Settings Transfer Wizard to copy your important data to your computer or hard disk drive.

RESULT:

Thus the Windows Operating System is installed and Executed successfully.

EX. NO. 2A **BASICS OF UNIX COMMANDS**

System calls used :

1. fork( )

Used to create new processes. The new process consistsof a copy of the address space of the original process. The value of process id for the child process is zero, whereas the value of process id for the parent is an integer value greater than zero.

Syntax : fork( )

2.execlp( )

Used after the fork() system call by one of the two processes to replace the process‟ memory space with a new program. It loads a binary file into memory destroying the memory image of the program containing the execlp system call and starts its execution.The child process overlays its address space with the UNIX command /bin/ls using the execlp system call.

Syntax : execlp( )

3. wait( )

The parent waits for the child process to complete using the wait system call. The wait system call returns the process identifier of a terminated child, so that the parentcan tell which of its possibly many children has terminated.

Syntax : wait(NULL)

4. exit( )

A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit system call. At that point, the process may return data (output) to its parent process (via the wait system call).

Syntax: exit(0)

Ex.no: 2a(i) **Simulation of ls command**

Date:

Aim:

To simulate ls command using UNIX system calls.

Algorithm:

```
 1.Store path of current working directory using getcwd system call.
 2. Scan directory of the stored path using scandir system call and sort the resultant
array of structure.
 3. Display dname member for all entries if it is not a hidden file.
4. Stop.
```

Program:-

```
#include <stdio.h>
#include <dirent.h>
main()
{
        struct dirent **namelist;
        int n,i;
        char pathname[100];
        getcwd(pathname);
        n = scandir(pathname, &namelist, 0, alphasort);
        if(n < 0)
                printf("Error\n");
        else
                for(i=0; i<n; i++)
        if(namelist[i]->d_name[0] != '.')
                printf("%-20s", namelist[i]->d_name);
}
```

Result

Thus the filenames/subdirectories are listed, similar to ls command

Ex.no: 2a(ii) **Simulation of grep command**

Date:

Aim:

To simulate grep command using UNIX system call.

Algorithm:

```
1. Get filename and search string as command-line argument.
2. Open the file in read-only mode using open system call.
3. If file does not exist, then stop.
4. Let length of the search string be n.
5. Read line-by-line until end-of-file
    a. Check to find out the occurrence of the search string in a line by examining
characters in the range 1–n, 2–n+1, etc.
    b. If search string exists, then print the line.
6. Close the file using close system call.
7. Stop.
```

Program:-

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
main(int argc,char *argv[])
{
	FILE *fd;
	char str[100];
	char c;
	int i, flag, j, m, k;
	char temp[30];
	if(argc != 3)
	{
		printf("Usage: gcc mygrep.c –o mygrep\n");
```

```
        printf("Usage: ./mygrep <search_text> <filename>\n");
        exit(-1);
}
fd = fopen(argv[2],"r");
if(fd == NULL)
{
        printf("%s is not exist\n",argv[2]);
        exit(-1);
}
while(!feof(fd))
{
        i = 0;
        while(1)
        {
                c = fgetc(fd);
                if(feof(fd))
                {
                        str[i++] = '\0';
                        break;
                }
                if(c == '\n')
                {
                        str[i++] = '\0';
                        break;
                }
                str[i++] = c;
        }
        if(strlen(str) >= strlen(argv[1]))
        for(k=0; k<=strlen(str)-strlen(argv[1]); k++)
        {
                for(m=0; m<strlen(argv[1]); m++)
```

```
                        temp[m] = str[k+m];
                        temp[m] = '\0';
                        if(strcmp(temp,argv[1]) == 0)
                        {
                                printf("%s\n",str);
                                break;
                        }
                }
        }
}
```

Result

Thus the program simulates grep command by listing lines containing the search text.

Ex.no:2a(iii) **Simulation of cp command**

Date:

Aim

To simulate cp command using UNIX system call.

Algorithm

```
1. Get source and destination filename as command-line argument.
2. Declare a buffer of size 1KB
3. Open the source file in readonly mode using open system call.
4. If file does not exist, then stop.
5. Create the destination file using creat system call.
6. If file cannot be created, then stop.
7. File copy is achieved as follows:
    a. Read 1KB data from source file and store onto buffer using read system call.
    b. Write the buffer contents onto destination file using write system call.
    c. If end-of-file then step 8 else step 7a.
8. Close source and destination file using close system call.
9. Stop.
```

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
#define SIZE 1024
main(int argc, char *argv[])
{
        int src, dst, nread;
        char buf[SIZE];
        if (argc != 3)
        {
                printf("Usage: gcc copy.c -o copy\n");
```

```
		printf("Usage: ./copy <filename> <newfile> \n");
		exit(-1);
	}
	if ((src = open(argv[1], O_RDONLY)) == -1)
	{
		perror(argv[1]);
		exit(-1);
	}
	if ((dst = creat(argv[2], 0644)) == -1)
	{
		perror(argv[1]);
		exit(-1);
	}
	while ((nread = read(src, buf, SIZE)) > 0)
	{
		if (write(dst, buf, nread) == -1)
		{
			printf("can't write\n");
			exit(-1);
		}
	}
	close(src);
	close(dst);
}
```

Result:

Thus a file is copied using file I/O. The cmp command can be used to verify that contents of both file are same

Ex.no:2B **STUDY OF SHELL PROGRAMMING**

Date:

Aim:

To study about the Unix Shell Programming Commands and its usages.

INTRODUCTION:

Shell programming is a group of commands grouped together under single filename. After logging onto the system a prompt for input appears which is generated by a Command String Interpreter program called the shell. The shell interprets the input, takes appropriate action, and finally prompts for more input. Shell scripts are dynamically interpreted, NOT compiled.

SHELL KEYWORDS:

echo, read, if fi, else, case, esac, for , while , do , done, until , set, unset, readonly, shift, export, break, continue, exit, return, trap , wait, eval ,exec, ulimit , umask.

General things about SHELL:

The shbang line: The "shbang" line is the very first line of the script and lets the kernel know what shell will be interpreting the lines in the script. The shbang line consists of a #! followed by the full pathname to the shell, and can be followed by options to control the behavior of the shell.

EXAMPLE

```
#!/bin/sh
```

Comments: Comments are descriptive material preceded by a # sign. They are in effect until the end of a line and can be started anywhere on the line.

EXAMPLE

```
# this text is not
# interpreted by the shell
```

Wildcards: There are some characters that are evaluated by the shell in a special way. They are called shell metacharacters or "wildcards." These characters are neither numbers nor letters. For example, the *, ?, and [ ] are used for filename expansion. The <, >, 2>, >>, and | symbols are used for standard I/O redirection and pipes. To prevent these characters from being interpreted by the shell they must be quoted.

SHELL VARIABLES: Shell variables change during the execution of the program. The C Shell offers a command **"Set"** to assign a value to a variable.

For example:

```
set myname= Fred
set myname = "Fred Bloggs"
set age=20
```

A **$** sign operator is used to recall the variable values.

For example:

```
echo $myname will display Fred Bloggs on the screen
```

A **@** sign can be used to assign the integer constant values.

For example:

```
@myage=20
@age1=10
@age2=20
@age=$age1+$age2
echo $age
```

Local variables: Local variables are in scope for the current shell. When a script ends, they are no longer available; i.e., they go out of scope. Local variables are set and assigned values.

For example:

```
variable_name=value
name="John Doe"
x=5
```

Global variables: Global variables are called environment variables. They are set for the currently running shell and any process spawned from that shell. They go out of scope when the script ends.

For example:

```
VARIABLE_NAME=value
export VARIABLE_NAME
PATH=/bin:/usr/bin:.
export PATH
```

Extracting values from variables: To extract the value from variables, a dollar sign is used.

For example:

```
echo $variable_name
echo $name
echo $PATH
```

Rules: -

1. A variable name is any combination of alphabets, digits and an underscore („-„);
2. No commas or blanks are allowed within a variable name.
3. The first character of a variable name must either be an alphabet or an underscore.
4. Variables names should be of any reasonable length.
5. Variables name are case sensitive. That is Name, NAME, name, NAme, are all different variables.

EXPRESSION Command: To perform all arithmetic operations.

Syntax:

```
var = ‘expr $value1 + $ value2’
```

Arithmetic: The Bourne shell does not support arithmetic. UNIX/Linux commands must be used to perform calculations.

EXAMPLE

```
n=`expr 5 + 5`
echo $n
```

Operators: The Bourne shell uses the built-in test command operators to test numbers and strings.

EXAMPLE

Equality:

```
= string
!= string
-eq number
-ne number
```

Logical:

```
-a and
-o or
! not
```

Logical:

AND &&

OR ||

Relational:

-gt greater than

-ge greater than, equal to

-lt less than

-le less than, equal to

Arithmetic:

+, -, *, /, %

Arguments (positional parameters): Arguments can be passed to a script from the command line. Positional parameters are used to receive their values from within the script.

For example:

At the command line:

$ scriptname arg1 arg2 arg3 ...

In a script:

```
echo $1 $2 $3  Positional parameters
echo $*  All the positional parameters
echo $#  The number of positional parameters
```

READ Statement:

To get the input from the user.

Syntax :

```
read x y
```

(no need of commas between variables)

ECHO Statement:

Similar to the output statement. To print output to the screen, the echo command is used. Wildcards must be escaped with either a backslash or matching quotes.

Syntax :

Echo “String” (or) echo $ b(for variable).

For example:

```
echo "What is your name?"
```

Reading user input: The read command takes a line of input from the user and assigns it to a variable(s) on the right-hand side. The read command can accept muliple variable names. Each variable will be assigned a word.

For example:

```
echo "What is your name?"
read name
read name1 name2 ...
```

CONDITIONAL STATEMENTS:

The if construct is followed by a command. If an expression is to be tested, it is enclosed in square brackets. The then keyword is placed after the closing parenthesis. An if must end with a fi.

1. if This is used to check a condition and if it satisfies the condition if then does the next action , if not it goes to the else part.

2. if else

```
If cp $ source $ target
Then
Echo File copied successfully
Else
Echo Failed to copy the file.
```

3. nested if

Here sequence of condition is checked and the corresponding operation performed accordingly.

Syntax :

```
if test condition
then
command
if test condition
then
command
else
command
fi
```

```
fi
```

4. case .. esac

This construct helps in execution of the shell script based on Choice. The case command construct is:

```
case variable_name in
pattern1)
statements
;;
pattern2)
statements
;;
pattern3)
;;
*) default value
;;
esac
Example:
case "$color" in
blue)
echo $color is blue
;;
green)
echo $color is green
;;
red|orange)
echo $color is red or orange
;;
*) echo "Not a color" # default
```

LOOPS

There are three types of loops: while, until and for.

The **while loop** is followed by a command or an expression enclosed in square brackets, a do keyword, a block of statements, and terminated with the done keyword. As long as the expression is true, the body of statements between do and done will be executed.

The **until loop** is just like the while loop, except the body of the loop will be executed as long as the expression is false.

The **for loop** used to iterate through a list of words, processing a word and then shifting it off, to process the next word. When all words have been shifted from the list, it ends. The for loop is followed by a variable name, the in keyword, and a list of words then a block of statements, and terminates with the done keyword.

The loop control commands are **break and continue.**

Syntax:

```
 (1)while test command
    do
    block of statements
    done
    ------------
(2)while [ expression ]
    do
    block of statements
    done
    until command
    do
    block of statements
    done
    ------------
(3)for variable in word1 word2   ...
    do block of statements
    done
```

```
for((i=0;i<40;i++))
sh fgh.sh
```

bash fgh.sh

Break Statement:

This command is used to jump out of the loop instantly, without waiting to get the control command.

EXECUTION OF SHELL SCRIPT:

1.By using change mode command

2.$chmod u + x sum.sh

3.$ sum.sh

or

$ sh sum.sh

Result:

Thus the Shell commands and their usages.

Ex. no: 2b(i) **Program to display student grades**

Date:

Aim:

To write a Shell program to display student grades.

Algorithm:

1. Get the name, register number and three subject marks.
2. Find the total.
3. If any one of the mark is less than 50 then print fail. Otherwise print false.
4. Find the total and average.
5. If the average is greater than 80 then print Distinction.
6. If the average is greater than 70 then print First Class.
7. If the average is greater than 60 then print Second Class.
8. If the average is greater than 50 then print Third Class.

Program:

```
echo "Enter the student name"
read name
echo "Enter the Reg.No"
read no
echo "enter the 3 subject mark"
read m1 m2 m3
total=`expr $m1 + $m2 + $m3`
echo "         Mark List"
echo " Student Name : $name"
echo " Reg No : $no"
echo " Subject marks : $m1 $m2 $m3"
echo "Total : $total"
if test $m1 -lt 50 -o $m2 -lt 50  -o  $m3 -lt 50
then
echo "Result: Fail"
else
echo "Result: Pass"
avg=`expr $total / 3`
echo "Average:$avg"
```

```
if test $avg -gt 80
then
echo "Grade:Distinction"
elif test $avg -gt 70
then
echo "Grade: FIrst Class"
elif test $avg -gt 60
then
echo "Grade: Second Class"
elif test $avg -gt 50
then
echo "Grade: Third Class"
fi
fi
```

Output:

```
student@acew:~$ sh pgm1.sh
Enter the student name
jano
Enter the Reg.No
97113104030
enter the 3 subject mark
90 95 92
        Mark List
 Student Name : jano
 Reg No : 97113104030
 Subject marks : 90 95 92
Total : 277
Result: Pass
Average:92
Grade:Distinction
student@acew:~$
```

Result:

Thus the shell program to display student grades has been implemented and verified.

Ex.no :2b(ii) Program to check whether the given year is leap year or not

Date:

Aim:

To write a Shell program to check whether the given year is leap year or not.

Algorithm:

1. Get the year.
2. Check whether it is divisible by 100, if it is true then check if it is divisible by 400. Then print the year is leap year.
3. Otherwise check whether it is divisible by 4 and if it is true then print the year is leap year.

Program:

```
echo "Enter the year"
read year
if test `expr $year % 100` -eq 0
then
if test `expr $year % 400` -eq 0
then
echo "$year is a leap year"
else
echo "$year is not a leap year"
fi
elif test `expr $year % 4` -eq 0
then
echo "$year is a leap year"
else
echo "$year is not a leap year"
fi
```

Output:

```
student@acew:~$ sh pgm2.sh
Enter the year
2004
2004 is a leap year
```

Result:

Thus the shell program to check whether the given year is leap year or not has been implemented and verified.

Ex.no: 2b(iii) **Program to print the first ‘N’ prime numbers**

Date:

Aim:

To write a Shell program to print the first ‘N’ prime numbers

Algorithm:

1. Get the range.
2. Use the while loop and set the loop counter to 2.
3. Repeat the loop until the loop counter reaches the range.
4. Check whether the number is divisible by the same number, if it is true then print the number.

Program:

```
echo "Enter range"
read n
j=3
echo "Prime numbers are"
echo 2
while test $j -le $n
do
  x=0
 i=2
while test $i -lt $j
 do
             if test `expr $j % $i` -eq 0
            then
                        x=`expr $x + 1`
             fi
             i=`expr $i + 1`
done
if test $x -eq 0
 then
            echo "$j    "
 fi
```

```
     j=`expr $j + 1`
    done
```

Output:

```
student@acew:~$ sh pgm3.sh
enter a range
15
the prime nos are
2
3
5
7
11
13
```

Result:

Thus the shell program to print the first ‘N’ prime numbers has been implemented and verified.

Ex.no:2b(iv)   Program to print the factorial of first ‘N’ numbers

Date:

Aim:

To write a Shell program to print the factorial of first ‘N’ numbers

Algorithm:

1. Get the number
2. Initialize a variable fact to 1.
3. Set a while loop ranges from 1 to the number.
4. Multiply the loop counter with fact and store it again in fact.
5. Print the result.

Program:

```
fact=1
echo -n "Enter number to find factorial : "
read n
i=1
while test $i -le  $n
do
fact=`expr $fact \* $i`
echo "factorial of $i is $fact"
 i=`expr $i + 1`
done
```

Output:

```
student@acew:~$ sh pgm4.sh
Enter number to find factorial : 5
factorial of 1 is 1
factorial of 2 is 2
factorial of 3 is 6
factorial of 4 is 24
factorial of 5 is 120
student@sxcce:~$
```

Result:

Thus the shell program to print the factorial of first ‘N’ numbers has been implemented and verified.

Ex.no: 2b(v) **Program to find the roots of a quadratic equation**

Date:

Aim:

To write a Shell program to find the roots of a quadratic equation

Algorithm:

1. Get three numbers.
2. Find the roots of quadratic equation using formulas.
3. Print the values of the roots.

Program:

```
echo "Enter a value: "
read a
echo "Enter a value: "
read b
echo "Enter a value: "
read c
d=`expr $b \* $b - 4 \* $a \* $c`
x1=`echo "scale=3; (-$b + sqrt($d)) / (2 * $a)" | bc`
x2=`echo "scale=3; (-$b - sqrt($d)) / (2 * $a)" | bc`
echo "Root 1 : $x1"
echo "Root 2 : $x2"
```

Output:

```
student@acew:~$ sh pgm5.sh
Enter a value:
1
Enter a value:
3
Enter a value:
1
Root 1 : -.382
Root 2 : -2.618
student@acew:~$
```

Result:

Thus the shell program to find the roots of a quadratic equation has been implemented and verified.

Ex.no: 2b(vi) **Program to find the smallest number from a set of numbers**

Date:

Aim:

To write a Shell program to find the smallest number from a set of numbers.

Algorithm:

1. Read the elements in to an array.
2. Compare the adjacent elements. If it is greater than next one, then interchange it.
3. Print the first element of the array

Program:

```
echo "enter a number"
read n
echo "enter nos one by one"
for((i=1;i<=n;i++))
do
read a[$i]
done
for((i=1;i<=n;i++))
do
for((j=`expr $i + 1`;j<=n;j++))
do
if [ ${a[$i]} -gt ${a[$j]} ]
then
  t=${a[$i]}
a[$i]=${a[$j]}
a[$j]=$t
fi
done
done
echo "The smallest number is"
echo "${a[1]}"
```

Output:

```
student@acew:~$ chmod +x sor.sh
```

```
student@acew:~$ ./sor.sh
enter a number
3
enter nos one by one
5
2
7
The smallest number is
2
student@sxcce:~$
```

Result:

Thus the shell program to find the smallest number from a set of numbers has been implemented and verified.

Ex.no: 2b(vii) Program to sort numbers using arrays

Date:

Aim:

To write a Shell program to sort numbers using arrays

Algorithm:

1. Read the elements in to an array.
2. Compare the adjacent elements. If it is greater than next one, then interchange it.
3. Print all the element of the array

Program:

```
echo "enter a number"
read n
echo "enter nos one by one"
for((i=1;i<=n;i++))
do
read a[$i]
done
for((i=1;i<=n;i++))
do
for((j=`expr $i + 1`;j<=n;j++))
do
if test ${a[$i]} -gt ${a[$j]}
then
  t=${a[$i]}
a[$i]=${a[$j]}
a[$j]=$t
fi
done
done
echo "The sorted numbers are"
for((i=1;i<=n;i++))
do
echo "${a[$i]}"
done
```

Output:

```
student@sxcce:~$ chmod +x sorting.sh
student@sxcce:~$ ./sorting.sh
enter a number
5
enter nos one by one
45
67
23
12
89
The sorted numbers are
12
23
45
67
89
```

Result:

Thus the shell program to sort numbers using arrays has been implemented and verified

.

Ex.no: 2b(viii) **Program using strings**

Date:

Aim:

To write a Shell program using strings.

Algorithm:

1. Get two strings.
2. Concatenate the strings and print it.
3. Change the case and print it.
4. Find the length of the string and print it.

Program:

```
echo enter first string
read st1
echo enter second string
read st2
st=$st1$st2
echo concatenated string is $st
t=`echo $st|tr [a-z] [A-Z]`
echo The changed case string is $t
len=`expr $st|wc -c`
len1=`expr $len - 1`
echo "length of the string is $len1"
```

Output:

```
studentacew:~$ sh pgm8.sh
enter first string
operating
enter second string
system
concatenated string is operatingsystem
The changed case string is OPERATINGSYSTEM
length of the string is 15
student@acew:~$
```

Result:

Thus the shell program using strings has been implemented and verified.

Ex.No 3 PROCESS MANAGEMENT USING SYSTEM CALLS

Ex.no: 3a Program using system call fork()

Date:

Aim : To write the program to create a Child Process using system call fork().

Algorithm :

1. Declare the variable pid.
2. Get the pid value using system call fork().
3. If pid value is less than zero then print as “Fork failed”.
4. Else if pid value is equal to zero include the new process in the system‟s
Fil. using execlp system call.
5.Else if pid is greater than zero then it is the parent
process and it waits till the child completes using the system call wait()
6. Then print “Child complete”.

Program :

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
void main(int argc,char *arg[])
{
int pid;
pid=fork();
if(pid<0)
{
        printf("fork failed");
        exit(1);
}
else if(pid==0)
{
        execlp("whoami","ls",NULL);
        exit(0);
}
else
{
```

```
                printf("\n Process id is -%d\n",getpid());
                wait(NULL);
                exit(0);
        }
        }
```

Output:

```
[cse6@localhost Pgm]$ cc prog4a.c

[cse6@localhost Pgm]$ ./a.out
```

Result:

Thus the program was executed and verified successfully

Ex.no : 3b   Program using system calls   getpid() & getppid()

Date:

Aim :

To write the program to implement the system calls getpid() and getppid().

Algorithm :

1. Declare the variables pid , parent pid , child id and grand chil id.
2 .Get the child id value using system call fork().
3.If child id value is less than zero then print as “error at fork() child”.
4 .If child id !=0 then using getpid() system call get the process id.
5. Print “I am parent” and print the process id.
6 .Get the grand child id value using system call fork().
7 .If the grand child id value is less than zero then print as “error at fork() grand child”.
8. If the grand child id !=0 then using getpid system call get the process id.
9 .Assign the value of pid to my pid.
10 .Print “I am child” and print the value of my pid.
11 .Get my parent pid value using system call getppid().
12. Print “My parent’s process id” and its value.
13. Else print “I am the grand child”.
14 .Get the grand child’s process id using getpid() and print it as “my process id”.
15. Get the grand child‟s parent process id using getppid() and print it as “my parent’s process id

System calls used :

1.getpid( )

Each process is identified by its id value**.** This function is used to get the id value of a particular process.

2.getppid( )

Used to get particular process parent‟s id value.

3.perror( )

Indicate the process error.

Program :

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

```
#include<unistd.h>
#include<stdlib.h>
int main( )
{
int pid;
pid=fork( ); if(pid==-1)
{
        perror("fork failed"); exit(0);
 }
if(pid==0)
{
        printf("\n Child process is under execution");
        printf("\n Process id of the child process is %d", getpid()); printf("\n
        Process id of the parent process is %d", getppid());
}
else
{
        printf("\n Parent process is under execution");
        printf("\n Process id of the parent process is %d", getpid());
        printf("\n Process id of the child process in parent is %d", pid());
        printf("\n Process id of the parent of parent is %d", getppid());
}
return(0);
}
```

Output:

```
Child process is under execution
Process id of the child process is 9314
Process id of the parent process is 9313 Parent
process is under execution
```

```
Process id of the parent process is 9313
Process id of the child process in parent is 9314 Process
id of the parent of parent is 2825
```

Result:

Thus the program was executed and verified successfully

Ex.no: 3c. Program using system calls opendir( ) readdir( ) closedir()

Date:

Aim :

To write the program to implement the system calls opendir( ), readdir( ).

Algorithm :

1 : Start the program.

2 : In the main function pass the arguments.

3 : Create structure as stat buff and the variables as integer.

4 : Using the for loop, initialization

<u>System calls used</u> :

1.opendir( )

Open a directory.

2.readdir( )

Read a directory.

3.closedir()

Close a directory.

Program :

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/dir.h>
void main(int age,char *argv[])
{
        DIR *dir;
        struct dirent *rddir;
        printf("\n Listing the directory content\n");
        dir=opendir(argv[1]);
        while((rddir=readdir(dir))!=NULL)
        {
                printf("%s\t\n",rddir->d_name);
        }
        closedir(dir);
}
```

Output:

```
RP
roshi.c
first.c
pk6.c f2
abc FILE1
```

Result:

Thus the program was executed and verified successfully

Ex.no:3d **Program using system call exec( )**

Date:

Aim :

To write the program to implement the system call exec( ).

Algorithm :

1 : Include the necessary header files.

2 : Print execution of exec system call for the date Unix command.

3 : Execute the execlp function using the appropriate syntax for the Unix command date.

4 : The system date is displayed.

System call used :

1.execlp( )

Used after the fork() system call by one of the two processes to replace the process memory space with a new program. It loads a binary file into memory destroying the memory image of the program containing the execlp system call and starts its execution. The child process overlays its address space with the UNIX command /bin/ls using the execlp system call.

Syntax : execlp( )

Program :

```
 #include<stdio.h>
#include<unistd.h> main( )
{
            printf("\n exec system call");
             printf("displaying the date");
            execlp( "/bin/date", "date", 0);
}
```

Output:

```
Sat Dec 14 02 : 57 : 38 IST 2010
```

Result:

Thus the program was executed and verified successfully

Ex.no:3e Program using system calls wait( ) & exit( )

Date:

Aim :

To write the program to implement the system calls wait( ) and exit( ).

Algorithm :

1 : Declare the variables pid and i as integers.

2 : Get the child id value using the system call fork().

3 : If child id value is less than zero then print "fork failed".

4 : Else if child id value is equal to zero , it is the id value of the child and then start the child process to execute and perform Steps 6 & 7.

5 : Else perform Step 8.

6 : Use a for loop for almost five child processes to be called. Step

7 : After execution of the for loop then print "child process ends".

8 : Execute the system call wait( ) to make the parent to wait for the child process to get over.

9 : Once the child processes are terminated , the parent terminates and hence print "Parent process ends".

10 : After both the parent and the chid processes get terminated it execute the wait( ) system call to permanently get deleted from the OS.

System call used :

1. fork ( )

Used to create new process. The new process consists of a copy of the address space of the original process. The value of process id for the child process is zero, whereas the value of process id for the parent is an integer value greater than zero.

Syntax: fork ( )

2. wait ( )

The parent waits for the child process to complete using the wait system call. The wait system call returns the process identifier of a terminated child, so that the parent can tell which of its possibly many children has terminated.

Syntax: wait (NULL)

3. exit ( )

A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit system call. At that point, the process may return data (output) to its parent process (via the wait system call).

Syntax: exit(0)

Program :

```
#include<stdio.h>
#include<unistd.h>
main( )
{
        int i, pid;
        pid=fork( );
        if(pid== -1)
        {
                perror(“fork failed”);
                exit(0);
        }
        else if(pid==0)
        {
                printf(“\n Child process starts‟);
                for(i=0; i<5; i++)
                {
                        printf(“\n Child process %d is called”, i);
                }
                printf(“\n Child process ends”);
        }
        else
        {
                wait(0);
                printf(“\n Parent process ends”);
        }
        exit(0);
}
```

Output:

```
Child process starts Child
process 0 is called Child
process 1 is called Child
process 2 is called Child
process 3 is called Child
process 4 is called Child
process ends Parent process
ends
```

Result:

Thus the program was executed and verified successfully

Ex.no: 3f Program using system call stat( ) and close()

Date:

Aim:

To write the program to implement the system call stat( ) and close().

Algorithm :

1 : Include the necessary header files to execute the system call stat( ).

2 : Create a stat structure thebuf. Similarly get the pointers for the two structures passwd and group.

3 : Declare an array named path[20] of character type to get the input file along with its extension and an integer i.

4 : Get the input file along with its extension.

5 : If not of stat of a particular file then do the Steps 6 to 18.

6 : Print the file‟s pathname as file‟s name along with its extension.

7 : To check the type of the file whether it is a regular file or a directory use S_ISREG( ).

8 : If the above function returns true value the file is an regular file else it is directory.

9 : Display the file mode in octal representation using thebuf.st_mode.

10 : Display the device id in integer representation using thebuf.st_dev.

11 : Display the user id in integer representation using thebuf.st_uid.

12 : Display the user‟s pathname.

13 : Display the group id in integer representation using thebuf.st_gid.

14 : Display the group‟s pathname.

15 : Display the size of the file in integer representation using thebuf.st_size.

16 : Display the last access in time and date format using ctime(&thebuf.st_atime).

17 : Display the last modification in time and date format using ctime(&thebuf.st_atime).

18 : Display the last status in time and date format using ctime(&thebuf.st_atime).

19 : If Step 5 fails then print "Cannot get the status details for the given file".

Program:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
```

```
#include<unistd.h>
#include<fcntl.h>
main()
{
        int fd1,fd2,n;
        char source[30],ch[5];
        struct stat s,t,w;
        fd1=creat("text.txt",0644);
        printf("Enter the file to be copied\n");
        scanf("%s",source);
        fd2=open(source,0);
        if(fd2==-1)
        {
                perror("file doesnot exist");
                exit(0);
        }
        while((n=read(fd2,ch,1))>0)
                write(fd1,ch,n);
                close(fd2);
                stat(source,&s);
                printf("Source file size=%d\n",s.st_size);
                fstat(fd1,&t);
                printf("Destination file size =%d\n",t.st_size);
        close(fd1);
}
```

Result:

Thus the program was executed successfully.

Ex.no: :3g   Program using system calls open( ), read() & write( )

Date:

Aim :

Bo write the program to implement the system calls open( ),read( ) and write( ).

Algorithm :

```
1 : Declare the structure elements.
2 : Create a temporary file named temp1.
3 : Open the file named “test” in a write mode.
4 : Enter the strings for the file.
5 : Write those strings in the file named “test”.
6 : Create a temporary file named temp2.
7 : Open the file named “test” in a read mode.
8 : Read those strings present in the file “test” and save it in temp2.
9 : Print the strings which are read.
```

Program:

```
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<fcntl.h>
main( )
{
        int fd[2];
        char buf1[25]= ”just a test\n”; char
        buf2[50];
        fd[0]=open(“file1”, O_RDWR);
        fd[1]=open(“file2”, O_RDWR);
        write(fd[0], buf1, strlen(buf1));
        printf(“\n Enter the text now….”);
        gets(buf1);
        write(fd[0], buf1, strlen(buf1));
        lseek(fd[0], SEEK_SET, 0);
        read(fd[0], buf2, sizeof(buf1));
        write(fd[1], buf2, sizeof(buf2));
        close(fd[0]);
        close(fd[1]);
        printf("\n");
        return0;
    }
```

Output:

```
Enter the text now….progress
Cat file1 Just a
test progress
Cat file2 Just a test progress
```

Result:

Thus the program was executed successfully

Ex.No: 4 **CPU SCHEDULING ALGORITHMS**

Ex.no: 4a **Simulation of First Come First Serve Scheduling Algorithm(FCFS)**

Date:

Aim:

To write a c program to simulate the First Come First Serve Scheduling Algorithm.

Algorithm:

1. Get the number of processes.
2. Get the process name, CPU burst time (execution time) and its corresponding arrival time.
3. Print the Gantt chart with respect to its arrival time.
4. Calculate the waiting time and turn-around time for each of the process.
5. Print the average waiting time and average turn-around time.

Program:

```
#include<stdio.h>
#include<string.h>
main()
{
    int n,Bu[20],Twt=0,Ttt=0,A[10],Wt[10],w,ch,i,j,temp,temp1;
    float Awt,Att;
    char pname[20][20],c[20][20];
    printf("\n Enter the number of processes: ");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("\n\n Enter the process name: ");
        scanf("%s",pname[i]);
        printf("\n BurstTime for %s",pname[i]);
        scanf("%d",&Bu[i]);
        printf("\n Arrival Time for %s = ",pname[i]);
        scanf("%d",&A[i]);
    }
    printf("\n\n FIRST COME FIRST SERVED ALGORITHM\n\n");
    for(i=1;i<=n;i++)
    {
        for(j=i+1;j<=n;j++)
        {
            if(A[i]>A[j])
            {
                temp=Bu[i];
                temp1=A[i];
                Bu[i]=Bu[j];
                A[i]=A[j];
                Bu[j]=temp;
                A[j]=temp1;
                strcpy(c[i],pname[i]);
                strcpy(pname[i],pname[j]);
                strcpy(pname[j],c[i]);
            }
        }
    }
    Wt[1]=0;
    for(i=2;i<=n;i++)

    {
        Wt[i]=Bu[i-1]+Wt[i-1];
    }
    for(i=1;i<=n;i++)
    {
        Twt=Twt+(Wt[i]-A[i]);
        Ttt=Ttt+((Wt[i]+Bu[i])-A[i]);
    }
    Att=(float)Ttt/n;
    Awt=(float)Twt/n;
    printf("\n\n Average Turn around time=%3.2f ms ",Att);
    printf("\n\n AverageWaiting Time=%3.2f ms",Awt);
    printf("\n\n\t\t\tGANTT CHART\n");
    for(i=1;i<=n;i++)
```

```
        printf("|\t%s\t",pname[i]);
    printf("|\t\n");
    printf("\n");
    for(i=1;i<=n;i++)
        printf("%d\t\t",Wt[i]);
    printf("%d",Wt[n]+Bu[n]);
    printf("\n");
}
```

Output:

```
Enter the number of processes: 3
 Enter the process name: a
 BurstTime for a24
Arrival Time for a = 0
```

```
 Enter the process name: b
 BurstTime for b3
Arrival Time for b = 0
 Enter the process name: c
 BurstTime for c3
 Arrival Time for c = 0

 FIRST COME FIRST SERVED ALGORITHM
 Average Turn around time=27 ms  AverageWaiting Time=17 ms

                    GANTT CHART
|  a        |        b        |        c        |


0           24                27                30

student@acew:~$
```

Result:

Thus the C program to simulate First Come First Serve Scheduling Algorithm has been implemented and verified.

Ex.no: 4b   Simulation of Shortest Job First Scheduling Algorithm(SJF)

Date:

Aim:

To write a c program to simulate the Shortest Job First Scheduling Algorithm.

Algorithm:

1. Get the number of processes.
2. Get the process name, CPU burst time (execution time) and its corresponding arrival time.
3. Print the Gantt chart with respect to its arrival time & shortest execution time also.
4. Calculate the waiting time and turn-around time for each of the process.
5. Print the average waiting time and average turn-around time.

Program:

```
#include<stdio.h>
#include<string.h>
main()
{
        int n,Bu[20],Twt=0,Ttt=0,A[10],Wt[10],w,ch,i,j,temp,temp1;
        float Awt,Att;
        char pname[20][20],c[20][20];
        printf("\n Enter the number of processes: ");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                printf("\n\n Enter the process name: ");
                scanf("%s",pname[i]);
                printf("\n BurstTime for %s",pname[i]);
                scanf("%d",&Bu[i]);
        }
        printf("\n\n SHORTEST JOB FIRST SCHEDULING  ALGORITHM\n\n");
        for(i=1;i<=n;i++)
        {
                for(j=i+1;j<=n;j++)
                {
                        if(Bu[i]>Bu[j])
                        {
                                temp=Bu[i];
                Bu[i]=Bu[j];
                Bu[j]=temp;
                strcpy(c[i],pname[i]);
                strcpy(pname[i],pname[j]);
                strcpy(pname[j],c[i]);
            }
        }
    }
    for(i=1;i<=n;i++)
    {
        A[i]=0;
    }

    Wt[1]=0;
    for(i=2;i<=n;i++)
    {
        Wt[i]=Bu[i-1]+Wt[i-1];
    }
    for(i=1;i<=n;i++)
    {

        Twt=Twt+(Wt[i]-A[i]);
        Ttt=Ttt+((Wt[i]+Bu[i])-A[i]);
    }
    Att=(float)Ttt/n;
    Awt=(float)Twt/n;
    printf("\n\n Average Turn around time=%3.2f ms ",Att);
    printf("\n\n AverageWaiting Time=%3.2f ms",Awt);
    printf("\n\n\t\t\tGANTT CHART\n");
    for(i=1;i<=n;i++)
        printf("|\t%s\t",pname[i]);
        printf("|\t\n");
        printf("\n");
        for(i=1;i<=n;i++)
                printf("%d\t\t",Wt[i]);
        printf("%d",Wt[n]+Bu[n]);
        printf("\n-------------------------------\n");
        printf("\n");
}
```

Output:

```
student@acew:~$ ./a.out
 Enter the number of processes: 3
 Enter the process name: a
 BurstTime for a 7
 Enter the process name: b
```

```
BurstTime for b 4
Enter the process name: c
BurstTime for c 10
SHORTEST JOB FIRST SCHEDULING  ALGORITHM
Average Turn around time=12.00 ms
AverageWaiting Time=5.00 ms
                    GANTT CHART
|  b        |          a        |          c        |


0           4                   11                  21
```

Result:

Thus the C program to simulate Shortest Job First Scheduling Algorithm has been implemented and verified.

Ex.no: 4c Simulation of Priority Scheduling Algorithm

Date:

Aim:

To write a c program to simulate the Priority Scheduling Algorithm.

Algorithm:

1. Get the number of processes.

2. Get the process name, CPU burst time (execution time) and its corresponding priority.
3. Print the Gantt chart with respect to its priority given by the user.
4. Calculate the waiting time and turn-around time for each of the process.
5. Print the average waiting time and average turn-around time.

Program:

```
#include<stdio.h>
#include<string.h>
void main()
{
        int n,Bu[20],Twt=0,Ttt=0,A[10],Wt[10],w,ch,i,j,temp,temp1,P[10];
        float Awt,Att;
        char pname[20][20],c[20][20];

        printf("\n Enter the number of processes: ");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                printf("\n\n Enter the process name: ");
                scanf("%s",pname[i]);
                printf("\n BurstTime for %s",pname[i]);
                scanf("%d",&Bu[i]);
            printf("\n\n Enter the priority ");
                 scanf("%d",&P[i]);
        }
        printf("\n\n PRIORITY SCHEDULING  ALGORITHM\n\n");
        for(i=1;i<=n;i++)
        {
                for(j=i+1;j<=n;j++)
                {
                        if(P[i]>P[j])
                        {
```

```
                        temp=Bu[i];
                       Bu[i]=Bu[j];
                        Bu[j]=temp;
                        strcpy(c[i],pname[i]);
                        strcpy(pname[i],pname[j]);
                        strcpy(pname[j],c[i]);
                }
        }
}
for(i=1;i<=n;i++)
{
        A[i]=0;
}
Wt[1]=0;
for(i=2;i<=n;i++)
{
        Wt[i]=Bu[i-1]+Wt[i-1];
}
for(i=1;i<=n;i++)
{


        Twt=Twt+(Wt[i]-A[i]);
        Ttt=Ttt+((Wt[i]+Bu[i])-A[i]);
}
Att=(float)Ttt/n;
Awt=(float)Twt/n;
printf("\n\n Average Turn around time=%3.2f ms ",Att);
printf("\n\n AverageWaiting Time=%3.2f ms",Awt);
printf("\n\n\t\t\tGANTT CHART\n");
for(i=1;i<=n;i++)
        printf("|\t%s\t",pname[i]);
        printf("|\t\n");
        printf("\n");
        for(i=1;i<=n;i++)
                printf("%d\t\t",Wt[i]);
        printf("%d",Wt[n]+Bu[n]);
            printf("\n");
}
```

Output:

```
          Enter the number of processes: 3
          Enter the process name: a
          BurstTime for a 6
          Enter the priority 2
          Enter the process name: b
```

```
BurstTime for b 7
Enter the priority 6
Enter the process name: c
BurstTime for c 3
Enter the priority 1
PRIORITY SCHEDULING  ALGORITHM
Average Turn around time=9.33 ms
AverageWaiting Time=4.00 ms

GANTT CHART
|  c        |         a        |         b        |


0          3                   9                  16
```

Result:

Thus the C program to simulate Priority Scheduling Algorithm has been implemented and verified.

Ex. no: 4d   Simulation of Round Robin Scheduling Algorithm

Date:

Aim:

To write a c program to simulate the Round Robin Scheduling Algorithm.

Procedure:

1. Get the number of processes.

2. Get the process name; CPU burst time (execution time), arrival time and time slice (quantum).
3. Print the Gantt chart as per the Round Robin Scheduling algorithm [i.e. processes are dispatched FIFO but are given a limited amount of CPU time called a time slice or a quantum. If a process does not complete before its CPU time expires, the CPU is preempted and given o the next waiting process. The preempted process is then placed at the back of the ready queue] i.e. with respect to its arrival time & time slice.
4. Calculate the waiting time and turn-around time for each of the process.
5. Print the average waiting time and average turn-around time.

Program:

```
#include<string.h>
#include<stdio.h>
void main()
{
        struct rrsa
        {
          char name[10];
          int st,pt;
        }r[15];
        int n,Bu[20],Twt=0,Ttt=0,A[10],Wt[20],rt[20],temp,temp1,size=0;
        int z,q,i,j,s,k=0,t,t1,tt[20];
        float Awt,Att;
        char pname[20][20],c[20][20];

        for(i=0;i<n;i++)
        {
        strcpy(r[i].name," ");
        r[i].st=0;
        r[i].pt=0;
        }
```

```
printf("\n Enter the number of processes: ");
scanf("%d",&n);
        for(i=0;i<n;i++)
{
        printf("\n\n Enter the process name: ");
        scanf("%s",pname[i]);
        printf("\n BurstTime for %s",pname[i]);
        scanf("%d",&Bu[i]);
        printf("Enter Arrival time ");
        scanf("%d",&A[i]);
}
printf("Enter the time slice ");
scanf("%d",&q);
printf("\n\n R R SCHEDULING ALGORITHM\n\n");
for(i=0;i<n;i++)
{
        for(j=0;j<n;j++)
        {
                if(A[i]<A[j])
                {
                        temp=Bu[i];
                        temp1=A[i];
                        Bu[i]=Bu[j];
                        A[i]=A[j];
                        Bu[j]=temp;
                        A[j]=temp1;
                        strcpy(c[i],pname[i]);
                        strcpy(pname[i],pname[j]);
                        strcpy(pname[j],c[i]);
                }
        }
```

```
}
k=0;
for(i=0;i<n;i++)
{
        rt[i]=Bu[i];
        Wt[i]=0;
}
r[0].st=A[0];
label1:
for(j=0;j<n;j++)
{
        if((rt[j]<=q)&&(rt[j]!=0))
        {
                if(r[k].st>=A[j])
                {
                        strcpy(r[k].name,pname[j]);
                        r[k].pt=r[k].st+rt[j];
                        r[k+1].st=r[k].pt;
                        rt[j]=0;
                        k++;
                        size++;
                }
        }
else if((rt[j]>q)&&(rt[j]!=0))
{
        if(r[k].st>=A[j])
        {
                strcpy(r[k].name,pname[j]);
                r[k].pt=r[k].st+q;
                r[k+1].st=r[k].pt;
                rt[j]=rt[j]-q;
                k++;
                size++;
        }
}
}
for(i=1;i<n;i++)
{
        if(rt[i-0]!=0)
        {
                goto label1;
        }
}
for(i=0;i<n;i++)
{
        s=i;
        label2:
        if(strcmp(pname[i],r[s].name)==0)
        {
                t=s;
                goto label3;
        }
        else
        {
                s++;
                t=s;
                goto label2;
        }
label3:
k=s+1;
for(;k<size;k++)
{
```

```
                if(strcmp(r[s].name,r[k].name)==0)
                {
                        t1=r[k].st-r[s].pt;
                        Wt[i]+=t1;
                        s=k;
                }
        }

        Wt[i]=Wt[i]+r[t].st-A[i];
        tt[i]=Bu[i]+Wt[i];
        Twt=Twt+Wt[i];
        Ttt=Ttt+tt[i];
        }
        Awt=(float)Twt/n;
        Att=(float)Ttt/n;
        printf("Gantt chart\n\n");
        for(i=0;i<size;i++)
        {
                printf("%d\t%s\t%d\n\n",r[i].st,r[i].name,r[i].pt);
        }
        printf("\n\nAvg. wt time :%f\n",Awt);
        printf("Avg. T Time:%f\n",Att);
}
```

Output

```
 Enter the number of processes: 3
 Enter the process name: a
 BurstTime for a 4
Enter Arrival time 2
 Enter the process name: b
 BurstTime for b 6
Enter Arrival time 0
 Enter the process name: c
 BurstTime for c 9
Enter Arrival time 3
Enter the time slice 5
 R R SCHEDULING ALGORITHM
Gantt chart


0          b        5


5          a        9


9          c        14


14         b        15


15         c        19
Avg. wt time: 6.333333
Avg. T Time: 12.666667
student@acew:~$
```

Result:

Thus the C program to simulate Round Robin Scheduling Algorithm has been implemented and verified.

Ex.no: 5 INTERPROCESS COMMUNICATION

Ex.no:5a Shared Memory

Aim:

To demonstrate communication between process using shared memory.

Algorithm:

Server

```
1. Initialize size of shared memory shmsize to 27.
2. Initialize key to 2013 (some random value).
3. Create a shared memory segment using shmget with key & IPC_CREAT as parameter.
    a. If shared memory identifier shmid is -1, then stop.
4. Display shmid.
5. Attach server process to the shared memory using shmmat with shmid as parameter.
    a. If pointer to the shared memory is not obtained, then stop.
6. Clear contents of the shared region using memset function.
7. Write a–z onto the shared memory.
8. Wait till client reads the shared memory contents
9. Detatch process from the shared memory using shmdt system call.
10. Remove shared memory from the system using shmctl with IPC_RMID argument
```

Client

```
1. Initialize size of shared memory shmsize to 27.
2. Initialize key to 2013 (same value as in server).
3. Obtain access to the same shared memory segment using same key.
    a. If obtained then display the shmid else print "Server not started"
4. Attach client process to the shared memory using shmmat with shmid as parameter.
    a. If pointer to the shared memory is not obtained, then stop.
5. Read contents of shared memory and print it.
6. After reading, modify the first character of shared memory to '*'
```

Program

Server: **/* Shared memory server - shms.c */**

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/un.h>
```

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define shmsize 27
main()
{
        char c;
        int shmid;
        key_t key = 2013;
        char *shm, *s;
        if ((shmid = shmget(key, shmsize, IPC_CREAT|0666)) < 0)
        {
                perror("shmget");
                exit(1);
        }
        printf("Shared memory id : %d\n", shmid);
        if ((shm = shmat(shmid, NULL, 0)) == (char *) -1)
        {
                perror("shmat");
                exit(1);
        }
        memset(shm, 0, shmsize);
        s = shm;
        printf("Writing (a-z) onto shared memory\n");
        for (c = 'a'; c <= 'z'; c++)
        *s++ = c;
        *s = '\0';
        while (*shm != '*');
        printf("Client finished reading\n");
        if(shmdt(shm) != 0)
        fprintf(stderr, "Could not close memory segment.\n");
```

```
	shmctl(shmid, IPC_RMID, 0);
}
```

Client:/* Shared memory client - shmc.c */

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define shmsize 27
main()
{
	int shmid;
	key_t key = 2013;
	char *shm, *s;
	if ((shmid = shmget(key, shmsize, 0666)) < 0)
	{
		printf("Server not started\n");
		exit(1);
	}
	else
	printf("Accessing shared memory id : %d\n",shmid);
	if ((shm = shmat(shmid, NULL, 0)) == (char *) -1)
	{
		perror("shmat");
		exit(1);
	}
	printf("Shared memory contents:\n");
	for (s = shm; *s != '\0'; s++)
	putchar(*s);
	putchar('\n');
	*shm = '*';
}
```

Output:-

Server

```
$ gcc shms.c -o shms
$ ./shms
Shared memory id : 196611
Writing (a-z) onto shared memory
Client finished reading
```

Client

```
$ gcc shmc.c -o shmc
$ ./shmc
Accessing shared memory id : 196611
Shared memory contents:
abcdefghijklmnopqrstuvwxyz
```

Result

Thus contents written onto shared memory by the server process is read by the client process.

Exp: 5b **Fibonacci & Prime Number**

Date:

Aim:

To write a C program to implement Inter Process Communication using pipes.

Algorithm:

1. Create a pipe using pipe()system call and check for error condition.

2. Create a child using fork system call and check for error condition.
3. If the return value is not zero, then display it as “Parent Process” and get the limit from the user. While getting the inputs, write each data into pipe1 by closing its read end and after completion, close the write end of pipe1.
4. If the return value is zero, read data from pipe0 by closing its write end and then display it as “Child Process” by printing the Fibonacci series and close the read end of pipe0.
5. Terminate the program.

Program:

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

Output:

```
student@acew:~$ ./a.out
Parent process
Fibonacci series
Enter the limit for the series:5
child process
```

```
Fibonacci Series is:
0
1
1
2
3
student@acew:~$
```

Result:

Thus the C program to implement Inter Process Communication using pipes has been implemented and verified.

Ex.no: 6 MUTUAL EXCLUSION BY SEMAPHORE

Aim:

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

Algorithm:

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

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 enter into the critical section.

iii) If the mutex value is 0, allow the producer to add value in the temporary variable to the buffer.

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 for mutex value, if the mutex==0, remove item from buffer

iii) Signal the mutex value and reduce the empty value by 1.

iv) Consume the item.

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:

```
student@acew:~$ ./a.out
1.Producer  2.Consumer
  Producer produces item1
 Consumer consumes  item1
Producer produces item2
 Consumer consumes  item2
Producer produces item3
 Consumer consumes  item3
```

Result:

Thus the program was executed successfully

Ex.no: 7 IMPLEMENTATION OF BANKERS ALGORITHM FOR DEAD LOCK AVOIDANCE

Date:

Aim:

To write a C program to implement Bankers algorithm for deadlock avoidance.

Algorithm:

1. Get total number of processes.
2. Get total number of resources of each resource type.
3. Get the maximum need of each process.
4. Enter allocation matrix.
5. Get the available resources of each resource type.
6. Find out the order in which we can complete the execution of processes with the available resources.
7. If all the processes can complete with the available resources then display the safe sequence.
8. Otherwise display deadlock occurs.

Program:

```
#include<stdio.h>
struct da
{
	int max[10],a1[10],need[10],before[10],after[10];
}p[10];
main()
{
	int i,j,k,l,r,n,tot[10],av[10],cn=0,cz=0,temp=0,c=0;
	printf("\nEnter the no.of processes ");
	scanf("%d",&n);
	printf("\nEnter 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("\nResource  Max  Allocated \t 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\nResource  Avail Before 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("\nP%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("\nDeadlock occured");
}
```

Output:

```
    student@acew:~$ ./a.out
    Enter the no.of processes 5
    Enter the no.of resources 3
    Process1
```

```
maximum value for resource 1:7
maximum value for resource 2:5
maximum value for resource 3:3
Allocated from resource 1:0
Allocated from resource 2:1
Allocated from resource 3:0
Process2
maximum value for resource 1:3
maximum value for resource 2:2
maximum value for resource 3:2
Allocated from resource 1:2
Allocated from resource 2:0
Allocated from resource 3:0
Process3
maximum value for resource 1:9
maximum value for resource 2:0
maximum value for resource 3:2
Allocated from resource 1:3
Allocated from resource 2:0
Allocated from resource 3:2
Process4
maximum value for resource 1:2
maximum value for resource 2:2
maximum value for resource 3:2
Allocated from resource 1:2
Allocated from resource 2:1
Allocated from resource 3:1
Process5
maximum value for resource 1:4
maximum value for resource 2:3
maximum value for resource 3:3
```

```
Allocated from resource 1:0
Allocated from resource 2:0
Allocated from resource 3:2
Enter total value of resource 1:10
Enter total value of resource 2:5
Enter total value of resource 3:7

Resource      Max    Allocated     Needed    Total      Avail
   P1       7 5 3    0  1  0              7 4 3    10 5 7    3 3 2
   P2       3 2 2    2  0  0              1 2 2
   P3       9 0 2    3  0  2        6 0 0
   P4       2 2 2    2  1  1             0 1 1
   P5       4 3 3    0  0  2       4 3 1

   Resource  Avail Before Avail After
   P2       2 1 0    5 3 2
   P4       5 2 1    7 4 3
   P1       0 0 0    7 5 3
   P3       1 5 3    10 5 5
   P5       6 2 4    10 5 7
   The above sequence is a safe sequence
   student@acew:~$
```

Result:

Thus the C program to implement Bankers algorithm for deadlock avoidance has been implemented and verified.

Ex.no: 8 IMPLEMENTATION OF DEAD LOCK DETECTION ALGORITHM

Date:

Aim:

To write a C program to implement deadlock detection algorithm.

Algorithm:

1. Get total number of processes.

2. Get request matrix.
3. Enter allocation matrix.
4. Get total number of resources of each resource type.
5. Get the available resources of each resource type.
6. Find out the processes that are not able to complete with the available resources.
7. Display the processes that are causing deadlock.

Program:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
        int found,flag,l,p[4][5],tp,c[4][5],i,j,k=1,m[5],r[5],a[5],temp[5],sum=0;
        printf("Enter total no of processes");
        scanf("%d",&tp);
        printf("Enter claim matrix");
        for(i=1;i<=4;i++)
        {
                for(j=1;j<=5;j++)
                {
                        scanf("%d",&c[i][j]);
                }
        }
        printf("Enter allocation matrix");
        for(i=1;i<=4;i++)
                for(j=1;j<=5;j++)
                {
                        scanf("%d",&p[i][j]);
                }
        printf("Enter resource vector:\n");
        for(i=1;i<=5;i++)
```

```
{
        scanf("%d",&r[i]);
}
printf("Enter availability vector:\n");
for(i=1;i<=5;i++)
{
        scanf("%d",&a[i]);
        temp[i]=a[i];
}
for(i=1;i<=4;i++)
{
        sum=0;
        for(j=1;j<=5;j++)
        {
                sum+=p[i][j];
        }
        if(sum==0)
        {
                m[k]=i;
                k++;
        }
}
for(i=1;i<=4;i++)
{
        for(l=1;l<k;l++)
        if(i!=m[l])
        {
                flag=1;
        for(j=1;j<=5;j++)
        if(c[i][j]>temp[j])
        {
                flag=0;
                break;
        }
}
if(flag==1)
{
        m[k]=i;
        k++;
        for(j=1;j<=5;j++)
                temp[j]+=p[i][j];
}
}
printf("deadlock causing processes are:");
for(j=1;j<=tp;j++)
{
        found=0;
        for(i=1;i<k;i++)
        {
                if(j==m[i])
                    found=1;
        }
if(found==0)
printf("%d\t",j);
}
```

Output:

```
Enter total no of processes4
Enter claim matrix
0 1 0 0 1
0 0 1 0 1
0 0 0 0 1
```

```
1 0 1 0 1
Enter allocation matrix1 0 1 1 0
1 1 0 0 0
0 0 0 1 0
0 0 0 0 0
Enter resource vector:
2 1 1 2 1
Enter availability vector:
0 0 0 0 1
deadlock causing processes are:1        2
```

Result:

Thus the C program to implement deadlock detection algorithm has been implemented and verified.

Ex.no:9 **MULTITHREADING**

Date:

Aim:

To write a C program to implement Multithreading.

Algorithm:

1. Declare two objects.

2. Create two threads from pthread library.
3. Call the functions simultaneously using thread.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
 void* fun()
{
   printf("\nThe thread 1 is running");
}
void* van()
{
   printf("\nthread 2 is running ");
}
int main()
{
   pthread_t t1,t2;
   pthread_create(&t1,NULL,fun,NULL);
   pthread_create(&t2,NULL,van,NULL);
   printf("\nI'm in main\n");
   pthread_join(t2,NULL);
}
```

Output:

```
student@acew:~$ cc mul.c -lpthread
student@acew:~$ ./a.out
I'm in main
thread 2 is running
The thread 1 is running
```

```
student@acew:~$
```

Result:

Thus the C program to implement Multithreading Technique has been implemented and verified.

Ex. no: 10 **PAGING**

Date:

Aim: To implement the Memory management policy- Paging.

Algorithm:

1. Read all the necessary input from the keyboard.

2. Pages - Logical memory is broken into fixed - sized blocks. 3: Frames – Physical memory is broken into fixed – sized blocks.

3.Calculate the physical address using the following

Physical address = (Frame number * Frame size) + offset

5.Display the physical address. 6: Stop the process.

Program:

```
#include <stdio.h>
#include <conio.h>
struct pstruct
{
	int fno;
	int pbit;
}ptable[10];
int pmsize,lmsize,psize,frame,page,ftable[20],frameno;
void info()
{
	printf("\n\nMEMORY MANAGEMENT USING PAGING\n\n");
	printf("\n\nEnter the Size of Physical memory: ");
	scanf("%d",&pmsize);
	printf("\n\nEnter the size of Logical memory: ");
	scanf("%d",&lmsize);
	printf("\n\nEnter the partition size: ");
	scanf("%d",&psize);
	frame = (int) pmsize/psize;
	page = (int) lmsize/psize;
	printf("\nThe physical memory is divided into %d no.of frames\n",frame);
	printf("\nThe Logical memory is divided into %d no.of pages",page);
}
void assign()
{
	int i;
	for (i=0;i<page;i++)
	{
	ptable[i].fno = -1;
	ptable[i].pbit= -1;
	}
	for(i=0; i<frame;i++)
		ftable[i] = 32555;
	for (i=0;i<page;i++)
	{
	printf("\n\nEnter the Frame number where page %d must be placed: ",i);
		scanf("%d",&frameno);
		ftable[frameno] = i;
		if(ptable[i].pbit == -1)
		{
			ptable[i].fno = frameno;
			ptable[i].pbit = 1;
		}
	}
	getch();
	printf("\n\nPAGE TABLE\n\n");
	printf("PageAddress  FrameNo. PresenceBit\n\n");
	for (i=0;i<page;i++)
		printf("%d\t\t%d\t\t%d\n",i,ptable[i].fno,ptable[i].pbit);
	printf("\n\n\n\tFRAME TABLE\n\n");
	printf("FrameAddress   PageNo\n\n");
	for(i=0;i<frame;i++)
		printf("%d\t\t%d\n",i,ftable[i]);
}
void cphyaddr()
{
	int laddr,paddr,disp,phyaddr,baddr;
```

```
        getch();
        printf("\n\n\n\tProcess to create the Physical Address\n\n");
        printf("\nEnter the Base Address: ");
        scanf("%d",&baddr);
        printf("\nEnter theLogical Address: ");
        scanf("%d",&laddr);

        paddr = laddr / psize;
        disp = laddr % psize;
         if(ptable[paddr].pbit == 1 )
                phyaddr = baddr + (ptable[paddr].fno*psize) + disp;
        printf("\nThe Physical Address where the instruction present: %d",phyaddr);
}
void main()
{
        clrscr();
        info();
        assign();
        cphyaddr();
        getch();
}
```

Output:

```
MEMORY MANAGEMENT USING PAGING
 Enter the Size of Physical memory: 16
Enter the size of Logical memory: 8
Enter the partition size: 2
The physical memory is divided into 8 no.of frames
The Logical memory is divided into 4 no.of pages
Enter the Frame number where page 0 must be placed: 5
Enter the Frame number where page 1 must be placed: 6
Enter the Frame number where page 2 must be placed: 7
Enter the Frame number where page 3 must be placed: 2
PAGE TABLE
PageAddress        FrameNo.           PresenceBit


     0                5                    1
     1                6                    1
     2                7                    1
     3                2                    1

FRAME TABLE
 FrameAddress          PageNo
      0               32555
      1               32555
      2                 3
      3               32555
      4               32555
      5                 0
      6                 1
      7                 2
Process to create the Physical Address
  Enter the Base Address: 1000
 Enter theLogical Address: 3
The Physical Address where the instruction present: 1013
```

Result:

Thus the C program for Paging technique is Implemented.

Ex.no:11 IMPLEMENTATION OF CONTIGUOUS MEMORY ALLOCATION TECHNIQUE

Date:

Aim:

To write a C program to implement Contiguous memory allocation Technique.

Algorithm:

1. Get the number of processes & its corresponding size.
2. Get the number of segment frames & its corresponding size.
3. Select the allocation strategy to place an incoming segment in primary storage, among the following
    a. Best Fit
    b. Worst Fit
    c. First Fit
4. First Fit strategy – An incoming process is placed in the main storage in the first available segment large enough to hold it i.e. scan the primary memory from the beginning and chooses the first available segment that is large enough.
5. Best Fit strategy – An incoming process is placed in the segment in main storage in which it fits most tightly and leaves the smallest amount of unused space i.e. this strategy chooses the segment that is closest in size to the request i.e. the unused space is minimum.
6. Worst Fit strategy – An incoming process is placed in the largest possible segment in the main storage i.e. the unused space is maximum.
7. Find the efficient placement strategy.

Program:

```
#include<stdio.h>
main()
{
        int p,s,i,n,j,unf=0,usedf=0,usedw=0,usedb=0,unb=0;
        int unw=0,total=0,t;
        struct segment
        {
                int size,pos,flag,flag2,flag3;
        }seg[20];
        struct process
        {
```

```
        int size,flag;
}proc[20];
printf("Enter the no. of process  : ");
scanf("%d",&p);
printf("\nEnter the process size  : ");
for(i=0;i<p;i++)
{
        scanf("%d",&proc[i].size);
        proc[i].flag=0;
}
printf("Enter the no.of partitions :");
scanf("%d",&s);
printf("Enter the partition size  :");
for(i=0;i<s;i++)
{
        scanf("%d",&seg[i].size);
        seg[i].pos=i;
        seg[i].flag=seg[i].flag2=seg[i].flag3=0;
        total=total+seg[i].size;
}
printf("\nFIRST FIT ALLOCATION STRATEGY\n");
printf("process-id\tprocess size\tpartition no.\tpartition size\n");
for(i=0;i<p;i++)
{
        proc[i].flag=0;
        for(j=0;j<s;j++)
        {
        if((proc[i].size<=seg[j].size)&&(seg[j].flag==0))
        {
                seg[j].flag=1;
                proc[i].flag=1;
```

```
                        usedf=usedf+proc[i].size;
                            printf("\n%d\t\t%d\t\t%d\t\t\t%d\t",i,proc[i].size,seg[j].pos,seg[j].size);
                break;
                }
        }
}
for(i=0;i<p;i++)
{
        if(proc[i].flag==0)
        {
                printf("\nprocess %d cannot be stored in any partitions.\n",i);
        }
 }
unf=total-usedf;
printf("\nUnused space after first fit allocation =%d",unf);
printf("\nBEST FIT ALLOCATION STRATEGY\n");
for(i=0;i<s;i++)
{
        for(j=i+1;j<s;j++)
        {
                if(seg[i].size>=seg[j].size)
                {
                        t=seg[i].size;
                        seg[i].size=seg[j].size;
                        seg[j].size=t;
                        t=seg[i].pos;
                        seg[i].pos=seg[j].pos;
                        seg[j].pos=t;
                }
        }
}
```

```
printf("process-id\tprocess size\tpartition number\tpartition size\n");
for(i=0;i<p;i++)
{
        proc[i].flag=0;
        for(j=0;j<s;j++)
         {
                 if((proc[i].size<=seg[j].size)&&(seg[j].flag2==0))
                  {
                     seg[j].flag2=1;
                        proc[i].flag=1;
                        usedb=usedb+proc[i].size;
                        printf("\n%d\t\t%d\t\t%d\t\t\t%d\t",i,proc[i].size,seg[j].pos,seg[j].size);
                        break;
                }
          }
  }
 for(i=0;i<p;i++)
  {
          if(proc[i].flag==0)
          {
               printf("\nprocess %d cannot be stored in any partitions.\n",i);
         }
 }
 unb=total-usedb;
 printf("\nUnused space after best fit allocation =%d",unb);
 printf("\nWORST FIT ALLOCATION STRATEGY\n");
 for(i=0;i<s;i++)
{
        for(j=i+1;j<s;j++)
        {
              if(seg[i].size<=seg[j].size)
```

```
            {
                t=seg[i].size;
                seg[i].size=seg[j].size;
                seg[j].size=t;
                t=seg[i].pos;
                seg[i].pos=seg[j].pos;
                seg[j].pos=t;
            }
        }
}
printf("process-id\tprocess size\tpartition number\tpartition size\n");
for(i=0;i<p;i++)
{
          proc[i].flag=0;
          for(j=0;j<s;j++)
          {
            if((proc[i].size<=seg[j].size)&&(seg[j].flag3==0))
                {
                      seg[j].flag3=1;
                      proc[i].flag=1;
                      usedw=usedw+proc[i].size;
                         printf("\n%d\t\t%d\t\t%d\t\t\t%d\t",i,proc[i].size,seg[j].pos,seg[j].size);
                         break;
                 }
          }

 }
 for(i=0;i<p;i++)
 {
        if(proc[i].flag==0)
```

```
            {
                printf("\nprocess %d cannot be stored in any partitions.\n",i);
            }

        }
        unw=total-usedw;
        printf("\nUnused space after worst fit allocation =%d",unw);
        printf("\nEfficient algorithm among the three strategies...\n");
          if((unf<=unb)&&(unb<=unw))
                printf("\nFirst fit algorithm is efficient\n");
           if((unb<=unw)&&(unb<=unf))
                printf("\nBest fit algorithm is efficient\n");
          if((unw<=unf)&&(unw<=unb))
               printf("\nworst fit algorithm is efficient\n");

}
```

Output:

```
student@acew:~$ ./a.out
Enter the no. of process  : 4
Enter the process size  : 212 417 112 426
Enter the no.of partitions :5
Enter the partition size  :100 500 200 300 600
FIRST FIT ALLOCATION STRATEGY
process-id        process size    partition no.    partition size


0        212            1                      500
1        417            4                      600
2        112            2                      200
process 3 cannot be stored in any partitions.
Unused space after first fit allocation =959
```

```
BEST FIT ALLOCATION STRATEGY
process-id      process size    partition number      partition size


0           212           3                         300
1           417           1                         500
2           112           2                         200
3           426           4                         600
Unused space after best fit allocation =533
WORST FIT ALLOCATION STRATEGY
process-id      process size    partition number      partition size


0           212           4                         600
1           417           1                         500
2           112           3                         300
process 3 cannot be stored in any partitions.
Unused space after worst fit allocation =959
Efficient algorithm among the three strategies...
Best fit algorithm is efficient
student@acew:~$
```

Result:

Thus the C program to implement Contiguous memory allocation Technique has been implemented and verified.

Ex.No :12 **PAGE REPLACEMENT**

Ex.no:12 a **FIFO Page Replacement**

Date:

Aim:

To implement demand paging for a reference string using FIFO method.

Algorithm:

```
1. Get length of the reference string, say l.
2. Get reference string and store it in an array, say rs.
3. Get number of frames, say nf.
4. Initalize frame array upto length nf to -1.
5. Initialize position of the oldest page, say j to 0.
6. Initialize no. of page faults, say count to 0.
7. For each page in reference string in the given order, examine:
    a. Check whether page exist in the frame array
    b. If it does not exist then
        i. Replace page in position j.
        ii. Compute page replacement position as (j+1) modulus nf.
        iii. Increment count by 1.
        iv. Display pages in frame array.
8. Print count.
```

Program:

```
#include <stdio.h>
main()
{
    int i,j,l,rs[50],frame[10],nf,k,avail,count=0;
    printf("Enter length of ref. string : ");
    scanf("%d", &l);
    printf("Enter reference string :\n");
    for(i=1; i<=l; i++)
    scanf("%d", &rs[i]);
    printf("Enter number of frames : ");
    scanf("%d", &nf);
    for(i=0; i<nf; i++)
    frame[i] = -1;
    j = 0;
```

```
        printf("\nRef. str Page frames");
        for(i=1; i<=l; i++)
        {
                printf("\n%4d\t", rs[i]);
                avail = 0;
                for(k=0; k<nf; k++)
                if(frame[k] == rs[i])
                avail = 1;
                if(avail == 0)
                {
                        frame[j] = rs[i];
                        j = (j+1) % nf;
                        count++;
                        for(k=0; k<nf; k++)
                        printf("%4d", frame[k]);
                }
        }
        printf("\n\nTotal no. of page faults : %d\n",count);
}
```

Output:

```
$ gcc fifopr.c
$ ./a.out
Enter length of ref. string : 20
Enter reference string :
1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6
Enter number of frames : 5
Ref. str Page frames
1       1       -1      -1      -1      -1
2       1       2       -1      -1      -1
3       1       2       3       -1      -1
4       1       2       3       4       -1
```

```
2
1
5	1	2	3	4	5
6	6	2	3	4	5
2
1	6	1	3	4	5
2	6	 1	2	4	5
3	 6	1	2	3	 5
7	6	1	2	3	7
6
3
2
1
2
3
6
Total no. of page faults : 10
```

Result

Thus page replacement was implemented using FIFO algorithm.

Ex.no:: 12b **LRU PAGE REPLACEMENT**

Date:

Aim:

To implement demand paging for a reference string using LRU method.

Algorithm:

```
    1. Get length of the reference string, say len.
    2. Get reference string and store it in an array, say rs.
    3. Get number of frames, say nf.
    4. Create access array to store counter that indicates a measure of recent usage.
    5. Create a function arrmin that returns position of minimum of the given array.
    6. Initalize frame array upto length nf to -1.
    7. Initialize position of the page replacement, say j to 0.
    8. Initialize freq to 0 to track page frequency
    9. Initialize no. of page faults, say count to 0.
    10. For each page in reference string in the given order, examine:
        a. Check whether page exist in the frame array.
        b. If page exist in memory then
            i. Store incremented freq for that page position in access array.
        c. If page does not exist in memory then
            i. Check for any empty frames.
            ii. If there is an empty frame,
                 Assign that frame to the page
                 Store incremented freq for that page position in access array.
                 Increment count.
        iii. If there is no free frame then
             Determine page to be replaced using arrmin function.
             Store incremented freq for that page position in access array.
             Increment count.
        iv. Display pages in frame array.
    11. Print count.
    12. Stop
```

Program:

```
#include <stdio.h>
int arrmin(int[], int);
```

```
main()
{
        int i,j,len,rs[50],frame[10],nf,k,avail,count=0;
        int access[10], freq=0, dm;
        printf("Length of Reference string : ");
        scanf("%d", &len);
        printf("Enter reference string :\n");
        for(i=1; i<=len; i++)
        scanf("%d", &rs[i]);
        printf("Enter no. of frames : ");
        scanf("%d", &nf);
        for(i=0; i<nf; i++)
        frame[i] = -1;
        j = 0;
        printf("\nRef. str Page frames");
        for(i=1; i<=len; i++)
        {
                printf("\n%4d\t", rs[i]);
                avail = 0;
                for(k=0; k<nf; k++)
                {
                        if(frame[k] == rs[i])
                        {
                                avail = 1;
                                access[k] = ++freq;
                                break;
                        }
                }
                if(avail == 0)
                {
                        dm = 0;
```

```
                    for(k=0; k<nf; k++)
                    {
                        if(frame[k] == -1)
                        dm = 1;
                        break;
                    }
                    if(dm == 1)
                    {
                        frame[k] = rs[i];
                        access[k] = ++freq;
                        count++;
                    }
                    else
                    {
                        j = arrmin(access, nf);
                        frame[j] = rs[i];
                        access[j] = ++freq;
                        count++;
                    }
                    for(k=0; k<nf; k++)
                    printf("%4d", frame[k]);
                }
        }
        printf("\n\nTotal no. of page faults : %d\n", count);
}
int arrmin(int a[], int n)
{
        int i, min = a[0];
        for(i=1; i<n; i++)
        if (min > a[i])
        min = a[i];
        for(i=0; i<n; i++)
        if (min == a[i])
        return i;
}
```

Output:

```
$ gcc lrupr.c
$ ./a.out
Length of Reference string : 20
Enter reference string :
1  2  3  4  2  1  5  6  2  1  2  3  7  6  3  2  1  2  3  6
Enter no. of frames : 5
Ref. str Page frames
1       1       -1      -1      -1      -1
2       1       2       -1      -1      -1
3       1       2       3       -1      -1
4       1       2       3       4       -1
2
1
5       1       2       3       4       5
6       1       2       6       4       5
2
1
2
3       1       2       6       3       5
7       1       2       6       3       7
6
3
2
1
2
```

```
3
6
Total no. of page faults: 8
```

Result

Thus page replacement was implemented using LRU algorithm.

Ex.no: 12c LFU PAGE REPLACEMENT (OPTIMAL)

Date:

Aim:

To implement demand paging for a reference string using LFU method.

Algorithm:

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

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: $ gcc lfupr.c

```
$ ./a.out
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 page replacement was implemented using LFU algorithm.

Ex.No: 13 **FILE ORGANIZATION TECHNIQUES**

Ex.no: 13a **Implementation of Single level directory structure**

Date:

Aim:

To write a C program to implement Single level directory structure.

Agorithm:

1. Get the number and names of directories.
2. Get the number of files in each directory.
3. Read the names of files in each directory.
4. If the entered name exists read a new name.
5. Display directories along with its files.

Program:

```
#include<stdio.h>
#include<string.h>
main()
{
        int master,s[20],i,j,k;
        char f[20][20][20],d[20][20];
        printf("Enter number of directories: ");
        scanf("%d",&master);
        printf("Enter names of directories : ");
        for(i=0;i<master;i++)
                scanf("%s",d[i]);
        for(i=0;i<master;i++)
        {
                printf("Enter number of files in directory %d  :",i+1);
                scanf("%d",&s[i]);
        }
        for(i=0;i<master;i++)
        {
                printf("Enter names of files in directory %d  :",i+1);
                for(j=0;j<s[i];j++)
                {
                        scanf("%s",f[i][j]);
                        for(k=0;k<j;k++)
                        {
                        if(strcmp(f[i][j],f[i][k])==0)
                        {
                                printf("File Exists...");
                                printf("Enter new file name");
                                scanf("%s",f[i][j]);
                        }
                        }
                }
                printf("\n");
        }
        printf("Directory\t Size\t File name\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\t\t",f[i][j]);
                printf("\n");
        }
        printf("\t\n");
}
```

Output:

```
student@acew:~$ ./a.out
Enter number of directories: 3
Enter names of directories : dir1
dir2
dir3
Enter number of files in directory 1  :2
Enter number of files in directory 2  :2
Enter number of files in directory 3  :2
Enter names of files in directory 1  :file1
file2
```

```
        Enter names of files in directory 2  :f1
        f2
        Enter names of files in directory 3  :f3
        f4
        Directory         Size    File name
         *************************
        dir1              2       file1          file2
        dir2              2       f1             f2
        dir3              2       f3             f4
student@acew:~$
```

Result:

Thus the C program to implement Single level directory structure has been implemented and verified.

Ex.no: 13b Implementation of Two level directory structures

Aim:

To write a C program to implement Two level directory structure.

Agorithm:

1. Get the number and names of master file directories.

2. Get the number of user file directories.
3. Read the names of files in each user file directory.
4. If the entered name exists read a new name.
5. Display master file directories along with user file directories and the files inside each user file directory.

Program:

```
#include<stdio.h>
struct st
{
	char dname[20],sdname[20][20],fname[20][20][20];
	int ds,sds[20];
}dir[20];
main()
{
	int i,j,k,n;
	printf("Enter number of master file directories  :");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
		printf("Enter Name of directory %d",i+1);
		scanf("%s",dir[i].dname);
		printf("Enter number of user file directories :");
		scanf("%d",&dir[i].ds);
		for(j=0;j<dir[i].ds;j++)
		{
			printf("Enter user file directory 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("\n Master dir name\tsize\t sub dir name\t size\t files\n");
            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:

```
student@acew:~$ ./a.out
Enter number of master file directories  :1
Enter Name of directory 1dir1
Enter number of user file directories :3
Enter user file directory name and size uf1
2
Enter file name :f1
Enter file name :f2
Enter user file directory name and size uf2
3
Enter file name :f3
```

```
        Enter file name :f4
        Enter file name :f5
        Enter user file directory name and size uf3
        2
        Enter file name :f6
        Enter file name :f7
         Master      size            sub dir       size          files
*****************************************************
        dir1                3          uf1                2          f1        f2
                                       uf2                3          f3        f4        f5
                                       uf3                2          f6        f7
        student@acew:~$
```

Result:

Thus the C program to implement Two level directory structure has been implemented and verified.

Ex:no 14 **FILE ALLOCATION STRATEGIES**

Ex.no. 14a **Sequential file allocation**

Date:

Aim:

Write a C Program to implement Sequential File Allocation method.

Algorithm:

```
1: Start the program.
2: Get the number of memory partition and their sizes.
3: Get the number of processes and values of block size for each process.
4: First fit algorithm searches all the entire memory block until a hole which is
       big enough is encountered. It allocates that memory block for the requesting
       process.
5: Best-fit algorithm searches the memory blocks for the smallest hole which can
       be allocated to requesting process and allocates if.
6: Worst fit algorithm searches the memory blocks for the largest hole and
       allocates it to the process.
7: Analyses all the three memory management techniques and display the best
     algorithm which utilizes the memory resources effectively and efficiently.
8: Stop the program.
```

Program:

```
#include<stdio.h>
#include<conio.h>
main()
{
 int n,i,j,b[20],sb[20],t[20],x,c[20][20];
 clrscr();
 printf("Enter no.of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
        printf("Enter no. of blocks occupied by file%d",i+1);
        scanf("%d",&b[i]);
        printf("Enter the starting block of file%d",i+1);
        scanf("%d",&sb[i]);
        t[i]=sb[i];
        for(j=0;j<b[i];j++)
                c[i][j]=sb[i]++;
```

```
            }
 printf("Filename\tStart block\tlength\n");
 for(i=0;i<n;i++)
            printf("%d\t  %d \t%d\n",i+1,t[i],b[i]);
 printf("Enter file name:");
 scanf("%d",&x);
 printf("File name is:%d",x);
 printf("length is:%d",b[x-1]);
 printf("blocks occupied:");
 for(i=0;i<b[x-1];i++)
            printf("%4d",c[x-1][i]);
 getch();
}
```

Output:

```
Enter no.of files: 2
Enter no. of blocks occupied by file1 4
Enter the starting block of file1 2
 Enter no. of blocks occupied by file2 10
Enter the starting block of file2 5
Filename        Start block     length
 1                    2               4
2                     5               10
Enter file name: rajesh
 File name is:12803 length is:0blocks occupied
```

Result:

Thus the C Program for Sequential File Allocation method is implemented

Ex. no: 14b **Indexed file allocation**

Date:

Aim: Write a C Program to implement Indexed File Allocation method.

Algorithm:

1: Start.

2: Let n be the size of the buffer

```
3: check if there are any producer
4: if yes check whether the buffer is full
5: If no the producer item is stored in the buffer
 6:  If the buffer is full the producer has to wait
7:  Check there is any cosumer. If yes check whether the buffer is empty
 8:  If no the consumer consumes them from the buffer
 9:  If the buffer is empty, the consumer has to wait.
 10: Repeat checking for the producer and consumer till required
11: Terminate the process.
```

Program:

```
#include<stdio.h>
#include<conio.h>
main()
{
 int n,m[20],i,j,sb[20],s[20],b[20][20],x;
 clrscr();
 printf("Enter no. of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {        printf("Enter starting block and size of file%d:",i+1);
          scanf("%d%d",&sb[i],&s[i]);
          printf("Enter blocks occupied by file%d:",i+1);
          scanf("%d",&m[i]);
          printf("enter blocks of file%d:",i+1);
          for(j=0;j<m[i];j++)
                    scanf("%d",&b[i][j]);
 } printf("\nFile\t index\tlength\n");
 for(i=0;i<n;i++)
 {
          printf("%d\t%d\t%d\n",i+1,sb[i],m[i]);
 }printf("\nEnter file name:");
 scanf("%d",&x);
 printf("file name is:%d\n",x);
 i=x-1;
 printf("Index is:%d",sb[i]);
 printf("Block occupied are:");
 for(j=0;j<m[i];j++)
            printf("%3d",b[i][j]);
 getch();
}
```

Output:

```
Enter no.of files: 2
Enter no. of blocks occupied by file1 4
Enter the starting block of file1 2
 Enter no. of blocks occupied by file2 10
Enter the starting block of file2 5
Filename        Start block     length
 1                    2               4
2                     5               10
Enter file name: rajesh
 File name is:12803 length is:0blocks occupied
```

Result:

Thus the C Program for Indexed File Allocation method is implemented.

Ex. no: 14c **Linked file allocation**

Date:

Aim:

Write a C Program to implement Linked File Allocation method.

Algorithm:

1. Create a queue to hold all pages in memory

2. When the page is required replace the page at the head of the queue
3. Now the new page is inserted at the tail of the queue
4. Create a stack
5. When the page fault occurs replace page present at the bottom of the stack
6. Stop the allocation.

Program:

```
#include<stdio.h>
#include<conio.h>
struct file
{
 char fname[10];
 int start,size,block[10];
}f[10];
main()
{
 int i,j,n;
 clrscr();
 printf("Enter no. of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
 printf("Enter file name:");
 scanf("%s",&f[i].fname);
 printf("Enter starting block:");
 scanf("%d",&f[i].start);
 f[i].block[0]=f[i].start;
 printf("Enter no.of blocks:");
 scanf("%d",&f[i].size);
 printf("Enter block numbers:");
 for(j=1;j<=f[i].size;j++)
 {
```

```
            scanf("%d",&f[i].block[j]);
 }
 }
 printf("File\tstart\tsize\tblock\n");
 for(i=0;i<n;i++)
 {
            printf("%s\t%d\t%d\t",f[i].fname,f[i].start,f[i].size);
            for(j=1;j<=f[i].size-1;j++)
                        printf("%d--->",f[i].block[j]);
            printf("%d",f[i].block[j]);
            printf("\n");
 }
 getch();
}
```

Output:

```
Enter no. of files:2
Enter file name:venkat
Enter starting block:20
Enter no.of blocks:6
Enter block numbers: 4
12
15
45
32
25
Enter file name:rajesh
Enter starting block:12
Enter no.of blocks:5
Enter block numbers:6
5
4
```

```
3
2
File    start   size    block
venkat  20      6       4--->12--->15--->45--->32--->25
rajesh  12      5       6--->5--->4--->3--->2
```

Result:

Thus the C Program for Linked File Allocation method is implemented.

Ex:No 15 IMPLEMENTATION OF VARIOUS DISK SCHEDULING ALGORITHMS

Aim: To Write a C program to simulateFCFS disk scheduling algorithms

Date:

15 a) FCFS DISK SCHEDULING ALGORITHM

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 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>

int main()

{

        int queue[20],n,head,i,j,k,seek=0,max,diff;

        float avg;

        printf("Enter the max range of disk\n");

        scanf("%d",&max);

        printf("Enter the size of queue request\n");

        scanf("%d",&n);

        printf("Enter the queue of disk positions to be read\n");

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

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

        printf("Enter the initial head position\n");

        scanf("%d",&head);

        queue[0]=head;

        for(j=0;j<=n-1;j++)

        {

                diff=abs(queue[j+1]-queue[j]);

                seek+=diff;

                printf("Disk head moves from %d to %d with seek %d\n", queue[j],queue[j+1],diff);

        }

        printf("Total seek time is %d\n",seek);

        avg=seek/(float)n;

        printf("Average seek time is %f\n",avg);

        return 0;
```

}

Output:

```
Enter the max range of disk 200
Enter the size of queue request 8
Enter the queue of disk positions to be read
90 120 35 122 38 128 65 68
Enter the initial head position 50
Disk head moves from 50 to 90 with seek 40
Disk head moves from 90 to 120 with seek 30
Disk head moves from 120 to 35 with seek 85
Disk head moves from 35 to 122 with seek 87
Disk head moves from 122 to 38 with seek 84
Disk head moves from 38 to 128 with seek 90
Disk head moves from 128 to 65 with seek 63
Disk head moves from 65 to 68 with seek 3
Total seek time is 482
Average seek time is 60.250000
```

Result:

Thus the C program for SCAN disk scheduling was written and executed successfully

15 b) SCAN DISK SCHEDULING ALGORITHM

Aim:

To Write a C program to simulate SCAN disk scheduling algorithm.

Description

In the SCAN algorithm, the disk arm starts at one end of the disk and moves toward the other end,servicing requests as it reaches each cylinder,until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk. The SCAN algorithm is sometimes called the elevator algorithm, since the disk arm behaves just like an elevator in a building, first servicing all the requests going up and then reversing to service requests the other way.

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>

int main()

{

        int queue[20],n,head,i,j,k,seek=0,max,diff,temp,queue1[20],queue2[20],

                temp1=0,temp2=0;

        float avg;

        printf("Enter the max range of disk\n");

        scanf("%d",&max);

        printf("Enter the initial head position\n");

        scanf("%d",&head);

        printf("Enter the size of queue request\n");

        scanf("%d",&n);

        printf("Enter the queue of disk positions to be read\n");

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

        {

                scanf("%d",&temp);

                if(temp>=head)
        {
                queue1[temp1]=temp;
                temp1++;
        }
        else
        {
                queue2[temp2]=temp;
                temp2++;
        }
}
for(i=0;i<temp1-1;i++)
{
        for(j=i+1;j<temp1;j++)
        {
                if(queue1[i]>queue1[j])
                {
                        temp=queue1[i];
                        queue1[i]=queue1[j];
                        queue1[j]=temp;
                }
        }
}
for(i=0;i<temp2-1;i++)
{
        for(j=i+1;j<temp2;j++)
        {
                if(queue2[i]<queue2[j])
                {
                        temp=queue2[i];
                        queue2[i]=queue2[j];
                        queue2[j]=temp;
                    }
                }
            }
            for(i=1,j=0;j<temp1;i++,j++)
            queue[i]=queue1[j];
            queue[i]=max;
            for(i=temp1+2,j=0;j<temp2;i++,j++)
            queue[i]=queue2[j];
            queue[i]=0;
            queue[0]=head;
            for(j=0;j<=n+1;j++)
            {
                diff=abs(queue[j+1]-queue[j]);
                seek+=diff;
                 printf("Disk head moves from %d to %d with seek %d\n”, queue[j],queue[j+1],diff);
             }
            printf("Total seek time is %d\n",seek);
            avg=seek/(float)n;
            printf("Average seek time is %f\n",avg);
            return 0;
}
```

Output:

```
     Enter the max range of disk 200
    Enter the initial head position 50
    Enter the size of queue request 8
    Enter the queue of disk positions to be read
    90 120 35 122 38 128 65 68
```

```
Disk head moves from 50 to 65 with seek   15
Disk head moves from 65 to 68 with seek 3
Disk head moves from 68 to 90 with seek   22
Disk head moves from 90 to 120 with seek    30
Disk head moves from 120 to 122 with seek   2
Disk head moves from 122 to 128 with seek   6
Disk head moves from 128 to 200 with seek    72
Disk head moves from 200 to 38 with seek     162
Disk head moves from 38 to 35 with seek    3
Disk head moves from 35 to 0 with seek   35
Total seek time is 350
Average seek time is 43.750000
```

Result:

Thus the C program for SCAN disk scheduling was written and executed successfully

15 c) C-SCAN DISK SCHEDULING ALGORITHM

Aim:

To write a C program to implement C-SCAN Disk Scheduling Algorithm

Description

Circular SCAN (C-SCAN) scheduling is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip .The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one

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. The head services only in the right direction from 0 to the size of the disk.
3. While moving in the left direction do not service any of the tracks.
4. When we reach the beginning(left end) reverse the direction.
5. While moving in the right direction it services all tracks one by one.
6. While moving in the right direction calculate the absolute distance of the track from the head.
7. Increment the total seek count with this distance.
8. Currently serviced track position now becomes the new head position.
9. Go to step 6 until we reach the right end of the disk.
10. If we reach the right end of the disk reverse the direction and go to step 3 until all tracks in the request array have not been serviced.

Program:

```
#include<stdio.h>
int main()
{
        int queue[20],n,head,i,j,k,seek=0,max,diff,temp,queue1[20],queue2[20],
                temp1=0,temp2=0;
        float avg;
        printf("Enter the max range of disk\n");
        scanf("%d",&max);
        printf("Enter the initial head position\n");
        scanf("%d",&head);
        printf("Enter the size of queue request\n");
        scanf("%d",&n);
        printf("Enter the queue of disk positions to be read\n");
        for(i=1;i<=n;i++)
        {
                scanf("%d",&temp);
                if(temp>=head)
                {
                        queue1[temp1]=temp;
                        temp1++;
                }
                else
                {
                        queue2[temp2]=temp;
                        temp2++;
                }
        }
        for(i=0;i<temp1-1;i++)
```

```
	{
		for(j=i+1;j<temp1;j++)
		{
			if(queue1[i]>queue1[j])
			{
				temp=queue1[i];
				queue1[i]=queue1[j];
				queue1[j]=temp;
			}
		}
	}
	for(i=0;i<temp2-1;i++)
	{
		for(j=i+1;j<temp2;j++)
		{
			if(queue2[i]>queue2[j])
			{
				temp=queue2[i];
				queue2[i]=queue2[j];
				queue2[j]=temp;
			}
		}
	}
	for(i=1,j=0;j<temp1;i++,j++)
	queue[i]=queue1[j];
	queue[i]=max;
	queue[i+1]=0;
	for(i=temp1+3,j=0;j<temp2;i++,j++)
	queue[i]=queue2[j];
	queue[0]=head;
	for(j=0;j<=n+1;j++)
	{
		diff=abs(queue[j+1]-queue[j]);
		seek+=diff;
	printf("Disk head moves from %d to %d with seek %d\n", queue[j],queue[j+1],diff);		}
	printf("Total seek time is %d\n",seek);
	avg=seek/(float)n;
	printf("Average seek time is %f\n",avg);
	return 0;
}
```

Output:

```
	Enter the max range of disk 200

	Enter the initial head position 50

	Enter the size of queue request 8
```

```
Enter the queue of disk positions to be read
90 120 35 122 38 128 65 68
Disk head moves from 50 to 65 with seek  15
Disk head moves from 65 to 68 with seek  3
Disk head moves from 68 to 90 with seek  22
Disk head moves from 90 to 120 with seek  30
Disk head moves from 120 to 122 with seek  2
Disk head moves from 122 to 128 with seek   6
Disk head moves from 128 to 200 with seek  72
Disk head moves from 200 to 0 with seek  200
Disk head moves from 0 to 35 with seek   35
Disk head moves from 35 to 38 with seek  3
Total seek time is 388
Average seek time is 48.500000
```

Result:

Thus the C program for C- SCAN disk scheduling was written and executed successfully

Ex: No 16 INSTALL LINUX OPERATING SYSTEM USING VMWARE.

Aim:

To install any guest operating system like linux using VMware

Description:

Installing a Linux operating System using a VMware consists of two steps.

Part 1: Prepare a Computer for Virtualization

Part 2: Create a Virtual Machine

Scenario:

Personal computing power and resources have increased tremendously over the last 5 to 10 years. One of the benefits of having access to multicore processors and large amounts of RAM memory is the ability to use virtualization on a personal computer. With virtualization, a user can run multiple virtual computers on one physical computer or server. Virtual computers that run within a physical computer system are called virtual machines. Today entire computer networks are being implemented where all of the end user computer stations are actually virtual machines run off of a centralized server. Anyone with a modern computer and operating system has the ability to run virtual machines from the desktop.

Part 1: Prepare a Computer for Virtualization

In Part 1, you will download and install virtualization software, and acquire a bootable image of a Linux distribution.

Step 1: Download and install the free VMware player.

There are two excellent virtualization programs that you can download and install for free, VMware Player and VirtualBox. In this lab, you will use the VMware player.

a. Go to http://vmware.com, hover the cursor over **Downloads**, and search for **Free Product Downloads**.

b. Under **Free Product Downloads**, click **Player**.

The VMware Player has 32-bit and 64-bit versions for Windows and Linux. To download the software, you must register a free user account with VMware.

Note: The Linux version of VMware Player might work on Mac OS X; if not, http://virtualbox.org has a free version of its VirtualBox software that will run on Mac OS X.

c. When you have downloaded the VMware Player installation file, run the installer and accept the default installation settings.

Step 2: Download a bootable image of Linux.

You need an operating system to install on your virtual machine's virtual hardware. Linux is a suitable choice for an operating system, because most of the distributions are free to download and run. It also allows you to explore an operating system that might be unfamiliar to you.

a. To download Linux, you first must select a distribution such as Mint, Ubuntu, Fedora, Suse, Debian, or CentOS. (There are many others to choose from.) In this lab, the instructions follow an installation of Linux Mint.

b. Go to http://linuxmint.com, hover over the **Download** button, and click **Linux Mint 16** (or current version number).

c. Scroll down the page until you see the version of the Mint code name, **Cinnamon** (or the current code name). Choose either the 32-bit or 64-bit version, depending on your current operating system platform, and click the link.

d. A new web page will appear. Select a download mirror site from which to download the operating system. Click a mirror site to activate the Linux file download. When prompted by the browser, choose to save the Linux .iso file to a local directory.

e. When the file has finished downloading, you should have a Linux Mint .iso bootable image.

Part 2: Create a Virtual Machine

In Part 2, using the VMware Player, you will create a virtual machine and customize its virtual hardware. Then, using the Linux Mint .iso file that you downloaded in Part 1, you will install the Linux Mint operating system on the virtual machine.

Step 1: Create a virtual computer and customize the virtual hardware.

a. Open **VMware Player**, and click **Create a New Virtual Machine**.

b. A new window will appear. Select **I will install the operating system later. The virtual machine will be created with a blank hard disk** option. Click **Next**.

c. A new window will appear. Select **Linux** as the guest operating system. Under **version**, you may notice that Mint is not listed. In this case, select an alternate Linux distribution, one that is closely related to Mint (like Ubuntu). Lastly, select either the 32-bit or 64-bit version and click **Next**.

d. A new window will appear. Select a name and storage location for the virtual machine.

e. A new window will appear. Select the maximum size of the virtual hard drive storage. You can also decide whether or not to store the virtual hard drive in a single file or in multiple files.

f. When a new window appears, click **Finish** to finish creating the virtual machine hardware, or click **Customize Hardware** to customize the virtual hardware. For example, you can specify the amount of RAM, how many CPU cores you want, and add additional drives and peripheral components (see video tutorial).

g. When you have customized and completed the process, you are now ready to run your virtual machine and install an operating system.

Step 2: Install the Linux operating system on the virtual computer.

To install from an .iso bootable image file, complete the following steps:

a. In VMware Player, click **Edit virtual machine settings**. If you have multiple virtual machines created, you must first select which one you intend to edit.

b. In the pop-up window, under the **Hardware** tab, select **CD/DVD (SATA)**, and on the right side (under **Connections**), select the **Use ISO image file** option, and click **Browse** to search through your file system for the Linux Mint .iso image file downloaded in Part 1.

Selecting the Linux .iso file causes it to be automatically mounted to the CD/DVD drive when the virtual machine boots, which, in turn, causes the system to boot and run the Linux installation image.

c. Select **Network Adapter**, and click **OK**.

The network adapter is currently set to NAT mode, which allows your virtual machine to access the network through the host computer system by translating all Internet requests through the VMware player, much like a gateway router. This gives your virtual machine a private network address separate from your home network.

(Optional) To have your virtual machine on the same network as the computer hosting the virtual machine, select the **Bridged** option and click **Configure Adapters** to identify to which physical network adapter the virtual machine should bridge.

Note: This is especially important on laptops that have both wireless and wired network adapters that can connect to the network.

d. When you are finished click **OK**.

e. Click **Play virtual machine** to launch your virtual machine and boot to Linux.

When the boot process has completed you should be presented with a Linux Mint desktop.

f. (Optional) To permanently install Linux Mint on the virtual machine hard disk drive, on the desktop, double-click the **Install Linux Mint disk** icon and follow the installation procedure (see video).

During the installation, you will be presented with a warning message that installing to the drive will erase everything on the disk drive; this refers to the virtual disk drive, not the host computer physical disk drive.

g. When you have finished the installation procedure, you should have a Linux Mint virtual machine that you can run in a window alongside your host computer system.

Result:

Thus the guest operating system like linux using VMware is successfully installed.

LIST OF EXPERIMENTS (Beyond the Syllabus)

Ex. no: 1 Write a C program to simulate the concept of Dining-Philosophers problem.

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

Date:

Description:

The dining-philosopher's problem is considered a classic synchronization problem because it is an example of a large class of concurrency-control problems. It is a simple representation of the need to allocate several resources among several processes in a deadlock-free and starvation-free manner. Consider five philosophers who spend their lives thinking and eating. The philosophers share a circular table surrounded by five chairs, each belonging to one philosopher. In the center of the table is a bowl of rice, and the table is laid with five single chopsticks. When a philosopher thinks, she does not interact with her colleagues. From time to time, a philosopher gets hungry and tries to pick up the two chopsticks that are closest to her (the chopsticks that are between her and her left and right neighbors). A philosopher may pick up only one chopstick at a time. Obviously, she cam1ot pick up a chopstick that is already in the hand of a neighbor. When a hungry philosopher has both her chopsticks at the same time, she eats without releasing her chopsticks. When she is finished eating, she puts down both of her chopsticks and starts thinking again. The dining-philosophers problem may lead to a deadlock situation and hence some rules have to be framed to avoid the occurrence of deadlock.

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..");
 }
```

```
else
{
for(i=0;i<howhung;i++)
{
 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 a time\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
rintf("\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: 5
            How 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 eat
             P 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 was written and executed successfully.

Ex. no: 2 Write a C program to simulate the MVT and MFT memory management techniques

Aim: To Write a C program to simulate the MVT and MFT memory management techniques

Date:

Description:

MFT (Multiprogramming with a Fixed number of Tasks) is one of the old memory management techniques in which the memory is partitioned into fixed size partitions and each job is assigned to a partition. The memory assigned to a partition does not change. MVT

(Multiprogramming with a Variable number of Tasks) is the memory management technique in which each job gets just the amount of memory it needs. That is, the partitioning of memory is dynamic and changes as jobs enter and leave the system. MVT is a more ``efficient" user of resources. MFT suffers with the problem of internal fragmentation and MVT suffers with external fragmentation

Program:

MFT MEMORY MANAGEMENT TECHNIQUE

```
 #include<stdio.h> #include<conio.h>
main()
 {
int ms, bs, nob, ef,n, mp[10],tif=0; int i,p=0;
clrscr();
printf("Enter the total memory available (in Bytes) -- ");
scanf("%d",&ms);
printf("Enter the block size (in Bytes) -- ");
scanf("%d", &bs);
nob=ms/bs;
ef=ms - nob*bs;
 printf("\nEnter the number of processes -- ");
 scanf("%d",&n); for(i=0;i<n;i++)
{ printf("Enter memory required for process %d (in Bytes)-- ",i+1);
 scanf("%d",&mp[i]);
 }
printf("\nNo. of Blocks available in memory -- %d",nob);
 printf("\n\nPROCESS\tMEMORY REQUIRED\t ALLOCATED\tINTERNAL
FRAGMENTATION");
 for(i=0;i<n && p<nob;i++)
 {
printf("\n %d\t\t%d",i+1,mp[i]);
 if(mp[i] > bs) printf("\t\tNO\t\t---");
else
```

```
 {
printf("\t\tYES\t%d",bs-mp[i]);
 tif = tif + bs-mp[i]; p++;
}
 }
 if(i<n)
printf("\nMemory is Full, Remaining Processes cannot be accomodated");
 printf("\n\nTotal Internal Fragmentation is %d",tif);
 printf("\nTotal External Fragmentation is %d",ef);
 getch();
}
```

Output:

```
 Enter the total memory available (in Bytes) -- 1000
Enter the block size (in Bytes)-- 300
Enter the number of processes – 5
 Enter memory required for process 1 (in Bytes) – 275
 Enter memory required for process 2 (in Bytes) – 400
 Enter memory required for process 3 (in Bytes) – 290
 Enter memory required for process 4 (in Bytes) -- 293 Enter memory required for
process 5 (in Bytes) -- 100 No. of Blocks available in memory – 3
     PROCESS    MEMORY          ALLOCATED       INTERNAL
                REQUIRED                        FRAGMENTATION
        1          275             YES               25
        2          400             NO                --
        3          290             YES               10
        4          293             YES               7

 Memory is Full, Remaining Processes cannot be accommodated
Total Internal Fragmentation is 42
Total External Fragmentation is 100
```

MVT MEMORY MANAGEMENT TECHNIQUE

```
 #include<stdio.h>
 #include<conio.h>
main()
{
int ms,mp[10],i, temp,n=0; char ch = 'y';
 clrscr();
 printf("\nEnter the total memory available (in Bytes)-- ");
scanf("%d",&ms); temp=ms; for(i=0;ch=='y';i++,n++)
 {
printf("\nEnter memory required for process %d (in Bytes) -- ",i+1);
 scanf("%d",&mp[i]);
 if(mp[i]<=temp)
 {
printf("\nMemory is allocated for Process %d ",i+1);
 temp = temp - mp[i];
 }
 Else
 {
printf("\nMemory is Full");
break;
}
printf("\nDo you want to continue(y/n) -- ");
scanf(" %c", &ch);
}
printf("\n\nTotal Memory Available -- %d", ms);
 printf("\n\n\tPROCESS\t\t MEMORY ALLOCATED ");
for(i=0;i<n;i++) printf("\n \t%d\t\t%d",i+1,mp[i]);
 printf("\n\nTotal Memory Allocated is %d",ms-temp);
printf("\nTotal External Fragmentation is %d",temp); 15
getch();
 }
```

Output:

```
 Enter the total memory available (in Bytes) – 1000
 Enter memory required for process 1 (in Bytes) – 400
 Memory is allocated for Process 1
 Do you want to continue(y/n) – y
Enter memory required for process 2 (in Bytes) – 275
 Memory is allocated for Process 2
 Do you want to continue(y/n) -- y
Enter memory required for process 3 (in Bytes) – 550

Memory is Full Total Memory Available -- 1000
PROCESS              MEMORY ALLOCATED
      1                   400
      2                   275
 Total Memory Allocated is 675
 Total External Fragmentation is 325
```

Result:

Thus the program was written and executed successfully.

Ex. no: 3 Write a C program to simulate Optimal page replacement algorithms

Aim: To simulate the optimal page replacement algorithm

Date:

Description:

Optimal page replacement algorithm has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly. The basic idea is to replace the page that will not be used for the longest period of time. Use of this page-replacement algorithm guarantees the lowest possible page fault rate for a fixed number of frames. Unfortunately, the optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.

Program:

```
 #include<stdio.h>
 int n;
main()
 {
 int seq[30],fr[5],pos[5],find,flag,max,i,j,m,k,t,s;
 int count=1,pf=0,p=0; float pfr;
clrscr();
 printf("Enter maximum limit of the sequence: ");
 scanf("%d",&max); printf("\nEnter the sequence: ");
for(i=0;i<max;i++) scanf("%d",&seq[i]);
 printf("\nEnter no. of frames: ");
 scanf("%d",&n);
 fr[0]=seq[0];
pf++;
 printf("%d\t",fr[0]);
 i=1;
while(count<n)
 {
flag=1;
p++;
 for(j=0;j<i;j++)
 {
if(seq[i]==seq[j])
 flag=0;
 }
 if(flag!=0)
 {
 fr[count]=seq[i];
 printf("%d\t",fr[count]);
 count++;
 pf++;
 }
 i++;
}
printf("\n");
 for(i=p;i<max;i++)
 {
 flag=1; for(j=0;j<n;j++)
 {
if(seq[i]==fr[j])
flag=0;
 }
 if(flag!=0)
{
for(j=0;j<n;j++)
{
m=fr[j];
 for(k=i;k<max;k++)
 {
 if(seq[k]==m)
 {
pos[j]=k;
break;
}
else pos[j]=1;
}
 for(k=0;k<n;k++)
{
 if(pos[k]==1)
 flag=0;
}
 if(flag!=0)
s=findmax(pos);
 if(flag==0)
{
for(k=0;k<n;k++)
 {
 if(pos[k]==1)
 {
 s=k; break;
}
 }
 }
 fr[s]=seq[i];
for(k=0;k<n;k++)
printf("%d\t",fr[k]);
 pf++; printf("\n");
 }
 }
 pfr=(float)pf/(float)max;
 printf("\nThe no. of page faults are %d",pf);
 printf("\nPage fault rate %f",pfr);
getch();
}
int findmax(int a[])
 {
 int max,i,k=0; max=a[0];
for(i=0;i<n;i++) { if(max<a[i])
 {
max=a[i];
```

```
 k=i;
 }
 }
return k;
 }
```

Output:

```
 Enter number of page references – 10
 Enter the reference string – 1  2  3   4  5 2  5  2  5  1  4  3
Enter the available no. of frames – 3
The Page Replacement Process is –
 1        -1      -1       PF No. 1
 1         2      -1        PF No. 2
 1         2       3       PF No. 3
 4         2      3        PF No. 4
 5         2       3        PF No. 5
5          2       3
 5         2      3
 5         2       1        PF No. 6
 5         2       4        PF No. 7
5          2       3        PF No. 8
Total number of page faults – 8
```

Result:

Thus the program was written and executed successfully.

Ex. no: 4 Simulate the algorithm for Dead Lock Prevention

Aim:

To implement deadlock prevention technique

Date:

Banker‘s Algorithm:

When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the user request a set of resources, the system must determine whether the allocation of each resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources.

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>
void
main()
{
char job[10][10];
int time[10],avail,tem[10],temp[10]; int safe[10];
int ind=1,i,j,q,n,t;
clrscr();
printf("Enter no of jobs: ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter name and time: ");
scanf("%s%d",&job[i],&time[i]);
}
printf("Enter the available resources:");
scanf("%d",&avail);
for(i=0;i<n;i++)
{
temp[i]=time[i];
tem[i]=i;
}
for(i=0;i<n;i++)
for(j=i+1;j<n;j++)
{
if(temp[i]>temp[j])
{
t=temp[i];
 temp[i]=temp[j];
temp[j]=t; t=tem[i];
tem[i]=tem[j];
tem[j]=t;
}
}
for(i=0;i<n;i++)
{
q=tem[i];
if(time[q]<=avail)
{
safe[ind]=tem[i];
avail=avail-tem[q];
printf("%s",job[safe[ind]]);
ind++;
}
else
{
printf("No safe sequence\n");
}
}
printf("Safe sequence is:");
for(i=1;i<ind; i++)
printf("%s %d\n",job[safe[i]],time[safe[i]]);
getch();
}
```

Output:

```
Enter no of jobs:4
Enter name and time: A 1
Enter name and time: B 4
Enter name and time: C 2
Enter name and time: D 3
Enter the available resources: 20
Safe sequence is: A 1, C 2, D 3, B 4.
```

Result:

Thus the program was written and executed successfully.

VIVA QUESTIONS

1. What is meant by System Calls?

The System Calls acts as a interface to a running program and the Operating system. These system calls available in assembly language instructions.

2. What is System Programs?

System programs provide a convenient environment for program development and execution. Some of these programs are user interfaces to system calls and others are more complex.

3. What is meant by Mainframe Systems?

Mainframe systems are the first computers developed to tackle many commercial and scientific applications. These systems are developed from the batch systems and then multiprogramming system and finally time sharing systems.

4. Define OS.

Operating system is system software which acts as an interface between man and machine.

5. Define Mutual Exclusion.

It is defined as each process accessing the shared data excludes all others from doing simultaneously.

6.What is meant by Co-operating process?

If a process can affect or be affected by the other processes executing in the system, that process which shares data with other process is called as Co-operating process.

7. What is meant by Interrupt?

An Interrupt is an event that alters the sequence in which a processor executes instructions. It is generated by the hardware of the computer System.

8. What are the conditions that must hold for Deadlock Prevention?

i. Mutual Exclusion Condition	iii. Hold and Wait Condition
ii. No Pre-emption condition	iv. Circular Wait Condition.

9. What are the options for breaking a Deadlock?

i. Simply abort one or more process to break the circular wait.

ii. Preempt some resources from one or more of the deadlocked processes.

10 .What is meant by Counting Semaphore?

A Counting Semaphore is a semaphore whose integer value that can range between 0 & 1.

11. What is meant by Binary Semaphore?

A Binary Semaphore is a semaphore with an integer value that can range between 0 and 1. It can be simpler to implement than a counting semaphore, depending on the underlying hardware architecture.

12.What is meant by Race Condition?

A condition, when several processes access and manipulate the same data on currently and the outcome of the execution depends on the particular order in which the access takes place is called as Race condition.

13. What does a solution for Critical-Section Problem must satisfy?

Mutual Exclusion

Bounded Waiting

Progress

14. What is meant by CPU Scheduler?

When the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed .This selection process is carried out by the CPU Scheduler.

15. What is meant by CPU Scheduling?

The process of selecting among the processes in memory that are ready to execute and allocates the CPU to one of them is called as CPU Scheduling.

16. What are the types of Scheduling available?

Preemptive Scheduling

Priority Scheduling

Non- preemptive Scheduling

17.What is meant by Priority Scheduling?

The basic idea here is straight toward. Each process is assigned a priority and the run able process with the highest priority is allowed to run.

18.What is Preemptive Scheduling?

A Scheduling discipline is Pre-emptive if the CPU can be taken away before the process completes.

19.What is Non - Preemptive Scheduling?

A Scheduling discipline is non pre-emptive if once a process has been given the CPU, the CPU cannot be taken away from the process.

20.What are the properties of Scheduling Algorithms?

Throughput

CPU Utilization

Response time

Waiting time

Turnaround time

21.What is meant by First Come, First Served Scheduling?

In this Scheduling, the process that requests the CPU first is allocated the CPU first. This Scheduling algorithm is Non Pre-emptive.

22. What is meant by Shortest Job First Scheduling?

When the CPU is available, it is assigned to the process that has the smallest next CPU burst. This Scheduling algorithm is either Pre-emptive or Non Pre-emptive.

23. What is meant by External Fragmentation and Internal Fragmentation?

External Fragmentation exists when enough total memory space exists to satisfy a request, but it is not contiguous and storage is fragmented into a large number of small holes. The memory allocated to a process may be slightly larger than the requested memory. The difference between these two numbers is called as Internal Fragmentation.

24. What is meant by Paging?.

Paging is a Memory-management scheme that permits the physical -address space of a process to be Non-contiguous.

25. What is meant by Page Fault?

Whenever memory management unit notices that the page is unmapped and causes the CPU to trap to the Operating System. This trap is called as Page Fault.

26. What are the characteristics of Disk Scheduling?

1) Throughput 2) Mean Response Time 3) Variance of Response time

27.What are the different types of Disk Scheduling ?.

Some of the Disk Scheduling are (i).SSTF Scheduling (ii).FCFS Scheduling (iii) SCAN Scheduling (iv).C-SCAN Scheduling (v).LOOK Scheduling.

28.What is meant by SSTF Scheduling?.

SSTF Algorithm selects the request with the minimum seek time from the current head position. and SSTF chooses the pending request to the current head position.

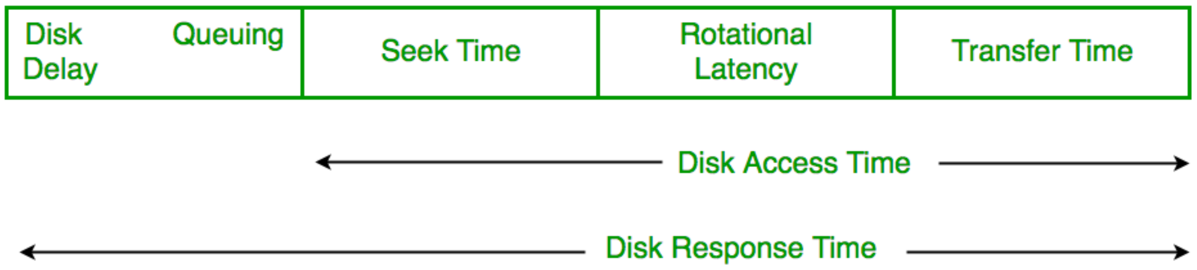
29.What is meant by FCFS Scheduling ?

It is simplest form of Disk Scheduling. This algorithm serves the first come process always and is does not provide Fast service.

30. What is Seek Time, Rotational Latency,Transfer trime, Disk access time?

Seek Time: Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.
Rotational Latency: Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.
Transfer Time: Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.
Disk Access Time: Disk Access Time is:

Disk Access Time = Seek Time + Rotational Latency + Transfer Time



- **Disk Response Time:** Response Time is the average of time spent by a request waiting to perform its I/O operation. *Average Response time* is the response time of the all requests. *Variance Response Time* is measure of how individual request are serviced with respect to average response time. So the disk scheduling algorithm that gives minimum variance response time is better.

1) What is an operating system?

The operating system is a software program that facilitates computer hardware to communicate and operate with the computer software. It is the most important part of a computer system without it computer is just like a box.

2) What is the main purpose of an operating system?

There are two main purposes of an operating system:

- It is designed to make sure that a computer system performs well by managing its computational activities.
- It provides an environment for the development and execution of programs.

3) What are the different operating systems?

- Batched operating systems
- Distributed operating systems

- Timesharing operating systems
- Multi-programmed operating systems
- Real-time operating systems

4) What is a socket?

A socket is used to make connection between two applications. Endpoints of the connection are called socket.

5) What is a real-time system?

Real-time system is used in the case when rigid-time requirements have been placed on the operation of a processor. It contains a well defined and fixed time constraints.

6) What is kernel?

Kernel is the core and most important part of a computer operating system which provides basic services for all parts of the OS.

7) What is monolithic kernel?

A monolithic kernel is a kernel which includes all operating system code is in single executable image.

8) What do you mean by a process?

An executing program is known as process. There are two types of processes:

- Operating System Processes
- User Processes

9) What are the different states of a process?

A list of different states of process:

- New Process
- Running Process
- Waiting Process
- Ready Process
- Terminated Process

10) What is the difference between micro kernel and macro kernel?

Micro kernel: micro kernel is the kernel which runs minimal performance affecting services for operating system. In micro kernel operating system all other operations are performed by processor.

Macro Kernel: Macro Kernel is a combination of micro and monolithic kernel.

11) What is the concept of reentrancy?

It is a very useful memory saving technique that is used for multi-programmed time sharing systems. It provides functionality that multiple users can share a single copy of program during the same period.

12) What is the difference between process and program?

A program while running or executing is known as a process.

13) What is the use of paging in operating system?

Paging is used to solve the external fragmentation problem in operating system. This technique ensures that the data you need is available as quickly as possible.

14) What is the concept of demand paging?

Demand paging specifies that if an area of memory is not currently being used, it is swapped to disk to make room for an application's need.

15) What is the advantage of a multiprocessor system?

As many as processors are increased, you will get the considerable increment in throughput. It is cost effective also because they can share resources. So, the overall reliability increases.

16) What is virtual memory?

Virtual memory is a very useful memory management technique which enables processes to execute outside of memory. This technique is especially used when an executing program cannot fit in the physical memory.

17) What is thrashing?

Thrashing is a phenomenon in virtual memory scheme when the processor spends most of its time in swapping pages, rather than executing instructions.

18) What are the four necessary and sufficient conditions behind the deadlock?

These are the 4 conditions:

1) **Mutual Exclusion Condition**: It specifies that the resources involved are non-sharable.

2) **Hold and Wait Condition**: It specifies that there must be a process that is holding a resource already allocated to it while waiting for additional resource that are currently being held by other processes.

3) **No-Preemptive Condition**: Resources cannot be taken away while they are being used by processes.

4) **Circular Wait Condition**: It is an explanation of the second condition. It specifies that the processes in the system form a circular list or a chain where each process in the chain is waiting for a resource held by next process in the chain.

19) What is a thread?

A thread is a basic unit of CPU utilization. It consists of a thread ID, program counter, register set and a stack.

20) What is FCFS?

FCFS stands for First Come, First Served. It is a type of scheduling algorithm. In this scheme, if a process requests the CPU first, it is allocated to the CPU first. Its implementation is managed by a FIFO queue.

21) What is SMP?

SMP stands for Symmetric MultiProcessing. It is the most common type of multiple processor system. In SMP, each processor runs an identical copy of the operating system, and these copies communicate with one another when required.

22) What is RAID? What are the different RAID levels?

RAID stands for Redundant Array of Independent Disks. It is used to store the same data redundantly to improve the overall performance.

Following are the different RAID levels:

RAID 0 - Stripped Disk Array without fault tolerance

RAID 1 - Mirroring and duplexing

RAID 2 - Memory-style error-correcting codes

RAID 3 - Bit-interleaved Parity

RAID 4 - Block-interleaved Parity

RAID 5 - Block-interleaved distributed Parity

RAID 6 - P+Q Redundancy

23) What is deadlock? Explain.

Deadlock is a specific situation or condition where two processes are waiting for each other to complete so that they can start. But this situation causes hang for both of them.

24) Which are the necessary conditions to achieve a deadlock?

There are 4 necessary conditions to achieve a deadlock:

- **Mutual Exclusion:** At least one resource must be held in a non-sharable mode. If any other process requests this resource, then that process must wait for the resource to be released.
- **Hold and Wait:** A process must be simultaneously holding at least one resource and waiting for at least one resource that is currently being held by some other process.
- **No preemption:** Once a process is holding a resource (i.e. once its request has been granted), then that resource cannot be taken away from that process until the process voluntarily releases it.
- **Circular Wait:** A set of processes { P0, P1, P2, . . ., PN } must exist such that every P[i] is waiting for P[(i + 1) % (N + 1)].

25) What is Banker's algorithm?

Banker's algorithm is used to avoid deadlock. It is the one of deadlock-avoidance method. It is named as Banker's algorithm on the banking system where bank never allocates available cash in such a manner that it can no longer satisfy the requirements of all of its customers.

26) What is the difference between logical address space and physical address space?

Logical address space specifies the address that is generated by CPU. On the other hand physical address space specifies the address that is seen by the memory unit.

27) What is fragmentation?

Fragmentation is a phenomenon of memory wastage. It reduces the capacity and performance because space is used inefficiently.

28) How many types of fragmentation occur in Operating System?

There are two types of fragmentation:

- **Internal fragmentation**: It is occurred when we deal with the systems that have fixed size allocation units.
- **External fragmentation**: It is occurred when we deal with systems that have variable-size allocation units.

29) What is spooling?

Spooling is a process in which data is temporarily gathered to be used and executed by a device, program or the system. It is associated with printing. When different applications send output to the printer at the same time, spooling keeps these all jobs into a disk file and queues them accordingly to the printer.

30) What is the difference between internal commands and external commands?

Internal commands are the built-in part of the operating system while external commands are the separate file programs that are stored in a separate folder or directory.

31) What is semaphore?

Semaphore is a protected variable or abstract data type that is used to lock the resource being used. The value of the semaphore indicates the status of a common resource.

There are two types of semaphore:

- Binary semaphores
- Counting semaphores

32) What is a binary Semaphore?

Binary semaphore takes only 0 and 1 as value and used to implement mutual exclusion and synchronize concurrent processes.

33) What is Belady's Anomaly?

Belady's Anomaly is also called FIFO anomaly. Usually, on increasing the number of frames allocated to a process virtual memory, the process execution is faster, because fewer page faults occur. Sometimes, the reverse happens, i.e., the execution time increases even when more frames are allocated to the process. This is Belady's Anomaly. This is true for certain page reference patterns.

34) What is starvation in Operating System?

Starvation is Resource management problem. In this problem, a waiting process does not get the resources it needs for a long time because the resources are being allocated to other processes.

35) What is aging in Operating System?

Aging is a technique used to avoid the starvation in resource scheduling system.

36) What are the advantages of multithreaded programming?

A list of advantages of multithreaded programming:

- Enhance the responsiveness to the users.
- Resource sharing within the process.
- Economical
- Completely utilize the multiprocessing architecture.

37) What is the difference between logical and physical address space?

Logical address specifies the address which is generated by the CPU whereas physical address specifies to the address which is seen by the memory unit.

After fragmentation

38) What are overlays?

Overlays makes a process to be larger than the amount of memory allocated to it. It ensures that only important instructions and data at any given time are kept in memory.

39) When does trashing occur?

Thrashing specifies an instance of high paging activity. This happens when it is spending more time paging instead of executing.

39) What is caching?

Caching is the processing of utilizing a region of fast memory for a limited data and process. A cache memory is usually much efficient because of its high access speed.

40) What is spooling?

Spooling is normally associated with printing. When different applications want to send an output to the printer at the same time, spooling takes all of these print jobs into a disk file and queues them accordingly to the printer.

41) What is an Assembler?

An assembler acts as a translator for low-level language. Assembly codes written using mnemonic commands are translated by the Assembler into machine language.

42) What are interrupts?

Interrupts are part of a hardware mechanism that sends a notification to the CPU when it wants to gain access to a particular resource. An interrupt handler receives this interrupt signal and “tells” the processor to take action based on the interrupt request.

43) What is GUI?

GUI is short for Graphical User Interface. It provides users with an interface wherein actions can be performed by interacting with icons and graphical symbols. People find it easier to interact with the computer when in a GUI especially when using the mouse. Instead of having to remember and type commands, users click on buttons to perform a process.

44) What is preemptive multitasking?

Preemptive multitasking allows an operating system to switch between software programs. This, in turn, allows multiple programs to run without necessarily taking complete control over the processor and resulting in system crashes.

45) Why partitioning and formatting is a prerequisite to installing an operating system?

Partitioning and formatting create a preparatory environment on the drive so that the operating system can be copied and installed properly. This includes allocating space on the drive, designating a drive name, determining and creating the appropriate file system and structure.

---

46) What is plumbing/piping?

It is the process of using the output of one program as an input to another. For example, instead of sending the listing of a folder or drive to the main screen, it can be piped and sent to a file, or sent to the printer to produce a hard copy.

47) What is NOS?

NOS is short for Network Operating System. It is a specialized software that will allow a computer to communicate with other devices over the network, including file/folder sharing.

48) Differentiate internal commands from external commands.

Internal commands are built-in commands that are already part of the operating system. External commands are separate file programs that are stored in a separate folder or directory.

49) Under DOS, what command will you type when you want to list down the files in a directory, and at the same time pause after every screen output?

a) dir /w
b) dir /p
c) dir /s
d) dir /w /p

Answer: d) dir /w /p

50) How would a file name EXAMPLEFILE.TXT appear when viewed under the DOS command console operating in Windows 98?

The filename would appear as EXAMPL~1.TXT . The reason behind this is that filenames under this operating system are limited to 8 characters when working under DOS environment.

51) What is a folder in Ubuntu?

There is no concept of Folder in Ubuntu. Everything included in your hardware is a FILE.

52) Explain why Ubuntu is safe and not affected by viruses?

- It does not support malicious e-mails and contents, and before any e-mail is opened by users it will go through many security checks
- Ubuntu uses Linux, which is a super secure O.S system
- Unlike other O.S, countless Linux users can see the code at any time and can fix the problem if there is any
- Malware and viruses are coded to take advantage of the weakness in Windows

---

53) Explain what is Unity in Ubuntu? How can you add new entries to the launcher?

In Ubuntu, Unity is the default graphical shell. On the left side of the Ubuntu, it introduces the launcher and Dash to start programs.

In order to add new entries to the launcher, you can create a file name like **.desktop** and then drag the file on the launcher.

54) Explain the purpose of using a libaio package in Ubuntu?

Libaio is Linux Kernel Asynchronous I/O (A/O). A/O allows even a single application thread to overlap I/O operations with other processing, by providing an interface for submitting one or more I/O requests in one system call without waiting for completion. And a separate interface to reap completed I/O operations associated with a given completion group.

55) What is the use of behavior tab in Ubuntu?

Through behaviors tab, you can make many changes on the appearance of the desktop

- Auto-hide the launcher: You can use this option to reveal the launcher when moving the pointer to the defined hot spot.
- Enable workspaces: By checking this option, you can enable workspace
- Add show desktop icon to the launcher: This option is used to display the desktop icon on the launcher

56) What is the meaning of "export" command in Ubuntu?

Export is a command in Bash shell language. When you try to set a variable, it is visible or exported to any subprocess started from that instance of bash. The variable will not exist in the sub-process without the export command.