INDEX

SL.NO.	DATE	NAME OF THE EXPERIMENT	PAGE NO.	SIGNATURE
1		Basics Of Unix Commands		
2a		Programs Using The Following System Calls Of Unix Operating System Fork, Exec, Getpid, Exit, Wait, Close, Stat, Opendir, Readdir		
3		C Programs To Simulate Unix Commands Like Cp, Ls, Grep, Etc.		
4		Shell Programming		
5		5a]Fcfs – Scheduling Algorithm		
		5b] Sjf – Scheduling Algorithm		
		5c] Round Robin Scheduling Algorithm		
		5d] Priority Scheduling algorithm		
6		Implementation of semaphores		
7		Implementation Of Shared Memory And IPC		
8		Bankers Algorithm For Deadlock Avoidance		
9		Implementation Of Deadlock Detection Algorithm		
10		C Program To Implement Threading & Synchronization Applications		

11		Implementation Of The Following Memory Allocation Methods For Fixed Partition A) First Fit B) Worst Fit C) Best Fit		
12		Implementation Of Paging Technique Of Memory Management		
13		Implementation Of The Following Page Replacement Algorithms A) FIFO B) LRU C) OPTIMAL		
14		Implementation Of The Various File Organization Techniques		
15		Implementation Of The Following File Allocation Strategies A) Sequential B) Indexed C) Linked		
CONTENT BEYOND SYLLABUS				
16		Linux OS Case Study		

OPERATING SYSTEM INTRODUCTION

Operating System:

OS is System software and it is defined as an organized collection of software consisting of procedures for operating a computer. It provides an environment for execution of programs and acts as a interface between the user and the hardware of the computer system.

Operating System interacts with user in two ways:

- Operating System commands.
- Enables user to interact directly with operating system.
- Operating System calls provide an interface to a running program and the operating system. System calls in UNIX are written in C.

Features of UNIX:

- Multi-user, Multitasking, timesharing.
- Portability
- Modularity
- File structure
- Security
- Strong network support & advanced graphics.

Features of LINUX:

- An Open-source UNIX like operating system
- Initially created by Linux towards for PC architecture.
- Developer community world-wide contribute to its enhancement and growth

Operating System Services:

1. **Kernel:**

It performs, Control of execution of Processes, Scheduling of Processes, Memory management, File System service, Controlled access to peripherals.

2. **Hardware Transparency**:

Two Modes used here, User Mode and Kernel Mode.

3. Interrupts and Exceptions:

Interrupt: It is a signal by which priorities and instruction-scheduling sequence Can be controlled

Exception:

It is an unexpected event like references to an illegitimate address caused by events external to process.

The Concept of a virtual machine is central to memory management under UNIX.

Result:

Thus the Introduction of Operating System has been studied.

EXNO:1 **BASIC LINUX GENERAL COMMANDS**
DATE:

AIM:

To study the basic UNIX commands.

COMMANDS:

1. TASK : To display the system date and time.
   COMMAND : date.
   SYNTAX : date.
   EXPLANATION: This command displays the current system date and time on the screen.
   OUTPUT : Tue Jun 19 11:37:17 GMT 2007.

2. TASK : To display the current month.
   COMMAND : date.
   SYNTAX : date +%m.
   EXPLANATION: This command displays the current month on the screen.
   OUTPUT : 06.

3. TASK : To display the name of the current month.
   COMMAND : date.
   SYNTAX : date +%h.
   EXPLANATION: This command displays the name of the current month on the screen.

4. TASK : To display the current system date.
   COMMAND : date.
   SYNTAX : date +%d.
   EXPLANATION: This command displays the current system date on the screen.
   OUTPUT : 19.

5. TASK : To display the current system date (year).
   COMMAND : date.
   SYNTAX : date +%y.
   EXPLANATION: This command displays the current year on the screen.
   OUTPUT : 09.

6. TASK : To display the current system time.
   COMMAND : date.

SYNTAX : date +%H.
EXPLANATION: This command displays the current system time (in hours) on the screen.
OUTPUT : 11

7. TASK : To display the current system time.
COMMAND : date.
SYNTAX : date +%M.
EXPLANATION: This command displays the current system time (in minutes) on the screen.
OUTPUT : 43.

8. TASK : To display the current system time.
COMMAND : date.
SYNTAX : date +%S.
EXPLANATION: This command displays the current system time (in seconds) on the screen.
OUTPUT : 15.

9. TASK : To display the calendar of the current month.
COMMAND : calendar.
SYNTAX : cal.
EXPLANATION: This command displays the calendar of the current month on the screen.
OUTPUT : Jun 07

S	M	T	W	T	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

10. TASK : To display user-defined message.
COMMAND : echo.
SYNTAX : echo “message”.
EXPLANATION: This command displays on the screen the argument of the echo command.
OUTPUT : echo “OS”.
 OS

11. TASK : To display the details of all users.
COMMAND : who.

SYNTAX : who.

EXPLANATION : This command lists the information about all the users who have logged on to that system.

OUTPUT :

12. TASK : To display the user detail.

COMMAND : who.

SYNTAX : whoami.

EXPLANATION : This command displays information about the current user of the system on the screen.

OUTPUT : root.

13. TASK : To create a directory.

COMMAND : make directory.

SYNTAX : mkdir.

EXPLANATION : This command is used to create a new directory with the specified name.

EXAMPLE : mkdir student.

OUTPUT : The directory “student” is created.

14. TASK : To change directory.

COMMAND : change directory.

SYNTAX : cd directory name.

EXPLANATION : This command is used to switch from one directory to another.

EXAMPLE : cd staff.

OUTPUT : The directory “staff” is switched onto.

15. TASK : To delete a directory.

COMMAND : remove directory.

SYNTAX : rmdir directory name

EXPLANATION : This command is used to delete the specified directory.

EXAMPLE : rmdir student.

OUTPUT : The “student” directory is deleted.

16. TASK : To come out of a sub-directory.

COMMAND : change directory.

SYNTAX : cd ..

EXPLANATION : This command helps in switching to the main directory.

OUTPUT :

17. TASK : To list all the files and directories.
COMMAND : list.
SYNTAX : ls.
EXPLANATION : This command displays all the files and directories of the system.
OUTPUT :

18. TASK : To create a file.
COMMAND : cat.
SYNTAX : cat> file name.
EXPLANATION : This command leads to the creation of a new file with the specified file name and contents.
EXAMPLE : cat> wind.
OUTPUT : A null file called “wind” is created.
19. TASK : To view a file.
COMMAND : cat.
SYNTAX : cat file name.
EXPLANATION : This command displays the contents of the specified file.
EXAMPLE : cat wind.
OUTPUT : Contents of the file called “wind” will be displayed on the screen.
20. TASK : To copy a file.
COMMAND : copy.
SYNTAX : cp sourcefile destinationfile.
EXPLANATION : This command produces a copy of the source file and is stored in the specified destination file by overwriting its previous contents.
EXAMPLE : cp sun moon.
OUTPUT : The contents of “sun” file will be copied to the “moon” file.

21. TASK : To move a file.
COMMAND : move.
SYNTAX : mv sourcefile destinationfile.
EXPLANATION : After moving the contents of the source file into destination file, the source file is deleted.
EXAMPLE : mv sun moon.
OUTPUT : After copying contents from the “sun” file to “moon” file, the “sun” file is deleted.

22. TASK : To display / cut a column from a file.

COMMAND : cut.

SYNTAX : cut –c no. file name.

EXPLANATION : This command displays the characters of a particular column in the Specified file.

EXAMPLE : cut –c3 moon.

OUTPUT : Those characters occurring in the 3rd column of the file called "moon" are displayed.

23. TASK : To delete a file.

COMMAND : remove.

SYNTAX : rm file name.

EXPLANATION : This command deletes the specified file from the directory.

EXAMPLE : rm sun.

OUTPUT : The file called "sun" will be deleted.

24. TASK : To retrieve a part of a file.

COMMAND : head.

SYNTAX : head -no. of rows file name.

EXPLANATION : This command displays the specified no. of rows form the top of the specified file.

EXAMPLE : head -1 sun.

OUTPUT : The first row of the file called "sun" is displayed.

25. TASK : To retrieve a file.

COMMAND : tail.

SYNTAX : tail -no. of rows file name.

EXPLANATION : This command displays the specified no. of rows form the bottom of the specified file.

EXAMPLE : tail -1 moon.

OUTPUT : The last row of the file called "moon" is displayed.

26. TASK : To sort the contents of a file.

COMMAND : sort.

SYNTAX : sort file name.

EXPLANATION : This command helps in sorting the contents of a file in ascending order.

EXAMPLE : sort win.

OUTPUT : The contents of the file "win" are displayed on the screen in a sorted order.

27. TASK : To display the no. of characters in a file.

COMMAND : word count.

SYNTAX : wc file name.

EXPLANATION : This command displays on the screen the no. of rows, words, and the sum of no. of characters and words.

EXAMPLE : wc ball.

OUTPUT : The no. of rows, words, and no. of characters present in the file "ball" are displayed.

28. TASK : To display the calendar of a year.

COMMAND : cal.

SYNTAX : cal year.

EXPLANATION : This command displays on the screen the calendar of the specified year.

EXAMPLE : cal 2007.

OUTPUT : The calendar of the year 2007 will be displayed.

OUTPUT :

```
Fedora release 8 (Werewolf)
Kernel 2.6.23.1-42.fc8 on an i686
login: student
Password:
Last login: Thu Feb 12 00:59:23 from 83.0.11.54
[student@localhost ~]$ mkdir unix
[student@localhost ~]$ cd unix
 [student@localhost unix]$ cat>file1
welcome to unix
[student@localhost unix]$ cat>file2
basic commands
[student@localhost unix]$ cat file1 file2>file3
[student@localhost unix]$ cat file3
welcome to unix
basic commands
[student@localhost unix]$ cp file3 file4
[student@localhost unix]$ cat file4
welcome to unix

basic commands
[student@localhost unix]$ mv file4 file5
[student@localhost unix]$ cat file5
welcome to unix
[student@localhost unix]$ wc file2
 1  3 16 file2
[student@localhost unix]$ file file2
file2: ASCII text
```

```
[student@localhost unix]$ rm file1 file2 file3 file5
[student@localhost unix]$ cat file3
cat: file3: No such file or directory
[student@localhost unix]$ cd ..
[student@localhost ~]$ rmdir unix
[student@localhost ~]$ file unix /*
unix:        directory
/bin:        directory
/boot:       directory
/dev:        directory
/etc:        directory
/home:       directory
/lib:        directory
/lost+found: directory
/media:      directory
/misc:       directory
/mnt:        directory
/net:        directory
/opt:        directory
/proc:       directory
/root:       directory
/sbin:       directory
/selinux:    directory
/srv:        directory
/sys:        directory
/tmp:        sticky directory
/usr:        directory
/var:        directory
[student@localhost ~]$ cat>file1
welcome to unix
 [student@localhost ~]$ mkdir unix
[student@localhost ~]$ cd unix
[student@localhost unix]$ cat>file3
its my third file
[student@localhost unix]$ cat > file4
it is my 4th file
 [student@localhost unix]$ ls -l
total 16
-rw-rw-r-- 1 student student 18 2009-02-12 01:12 file3
```

-rw-rw-r-- 1 student student 18 2009-02-12 01:13 file4
[student@localhost unix]$ ls file4 -l
-rw-rw-r-- 1 student student 18 2009-02-12 01:13 file4
[student@localhost unix]$ chmod u+x file3
[student@localhost unix]$ chmod g+x file3
[student@localhost unix]$ ls file3 -l
-rwxrwxr-- 1 student student 18 2009-02-12 01:12 file3
[student@localhost unix]$ pwd
/home/student/unix
[student@localhost unix]$ echo UNIX

UNIX
[student@localhost unix]$ ls [a-m]*
file3 file4
[student@localhost unix]$ date
Thu Feb 12 01:22:34 IST 2009
[student@localhost unix]$ date +%m
02
[student@localhost unix]$ date +%a
Thu
[student@localhost unix]$ cal
  February 2009
Su Mo Tu We Th Fr Sa
 1 2 3 4 5 6 7
 8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
[student@localhost unix]$ cal 2009
                              2009

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

       April   May   June
Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa
          1 2 3 4   1 2   1 2 3 4 5 6

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

   October                     November                  December
Su Mo Tu We Th Fr Sa        Su Mo Tu We Th Fr Sa       Su Mo Tu We Th Fr Sa
                1  2  3         1  2  3  4  5  6  7              1  2  3  4  5
 4  5  6  7  8  9 10          8  9 10 11 12 13 14        6  7  8  9 10 11 12
11 12 13 14 15 16 17        15 16 17 18 19 20 21       13 14 15 16 17 18 19
18 19 20 21 22 23 24        22 23 24 25 26 27 28       20 21 22 23 24 25 26
25 26 27 28 29 30 31        29 30                      27 28 29 30 31
[student@localhost unix]$ cal 8 2010
   August 2010
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
[student@localhost unix]$ bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
5*5
25
[student@localhost unix]$ who
student  pts/1        2009-02-11 23:56 (83.0.11.200)
student  pts/3        2009-02-11 23:59 (83.0.11.43)
```

```
student  pts/9        2009-02-12 00:02 (83.0.11.45)
student  pts/8        2009-02-12 00:04 (83.0.11.49)
student  pts/4        2009-02-12 00:06 (83.0.11.46)
student  pts/16       2009-02-12 00:17 (83.0.11.39)
student  pts/18       2009-02-12 00:19 (83.0.11.52)
student  pts/19       2009-02-12 00:20 (83.0.11.50)
student  pts/6        2009-02-12 00:22 (83.0.11.56)
student  pts/22       2009-02-12 00:24 (83.0.11.37)
student  pts/24       2009-02-12 00:25 (83.0.11.41)
student  pts/12       2009-02-12 00:26 (83.0.11.48)
student  pts/13       2009-02-12 00:28 (83.0.11.7)
student  pts/23       2009-02-12 00:38 (83.0.11.8)
student  pts/15       2009-02-12 00:41 (83.0.11.156)
student  pts/25       2009-02-12 00:42 (83.0.11.9)
student  pts/7        2009-02-12 00:48 (83.0.11.38)
student  pts/2        2009-02-12 00:49 (83.11.0.35)
student  pts/14       2009-02-12 00:50 (83.0.11.36)
student  pts/26       2009-02-12 00:51 (83.0.11.59)
student  pts/27       2009-02-12 00:51 (83.0.11.47)
student  pts/5        2009-02-12 00:58 (83.0.11.44)
student  pts/28       2009-02-12 01:01 (83.0.11.200)
student  pts/21       2009-02-12 01:08 (83.0.11.42)
student  pts/10       2009-02-12 01:09 (83.0.11.54)
student  pts/20       2009-02-12 01:11 (83.0.11.55)
student  pts/11       2009-02-12 01:24 (83.0.11.53)

[student@localhost unix]$ who am i
student  pts/28       2009-02-12 01:01 (83.0.11.200)
[student@localhost unix]$ telnet hostname
telnet: hostname: Temporary failure in name resolution
hostname: Host name lookup failure
[student@localhost unix]$ telnet 83.0.11.100
Trying 83.0.11.100...
Connected to 83.0.11.100.
Escape character is '^]'.
Fedora release 8 (Werewolf)
Kernel 2.6.23.1-42.fc8 on an i686
login: student
Password:
```

```
Last login: Thu Feb 12 01:26:10 from 83.0.11.59
[student@localhost ~]$ which cat
/bin/cat
 [student@localhost ~]$ who | wc -l
30
[student@localhost ~]$ who | wc
    30     150    1559
[student@localhost ~]$
[student@localhost ~]$ comm file1 file2
unix
        Unix
[student@localhost ~]$ diff file1 file2
1c1
< unix
\ No newline at end of file
> Unix
\ No newline at end of file
 [student@localhost ~]$ cat >fil1abcdefghijklm
[student@localhost ~]$ head fil1abcdefghij
[student@localhost ~]$ tail fil1efghijklm
[student@localhost ~]$ cat>sp1jkdfgjkfghl;jjob
[student@localhost ~]$ sort sp1fghl;jjkdfgjkjob
[student@localhost ~]$ spell sp1 fghljkdfgjk
[student@localhost ~]$ uniq sp1  jkdfgjkfghl;j job
[student@localhost ~]$ wc sp1  2  3 18 sp1
[student@localhost ~]$ cat>st1
working in unix
basic unix commands
basic shell commands
c programs in unix
[student@localhost ~]$ cat st1 | sort | uniq
basic shell commands
basic unix commands
c programs in unix
working in unix
```

RESULT:

Thus the program was executed successfully and hence output verified.

VI EDITOR COMMANDS

Aim:

To study the VI editor commands.

Theory:

The vi editor is a line-oriented editor and is not very easy to use. But it is simple; you can quickly learn enough commands to use it for editing your java programs.

Command mode and edit mode:

In vi, you will need to shift from command mode to edit mode and back again. You need to be in command mode to move the cursor around on replace in text to another.

Opening and exiting commands

: vi :-	Invokes vi with blank editing screen in command mode
: vi:-	filename:-invokes vi on existing file
:w :-	writes(saves)existing file(command mode only;<ESC>from insert mode
:wFN*	writes(saves)to new file(command mode only;<ESC>from insert mode
:x:-	writes(saves)file and exits vi(command mode only; <ESC>from insert mode
:q :-	Quits vi without saving(command mode only; <ESC>from insert mode
q!:-	Quits vi without saving any changes to any file (command mode only; <ESC>from insert mode

Movement commands

:h :	Move cursor left one character
:j :	Move cursor down one line
:k :	Move cursor up one line
:l :	Move cursor right one character
:w:	Move cursor forward one word
:b :	Move cursor backward one word
:e :	Move cursor to the end of word
^F:	Move cursor forward one screenful
^B:	Move cursor back one screenful
^D:	Move cursor down half screenful
^U:	Move cursor up half screenful

Editing commands

(i) :-	Insert mode; inserts text before current cursor position
A	Insert mode; append text following cursor position
A	Insert mode; appends text at the end of line
O	Open a new line below the current line and insert text
O	Open a new line above the current line and insert text
R	Replace character under cursor
R	Over type mode;<ESC>terminates over type
S	Substitutefollowingtextforcharacteratcursorposition;<ESC>terminates Text entry mode
S	Substitute text on entire line
<ESC>	Return to visual command mode from insert mode.
X	Delete Character at cursor position
X	Delete character before cursor position
Dw	Delete word at cursor position
Dd	Delete current line
Ex	Change text object at cursor position.
P	Put yanked text after or below cursor

P	Put yanked text before or above the cursor
.	Repeat last edit
U	Undo last edit
U	Restore current line

Result:

Thus the VI Editor commands of Unix and Linux have been studied

EX.NO:2.A

CREATING NEW PROCESS USING FORK AND VFORK

DATE:

Aim

To write a program to create a new process using fork and vfork.

Algorithm

Fork()

1. Create child process using fork ().
2. Check the process id returned by fork, if the process id is 0 displays the running process is child.
3. If the process id is non-zero display the running process is parent process.
4. Show that the variables are not shared between parent and child process

Vfork().

1. Create child process using fork ().
2. Check the process id returned by fork, if the process id is 0 displays the running process is child.
3. Check the process id returned by fork, if the process id is non-zero display the running process is parent.
4. Show how the variables are not shared between parent and child process.

Source code:FORK()

```
#include<sys/types.h>
#include<unistd.h>
main()
{
int pid;
int data=200; pid=fork(); if(pid==0)
{
printf("\n\t\t\t\tchild process\n"); data=data+100;
printf("process id=%d\n",getpid());
printf("process id of the parent is=%d\n",getppid()); printf("Value of data is %d\n",data);
}
else
{
printf("\n\t\t\tparent process\n");
printf("process id of the parent=%d\n",getpid()); printf("process id=%d\n",getpid()); printf("Value
of data is %d\n",data);
}
}
```

OUTPUT:

Child process
Process id=4474
Process id of the parent is=4473 Value of data is 300

VFORK():

```
#include<sys/types.h>
 #include<unistd.h>
main()
{
int data=100; int pid; pid=vfork(); if(pid==0)
{
printf("child process\n"); data=data+100;
printf("process id is %d\n",getpid()); printf("procee id of parent is %d\n",getppid()); printf("value of
data is %d",data); printf("\n.  \n");
}
else
{
printf("parent process\n"); printf("process id is %d\n",getpid());
printf("process id of parent is %d\n",getppid()); printf("value of data is %d\n",data);
}
}
```

OUTPUT:

```
child process process id is 5282
procee id of parent is 5281 value of data is 200
parent process process id is 5281
process id of parent is 3928 value of data is 200
```

Result

Thus creating new process using fork and vfork has been verified.

EX.NO:2.B

HANDLING WAIT, EXIT AND SLEEP WITHIN PROCESS

DATE:

Aim

To write a program using wait, sleep and exit system calls.

Algorithm

1. Create child process using fork() system call.
2. Check the process id,if the process id is 0 then display that the running process is child.
3. Delay the child process completion by using sleep() system call.
4. Make the parent process do wait for the child process completion by using wait system call.
5. Use exit() system call in child process after sleep time gets expired.

Source code:

```
#include<sys/types.h> #include<sys/wait.h> main(void)
{
int pid,status,k,i,j; if((pid=fork())<0) printf("fork error"); else if(pid==0)
{
printf("CHILD PROCESS ID=%d\n",getpid()); sleep(20);
exit(7);
}
else
{
printf("PARENT PROCESS\n"); if((k=wait(&status))!=pid) printf("Error");
else if(WIFEXITED(status))
{
printf("NORMAL TERMINATION,EXIT STATUS=%d\n",WEXITSTATUS(status));
}
else
printf("OTHER THAN NORMAL TERMINATION");
exit(0);
}
}
```

OUTPUT:

CHILD PROCESS ID=6144

PARENT PROCESS
NORMAL TERMINATION,EXIT STATUS=7

Result:

Thus the program for handling wait, exit and sleep within process has been executed.

EX.NO:3A

IMPLEMENTATION OF LS SYSTEM CALL

DATE:

Aim:

To implement the unix command _ls' which displays the files in the directory.

Algorithm:

1. Include the header file dirent.h.
2. DIR is the internal structure to maintain information about the directory being read.
3. DECLARE the structures dirent to get the files in the directory.
4. Open the directory given at the command line.
5. Set a while loop to read the files under the directory.
6. Display the names of the files that reside in the given directory.
7. Lose the directory that was opened.

Source code:

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

OUTPUT:

[44208104008@localhost oslab]$ cc ls.c [44208104008@localhost oslab]$ ./a.out

.

..

.contiguous.c.swo

.contiguous.c.swp

.linked.c.swp

.worstfit.c.swp INPUT

a.out bestfit.c contiguous.c fifopage.c firstfit.c fork.c getpid.c iosystem.c ipc.c linked.c lrupage.c ls.c opendir.c pa.c

prio.c

RESULT:

Thus implementation of the UNIX commands _ls ‘which displays the files in the directory is executed and the output is verified.

EX.NO:3.b

IMPLEMENTATION OF GREP SYSTEM CALL

DATE:

Aim:

To implement the unix command _grep' which displays the files in the directory.

Algorithm:

1. Include the header file dirent.h.
2. DIR is the internal structure to maintain information about the directory being read.
3. DECLARE the structures dirent to get the files in the directory.
4. Open the directory given at the command line.
5. Set a while loop to read the files under the directory.
6. Display the names of the files that reside in the given directory.
7. Lose the directory that was opened.

Source code:

```
#include<stdio.h> void main()
{
int i=0,j=0; char a[5],b[1];
printf("Enter the string : "); scanf("%s",a);
printf("Enter the char to be searched : "); scanf("%s",b);
while(j<5)
{
if(a[j]==b[0]) i++;
j++;
}
if(i!=0)
printf("The char is found"); else
printf("\n Not found");
}
```

OUTPUT:[44208104008@localhost cc grep.c [44208104008@localhost oslab]$
./a.out
Enter the string : Bala Enter the char to be searched : l The char is found

RESULT:

Thus implementation of the UNIX commands _grep ‘which displays the files in the directory is executed and the output is verified.

EXNO::4 **SHELL PROGRAMMING**
DATE:

A Linux shell is a command language interpreter, the primary purpose of which is to translate the command lines typed at the terminal into system actions. The shell itself is a program, through which other programs are invoked

What is a shell script ?

- A shell script is a file containing a list of commands to be executed by the Linux shell. shell script provides the ability to create your own customized Linux commands
- Linux shell have sophisticated programming capabilities which makes shell script powerful Linux tools

How to work with shell ?

Step1:

In the dollar prompt type

$ vi < file name>

Where vi is the editor ,it will open a new window in which you can type the program you want

Step2:

After typing the program press ESC and : together then at the bottom of the vi screen you can see i.e. prompt .In that type as wq which means write and quit i.e. the content what is typed will be written and saved into that file that has been created

Step3:

Once wq is typed at the : prompt ,the prompt would change to $ symbol in which you have to do the following

$ sh < file name >

Sh – command is used to run the shell program

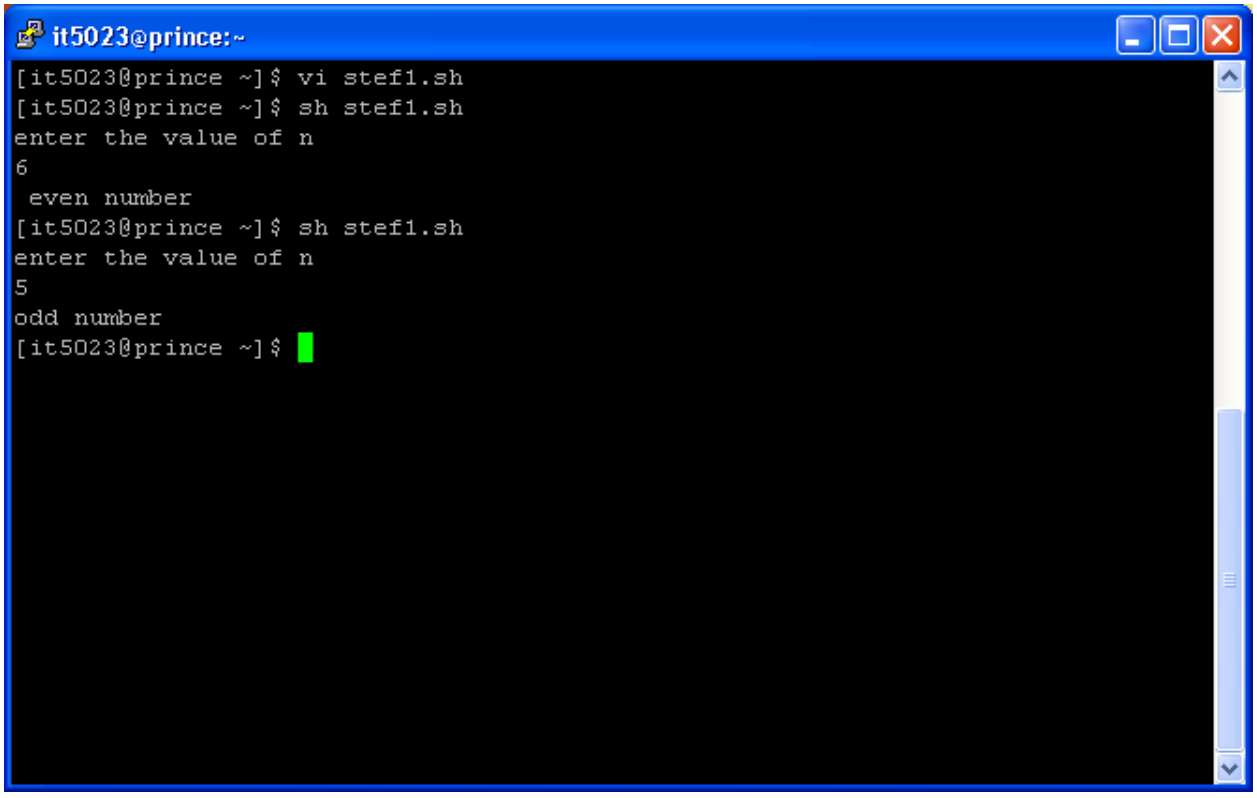
<file name> - is the name of the file for which the output is to be got Basically to print a text in the your shell programs echo command is used

FINDING ODD OR EVEN NUMBER

PROGRAM:

```
echo "enter the value of n:"
read n
r=`expr $n % 2`
if test $r -eq 0
then
echo "even number"
else
echo "odd number"
fi
```

OUTPUT:

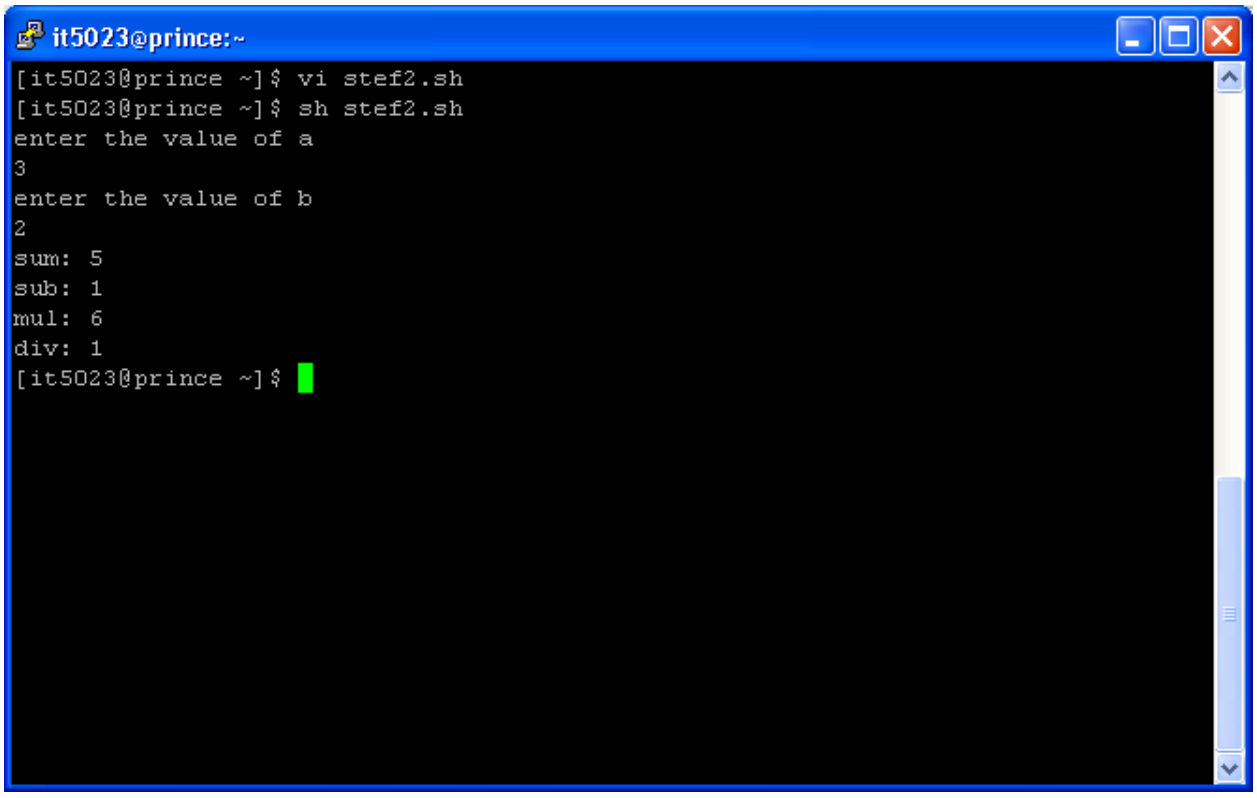


ARITHEMETIC OPERATION USING SWITCH CASE

PROGRAM:

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

OUTPUT:

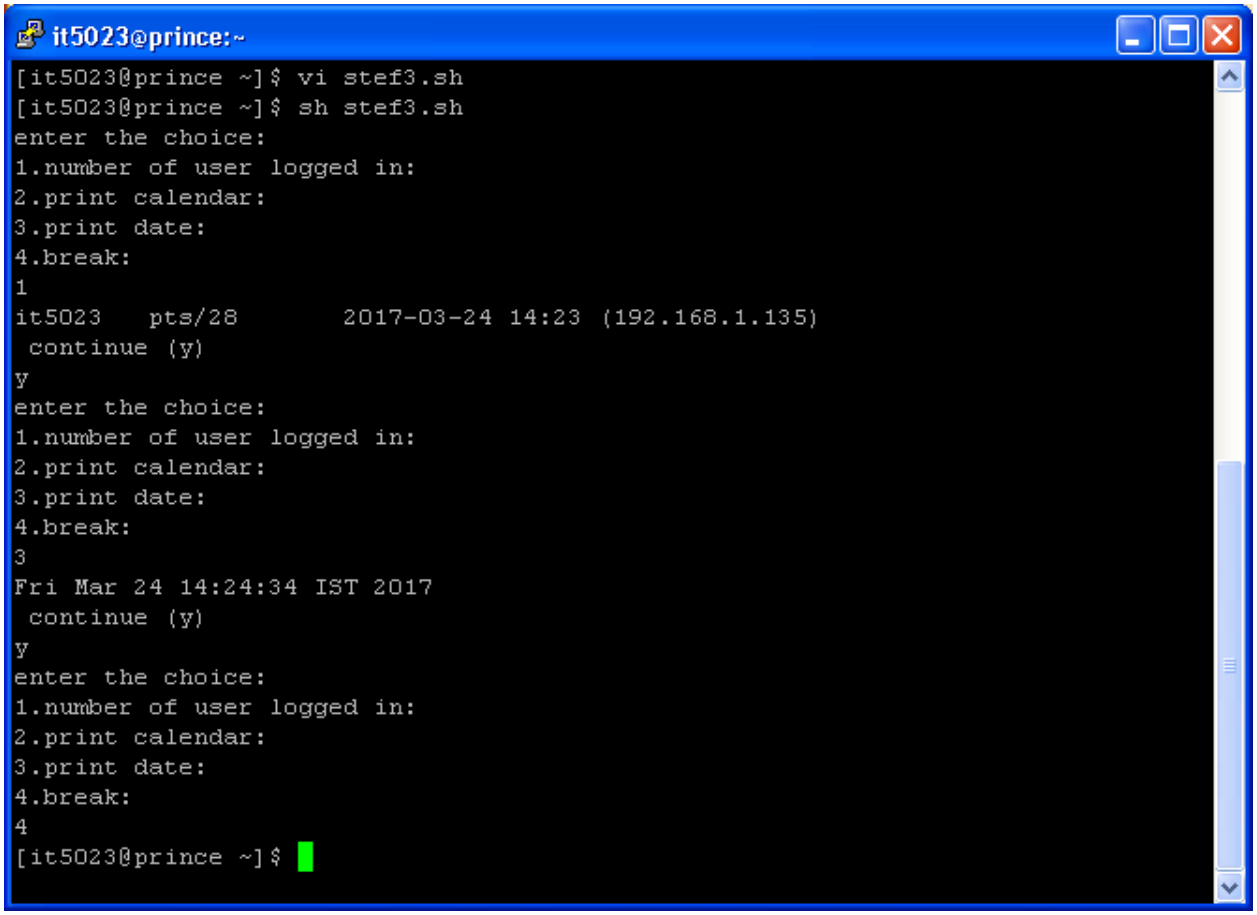


EXECUTING SHELL COMMANDS USING SWITCH CASE

PROGRAM:

```
while test $ch='y'
do
echo enter the choice:
echo 1.number of user logged in:
echo 2.print calendar:
echo 3.print date:
echo 4.break:
read d
case $d in
1) who i am;;
2) cal 20;;
3) date;;
4) break;;
esac
echo " continue (y)"
read ch
done
```

OUTPUT:

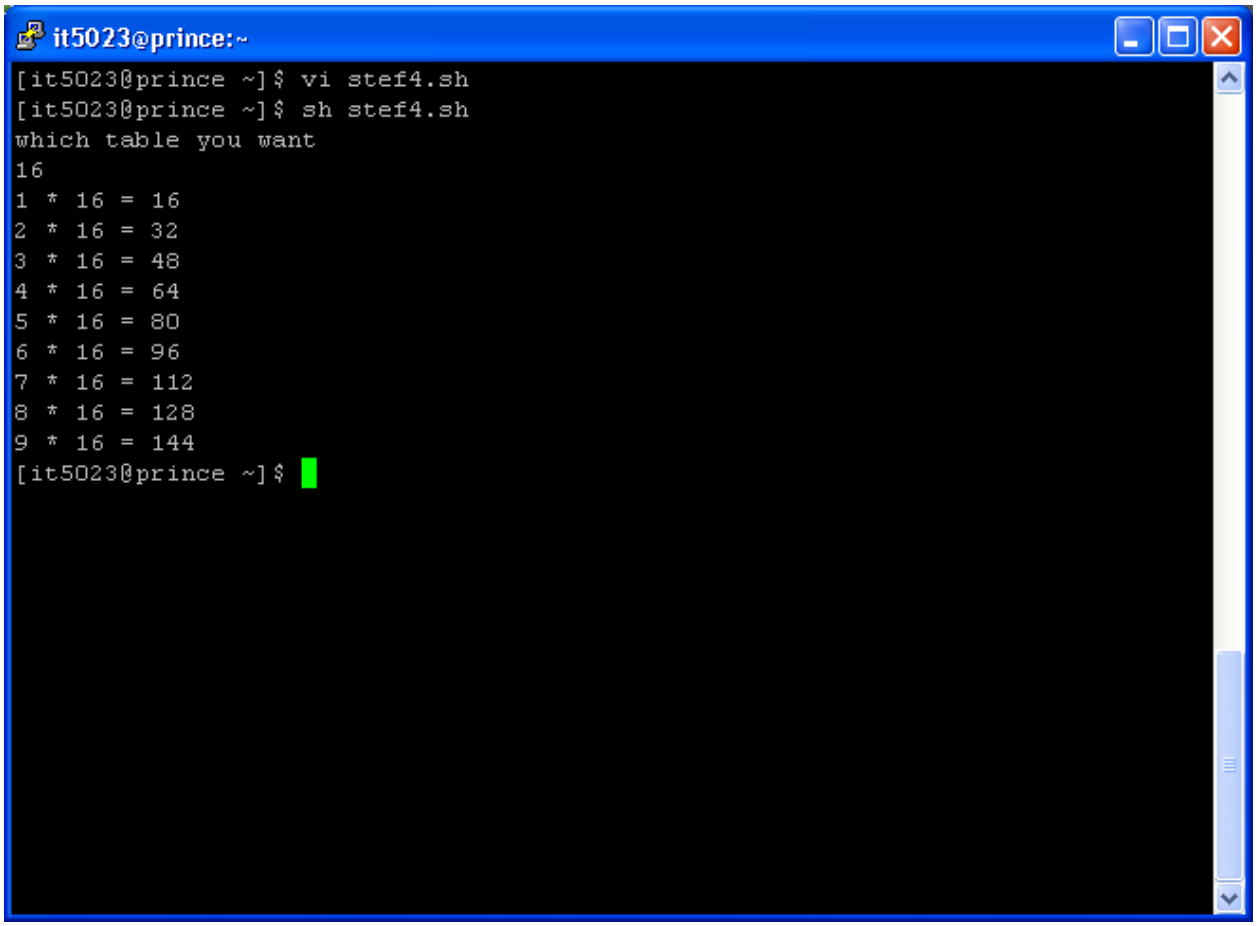

it5023@prince:~
[it5023@prince ~]$ vi stef3.sh
[it5023@prince ~]$ sh stef3.sh
enter the choice:
1.number of user logged in:
2.print calendar:
3.print date:
4.break:
1
it5023 pts/28 2017-03-24 14:23 (192.168.1.135)
continue (y)
y
enter the choice:
1.number of user logged in:
2.print calendar:
3.print date:
4.break:
3
Fri Mar 24 14:24:34 IST 2017
continue (y)
y
enter the choice:
1.number of user logged in:
2.print calendar:
3.print date:
4.break:
4
[it5023@prince ~]$

GENERATING MULTIPLICATION TABLE

PROGRAM:

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

OUTPUT:



PALINDROME

PROGRAM:

```
echo "enter the string"

read s

n=`expr $s | wc -c`

a=''

while test $n -gt 0

do
x=`expr $s | cut -c $n`
a=`echo $a$x`
n=`expr $n-1 | bc`
done
echo "the reversed string is" $a
if test "$s" = "$a"
then
echo "the given string is palindrome"
else
echo "the given string is not palindrome"
fi
```

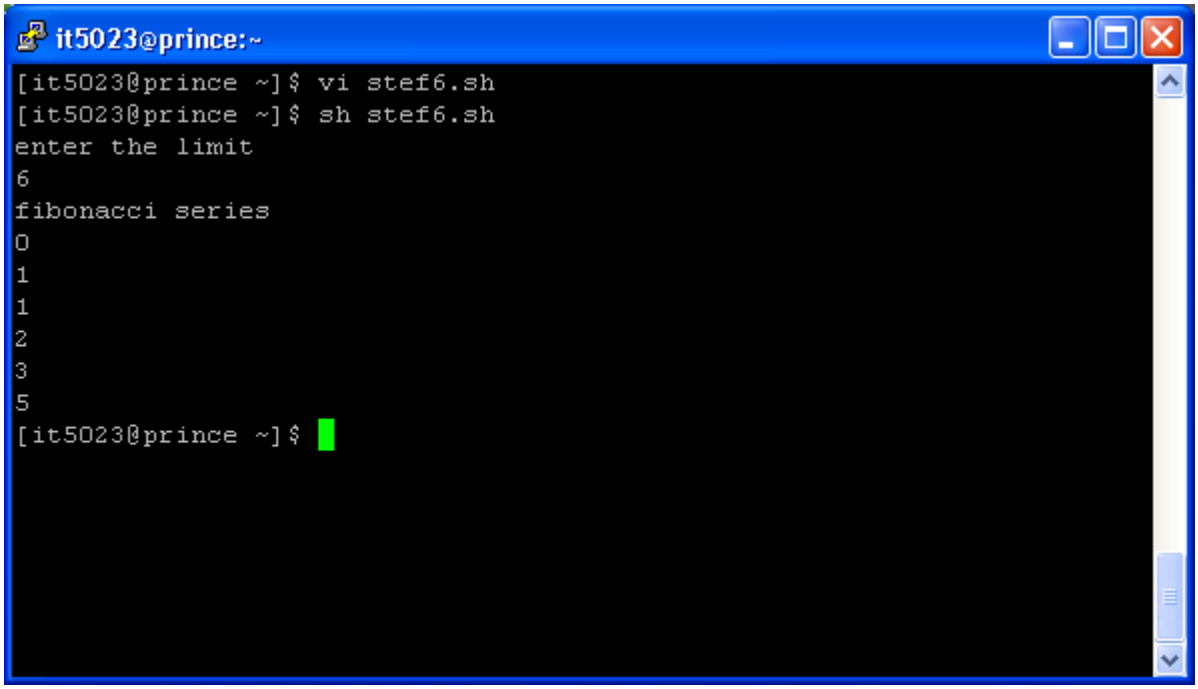
OUTPUT:

FIBONACCI SERIES

PROGRAM:

```
echo "enter the limit"
read i
n=2
x=0
y=1
echo "fibonacci series"
echo $x
echo $y
while test $n -lt $i
do
z=`expr $x+$y|bc`
echo $z
x=$y
y=$z
n=`expr $n+1|bc`
done
```

OUTPUT:


it5023@prince:~
[it5023@prince ~]$ vi stef6.sh
[it5023@prince ~]$ sh stef6.sh
enter the limit
6
fibonacci series
0
1
1
2
3
5
[it5023@prince ~]$

CHECK FOR PRIME NUMBER

PROGRAM:

```
echo "enter the value of n"
read n
flag=0
for ((i=2;i<n/2;i++))
do
r=`expr $n%$i|bc`
if test $r -eq 0
then flag=1
break
fi
done
if test $flag -eq 0
then
echo "$n is a prime number"
else
echo "$n is not a prime number"
fi
```

OUTPUT:

Result:

The program has been executed successfully and output verified.

EXNO: 5

PROGRAMS TO IMPLEMENT THE VARIOUS CPU SCHEDULING DATE: ALGORITHMS

OBJECTIVE

Write a C program to simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time for the above problem.
FCFS b) SJF c) Round Robin d) Priority

DESCRIPTION

Assume all the processes arrive at the same time.

FCFS CPU SCHEDULING ALGORITHM

For FCFS scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. The scheduling is performed on the basis of arrival time of the processes irrespective of their other parameters. Each process will be executed according to its arrival time. Calculate the waiting time and turnaround time of each of the processes accordingly.

SJF CPU SCHEDULING ALGORITHM

For SJF scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. Arrange all the jobs in order with respect to their burst times. There may be two jobs in queue with the same execution time, and then FCFS approach is to be performed. Each process will be executed according to the length of its burst time. Then calculate the waiting time and turnaround time of each of the processes accordingly.

ROUND ROBIN CPU SCHEDULING ALGORITHM

For round robin scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the size of the time slice. Time slices are assigned to each process in equal portions and in circular order, handling all processes execution. This allows every process to get an equal chance. Calculate the waiting time and turnaround time of each of the processes accordingly.

PRIORITY CPU SCHEDULING ALGORITHM

For priority scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the priorities. Arrange all the jobs in order with respect to their priorities. There may be two jobs in queue with the same priority, and then FCFS approach is to be performed. Each process will be executed according to its priority. Calculate the waiting time and turnaround time of each of the processes accordingly.

EXNO:5a

CPU SCHEDULING – FIRST COME FIRST SERVE

DATE:

AIM:

To write a C program to implement the FCFS process scheduling mechanisms.

ALGORITHM:

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Set the waiting of the first process as '0' and its burst time as its turn around time
Step 5: for each process in the Ready Q calculate
  (a) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
  (b) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)
Step 6: Calculate
  (a) Average waiting time = Total waiting Time / Number of process
  (b) Average Turnaround time = Total Turnaround Time / Number of process
Step 7: Stop the process

PROGRAM:

```
#include<stdio.h>

 int main()

{
   int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
   printf("Enter total number of processes(maximum 20):");
   scanf("%d",&n);

   printf("nEnter Process Burst Timen");
   for(i=0;i<n;i++)
   {
      printf("P[%d]:",i+1);
      scanf("%d",&bt[i]);
   }

   wt[0]=0;

   for(i=1;i<n;i++)
   {
      wt[i]=0;
      for(j=0;j<i;j++)
         wt[i]+=bt[j];
   }

   printf("nProcessttBurst TimetWaiting TimetTurnaround Time");
   for(i=0;i<n;i++)
   {
      tat[i]=bt[i]+wt[i];
```

```
        avwt+=wt[i];
        avtat+=tat[i];
        printf("nP[%d]tt%dtt%dtt%d",i+1,bt[i],wt[i],tat[i]);
    }
    avwt/=i;
    avtat/=i;
    printf("nnAverage Waiting Time:%d",avwt);
    printf("nAverage Turnaround Time:%d",avtat);

    return 0;
}
```

OUTPUT:

```
[csestudent@fedora vajitha]$ ./a.out

Enter the number of processs--5

Enter Burst Time for process 0--20

Enter Burst Time for process 1--3

Enter Burst Time for process 2--6

Enter Burst Time for process 3--4

Enter Burst Time for process 4--5
         PROCESS        BURST TIME       WAITING TIME   TURNAROUND TIME

        P0               20              0               20
        P1               3               20              23
        P2               6               23              29
        P3               4               29              33
        P4               5               33              38
Average Waiting Time--21.000000
```

Result:

The program has been executed successfully and output verified

Ex.No:5b

CPU SCHEDULING –SHORTEST JOB FIRST

DATE:

AIM:

To write a C program to implement the SJF process scheduling mechanisms.

ALGORITHM

Step 1: Start the process

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

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

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

Step 5: Set the waiting time of the first process as '0' and its turnaround time as its burst time.

Step 6: For each process in the ready queue, calculate

(c) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)

(d) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate

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

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

Step 7: Stop the process

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
main()
{
 int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
 float wtavg, tatavg;
 clrscr();
 printf("\nEnter the number of processes --  ");
 scanf("%d", &n);
 for(i=0;i<n;i++)
 {
  p[i]=i;
  printf("Enter Burst Time for Process %d --  ", i);
  scanf("%d", &bt[i]);

 }
 for(i=0;i<n;i++)
  for(k=i+1;k<n;k++)
   if(bt[i]>bt[k])
   {
    temp=bt[i];
    bt[i]=bt[k];



    bt[k]=temp;
    temp=p[i];
    p[i]=p[k];
    p[k]=temp;
    }
```

```
 wt[0] = wtavg = 0;
 tat[0] = tatavg = bt[0];
 for(i=1;i<n;i++)
 {
  wt[i] = wt[i-1] +bt[i-1];
  tat[i] = tat[i-1] +bt[i];
  wtavg = wtavg + wt[i];
  tatavg = tatavg + tat[i];
 }
 printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
 for(i=0;i<n;i++)
  printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);
 printf("\nAverage Waiting Time -- %f", wtavg/n);
 printf("\nAverage Turnaround Time -- %f", tatavg/n);
 getch();
}
```

INPUT

```
Enter the number of processes --   4
Enter Burst Time for Process  0 --   6
Enter Burst Time for Process  1 --   8
Enter Burst Time for Process  2 --   7
Enter Burst Time for Process  3 --   3
```

OUTPUT

```
PROCESS        BURST TIME    WAITING TIME  TURNAROUND TIME
P3   3   0  3
P0   6   3  9
P2   7   9  16
P1   8   16  24
Average Waiting Time --  7.000000
Average Turnaround Time -- 13.000000
```

Result:

The program has been executed successfully and output verified

Ex.No:5c

CPU SCHEDULING – ROUND ROBIN

DATE:

AIM:

To write a C program to implement the Round Robin process scheduling mechanisms.

ALGORITHM

Step 1: Start the process

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

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

Step 4: Calculate the no. of time slices for each process where

No. of time slice for process(n) = burst time process(n)/time slice

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

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

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

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

Step 7: Calculate

(e) Average waiting time = Total waiting Time / Number of process

(f) Average Turnaround time = Total Turnaround Time / Number of process

Step 8: Stop the process

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
struct process
{
int pn,bt,wt,tat;
}p[10];
void main()
{
int tq,i,j=0,n,b[10],ttat=0;
float awt=0,atat=0;
clrscr();
printf("\n enter no. of process:");
scanf("%d",&n);
printf("\n enter input:\np.no\tburs tim\n");
for(i=1;i<=n;i++)
{
scanf("%d %d",&p[i].pn,&p[i].bt);
b[i]=p[i].bt;
ttat=ttat+p[i].bt;
}
printf("\n enter time quantum:");
scanf("%d",&tq);
while(j<ttat)
{
  for(i=1;i<=n;i++)
```

```
  {
   if(p[i].bt!=0)
   {
     if(p[i].bt>tq)
     {
           p[i].bt=p[i].bt-tq;
           j=j+tq;
     }
     else
     {

  j=j+p[i].bt;
     p[i].tat=j;
     p[i].wt=p[i].tat-b[i];
    atat=atat+p[i].tat;
    awt=awt+p[i].wt;
    p[i].bt=0;
      } } }}
printf("\np.no\tburst\twait\tturn\n:");
for(i=1;i<=n;i++)
{
printf("\n%d\t%d\t%d\t%d",p[i].pn,b[i],p[i].wt,p[i].tat);
printf("\n");
}
printf("\n avg wait time is %f",awt/n);
printf("\n avg turn time is %f",atat/n);
getch();
}
```

OUTPUT:

```
enter no. of process:3
 enter input:
p.no    burs tim
1       24
2       3
3       3
 enter time quantum:4
p.no    burst   wait    turn:
1       24      6       30
2       3       4       7
3       3       7       10
 avg wait time is 5.666667
 avg turn time is 15.666667
```

Result:

The program has been executed successfully and output verified

Ex.No:5d

CPU SCHEDULING – PRIORITY

DATE:

AIM:

To write a C program to implement the priority process scheduling mechanisms.

ALGORITHM:

1. Start the process
2. Get the number of processes to be inserted
3. Get the corresponding priority of processes
4. Sort the processes according to the priority and allocate the one with highest priority to execute first
5. If two process have same priority then FCFS scheduling algorithm is used
6. Calculate the total and average waiting time and turnaround time
7. Display the values
8. Stop the process

PROGRAM:

```
#include<stdio.h>
main()
{
 int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp;
 float wtavg, tatavg;
 clrscr();
 printf("Enter the number of processes  ---  ");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
  p[i] = i;
  printf("Enter the Burst Time & Priority of Process %d ---  ",i);
  scanf("%d %d",&bt[i], &pri[i]);
 }
 for(i=0;i<n;i++)
  for(k=i+1;k<n;k++)
   if(pri[i] > pri[k])
   {
    temp=p[i];
    p[i]=p[k];
    p[k]=temp;
    temp=bt[i];
    bt[i]=bt[k];
    bt[k]=temp;
    temp=pri[i];

    pri[i]=pri[k];
    pri[k]=temp;

    }
```

```
 wtavg = wt[0] = 0;
 tatavg = tat[0] = bt[0];
for(i=1;i<n;i++)
{
 wt[i] = wt[i-1] + bt[i-1];
 tat[i] = tat[i-1] + bt[i];
 wtavg = wtavg + wt[i];
 tatavg = tatavg + tat[i];
}
printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME");
for(i=0;i<n;i++)
 printf("\n%d \t\t %d \t\t  %d \t\t %d \t\t %d ",p[i],pri[i],bt[i],wt[i],tat[i]);
printf("\nAverage Waiting Time is  ---  %f",wtavg/n);
printf("\nAverage Turnaround Time is --- %f",tatavg/n);
getch();
}
```

INPUT

```
Enter the number of processes --   5
Enter the Burst Time & Priority of Process 0 ---  10 3
Enter the Burst Time & Priority of Process 1 ---  1 1
Enter the Burst Time & Priority of Process 2 ---  2 4
Enter the Burst Time & Priority of Process 3 ---  1 5
Enter the Burst Time & Priority of Process 4 ---  5 2
```

OUTPUT

```
PROCESS PRIORITY BURST TIME  WAITING TIME TURNAROUND TIME
1            1            1            0            1
4            2            5            1            6
0            3            10           6            16
2            4            2            16           18
3            5            1            18           19
Average Waiting Time is  ---  8.200000
Average Turnaround Time is --- 12.000000
```

RESULT :

Thus the program was executed successfully and hence output verified

Ex.No :6

IMPLEMENT THE PRODUCER – CONSUMER PROBLEM USING SEMAPHORES (USING UNIX SYSTEM CALLS).

DATE:

AIM:

To write a program for Implement the Producer – Consumer problem using semaphores (using UNIX system calls).

ALGORITHM:

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

Step 2: In the case of producer process

i)Produce an item in to temporary variable.

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

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

Step 3: In the case of consumer process

i) It should wait if the buffer is empty

ii) If there is any item in the buffer check for mutex value,

iii) if the mutex==0, remove item from buffer

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

v) Consume the item.

Step 4: Print the result

PROGRAM :

```
#include<stdio.h>
void main()
{
 int buffer[10], bufsize, in, out, produce, consume, choice=0;
 in = 0;
 out = 0;
 bufsize = 10;
 while(choice !=3)
 {
  printf("\n1. Produce  \t 2. Consume \t3. Exit");
  printf("\nEnter your choice:  ");
  scanf("%d", &choice);
  switch(choice) {
   case 1:  if((in+1)%bufsize==out)
     printf("\nBuffer is Full");
    else
    {
     printf("\nEnter the value: ");
     scanf("%d", &produce);
     buffer[in] = produce;
     in = (in+1)%bufsize;
```

```
  }
    Break;
   case 2: if(in == out)
     printf("\nBuffer is Empty");
    else
    {
     consume = buffer[out];
     printf("\nThe consumed value is %d", consume);
     out = (out+1)%bufsize;
    }
    break;
}
 }
}
```

OUTPUT

```
1. Produce 2. Consume 3. Exit
Enter your choice: 2
Buffer is Empty
1. Produce 2. Consume 3. Exit
Enter your choice: 1
Enter the value: 100
1. Produce 2. Consume 3. Exit
Enter your choice: 2
The consumed value is 100
1. Produce 2. Consume 3. Exit
Enter your choice: 3
```

RESULT:

Thus the program for Implement the Producer – Consumer problem using semaphores (using UNIX system calls) was written and successfully executed.

EX NO: 7

DEVELOPING APPLICATION USING INTER PROCESS COMMUNICATION (USING SHARED MEMORY, PIPES OR MESSAGE QUEUES)

DATE:

AIM:

To write a program for developing Application using Inter Process communication with pipes.

ALGORITHM:

1. Start the program.
2. Read the input from parent process and perform in child process.
3. Write the date in parent process and read it in child process.
4. Fibonacci Series was performed in child process.
5. Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/ipc.h>
#include<sys/uio.h>
#include<sys/types.h>
main()
{
        int pid,pfd[2],n,a,b,c;
        if(pipe(pfd)==-1)
       {
                printf("\nError in pipe connection\n");
                exit(1);
       }
       pid=fork()
       ; if(pid>0)
      {
                printf("\nParent Process");\
                printf("\n\n\tFibonacci Series");
                printf("\nEnter the limit for the series:");
                scanf("%d",&n);
                close(pfd[0]);
                write(pfd[1],&n,sizeof(n));
                close(pfd[1]);
                exit(0);
       }
       else
     {
            close(pfd[1]);
            read(pfd[0],&n,sizeof(n));
            printf("\nChild Process");
            a=0;
             b=1;
```

```
        close(pfd[0]); printf("\nFibonacci Series is:");
        printf("\n\n%d\n%d",a,b);
        while(n>2)
      {
            c=a+b;
            printf("\n%d",c);
            a=b;
            b=
            c;
            n--;
      }
}
}
```

OUTPUT:

```
[root@localhost ~]# ./a.out
Parent Process
Fibonacci Series
Enter the limit for the series:5
Child Process Fibonacci Series is: 01123
```

RESULT:

Thus the program for Implementation of shared memory and IPC was written and successfully executed.

EX NO: 8

BANKERS ALGORITHM FOR DEAD LOCK AVOIDANCE

DATE:

AIM:

Implement the bankers algorithm for dead lock avoidance.

ALGORITHM:

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

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n,r,i,j,k,p,u=0,s=0,m;
int block[10],run[10],active[10],newreq[10];
int max[10][10],resalloc[10][10],resreq[10][10];
int totalloc[10],totext[10],simalloc[10];
//clrscr();
printf("Enter the no of processes:");
scanf("%d",&n);
printf("Enter the no ofresource classes:");
scanf("%d",&r);
printf("Enter the total existed resource in each class:");
for(k=1; k<=r; k++)
scanf("%d",&totext[k]);
printf("Enter the allocated resources:");
for(i=1; i<=n; i++)
for(k=1; k<=r; k++)
scanf("%d",&resalloc);
printf("Enter the process making the new request:");
scanf("%d",&p);
printf("Enter the requested resource:");
for(k=1; k<=r; k++)
scanf("%d",&newreq[k]);
```

```
printf("Enter the process which are n blocked or running:");
for(i=1; i<=n; i++)
{
if(i!=p)
{
printf("process %d:\n",i+1);
scanf("%d%d",&block[i],&run[i]);
}
}
block[p]=0;
run[p]=0;
for(k=1; k<=r; k++)
{

j=0;
for(i=1; i<=n; i++)
{
totalloc[k]=j+resalloc[i][k];
j=totalloc[k];
}
}
for(i=1; i<=n; i++)
{
if(block[i]==1||run[i]==1)
active[i]=1;
else
active[i]=0;
}
for(k=1; k<=r; k++)
{
resalloc[p][k]+=newreq[k];
totalloc[k]+=newreq[k];
}
for(k=1; k<=r; k++)
{
if(totext[k]-totalloc[k]<0)
{
u=1;
break;
}
}
if(u==0)
{
for(k=1; k<=r; k++)
simalloc[k]=totalloc[k];
```

```
for(s=1; s<=n; s++)
for(i=1; i<=n; i++)
{
if(active[i]==1)
{
j=0;
for(k=1; k<=r; k++)
{
if((totext[k]-simalloc[k])<(max[i][k]-resalloc[i][k]))
{
j=1;
break;
}
}
}
if(j==0)

{
active[i]=0;
for(k=1; k<=r; k++)
simalloc[k]=resalloc[i][k];
}
}
m=0;
for(k=1; k<=r; k++)
resreq[p][k]=newreq[k];
printf("Deadlock willn't occur");
}
else
{
for(k=1; k<=r; k++)
{
resalloc[p][k]=newreq[k];
totalloc[k]=newreq[k];
}
printf("Deadlock will occur");
}
getch();
}
```

OUTPUT:

```
Enter the no of processes:4
Enter the no ofresource classes:3
Enter the total existed resource in each class:3 2 2
Enter the allocated resources:1 0 0 5 1 1 2 1 1 0 0 2
Enter the process making the new request:2
Enter the requested resource:1 1 2
Enter the process which are n blocked or running:process 2:
1 2
process 4:
1 0
process 5:
1 0
Deadlock will occur
```

RESULT:

Thus the program for Implementation of Bankers algorithm was written and successfully executed

Ex.No:9

IMPLEMENTATION OF DEADLOCK DETECTION ALGORITHM

DATE:

AIM:

To Implement the algorithm for dead lock detection.

ALGORITHM:

1.Mark each process that has a row in the Allocation matrix of all zeros.
2. Initialize a temporary vectorW to equal the Available vector.
3. Find an index*i* such that process*i* is currently unmarked and the*i* th row ofQ is less than or equal to W . That is,*Q ik … Wk*, for 1 … *k* … *m* . If no such row is found, terminate the algorithm.
4. If such a row is found, mark process*i* and add the corresponding row of the allocation matrix to W . That is, set*Wk = Wk* + A*ik*, for 1 … *k* … *m* . Return to step 3.

PROGRAM:

```
#include<stdio.h>
static int mark[20];
int i,j,np,nr;

int main()
{
int alloc[10][10],request[10][10],avail[10],r[10],w[10];

printf("\nEnter the no of process: ");
scanf("%d",&np);
printf("\nEnter the no of resources: ");
scanf("%d",&nr);
for(i=0;i<nr;i++)
{
printf("\nTotal Amount of the Resource R%d: ",i+1);
scanf("%d",&r[i]);
}
printf("\nEnter the request matrix:");

for(i=0;i<np;i++)
for(j=0;j<nr;j++)
scanf("%d",&request[i][j]);

printf("\nEnter the allocation matrix:");
```

```
for(i=0;i<np;i++)
for(j=0;j<nr;j++)
scanf("%d",&alloc[i][j]);
/*Available Resource calculation*/
for(j=0;j<nr;j++)
{
avail[j]=r[j];
for(i=0;i<np;i++)
{
avail[j]-=alloc[i][j];

}
}

//marking processes with zero allocation

for(i=0;i<np;i++)
{
int count=0;
 for(j=0;j<nr;j++)
   {
     if(alloc[i][j]==0)
       count++;
     else
       break;
   }
 if(count==nr)
 mark[i]=1;
}
// initialize W with avail

for(j=0;j<nr;j++)
    w[j]=avail[j];

//mark processes with request less than or equal to W
for(i=0;i<np;i++)
{
int canbeprocessed=0;
 if(mark[i]!=1)
{
   for(j=0;j<nr;j++)
    {
     if(request[i][j]<=w[j])
       canbeprocessed=1;
     else
```

```
        {
        canbeprocessed=0;
        break;
         }
    }
if(canbeprocessed)
{
mark[i]=1;

for(j=0;j<nr;j++)
w[j]+=alloc[i][j];
}
}
}

//checking for unmarked processes
int deadlock=0;
for(i=0;i<np;i++)
if(mark[i]!=1)
deadlock=1;


if(deadlock)
printf("\n Deadlock detected");
else
printf("\n No Deadlock possible");
}
```

OUTPUT:

```
Enter the no of process: 4
Enter the no of resources: 5

Total Amount of the Resource R1: 2
Total Amount of the Resource R2: 1
Total Amount of the Resource R3: 1
Total Amount of the Resource R4: 2
Total Amount of the Resource R5: 1

Enter the request matrix:0 1 0 0 1
0 0 1 0 1
0 0 0 0 1
1 0 1 0 1

Enter the allocation matrix:1 0 1 1 0
1 1 0 0 0
0 0 0 1 0
0 0 0 0 0

 Deadlock detected
--------------------------------
```

RESULT:

Thus the program for Implementation of Bankers algorithm was written and successfully executed

Ex.No:10

IMPLEMENT THREADING & SYNCHRONIZATION APPLICATIONS

DATE:

AIM:

To implement the threading and synchronization using linux.

ALGORITHM:

Step 1: Start the Program

Step 2: Initialize the process thread array.

Step 3: Print the job started status.

Step 4: Print the job finished status.

Step 5: Start the main function

Step 6: Check for the process creation if not print error message.

Step 7: Stop the execution

PROGRAM:

```
#include<stdio.h>
#include <string.h>
#include<pthread.h>
#include <stdlib.h>
#include <unistd.h>
pthread_t tid[2];
 int counter;
 void* doSomeThing(void *arg)
{
unsigned long i = 0;
 counter += 1;
 printf("\n Job %d started\n", counter);
 for(i=0; i< (0xFFFFFFFF);i++);
printf("\n Job %d finished\n", counter);
return NULL;
}
 int main(void)
{
 int i = 0;
int err;
```

```
while(i < 2)
 {
 err = pthread_create(&(tid[i]), NULL, &doSomeThing, NULL);
if (err != 0)
 printf ("\ncan't create thread :[%s]", strerror(err));
 i++;
 }
 pthread_join(tid[0], NULL);
pthread_join(tid[1],  NULL);
return 0;
 }
```

OUTPUT:

Job 1 started

Job 1finished

can't create thread :

RESULT : Thus the program was executed successfully and hence output verified

Ex.No:11

IMPLEMENTATION OF THE FOLLOWING MEMORY ALLOCATION METHODS FOR FIXED PARTITION

A) FIRST FIT B) WORST FIT C) BEST FIT

DATE:

AIM:

To implement the Following Memory Allocation Methods For Fixed Partition First Fit Worst Fit , Best Fit

MEMORY MANAGEMENT

First fit

The first-fit, best-fit, or worst-fit strategy is used to select a free hole from the set of available holes. Allocate the first hole that is big enough. Searching starts from the beginning of set of holes.

Best fit

Allocate the smallest hole that is big enough.

The list of free holes is kept sorted according to size in ascending order. This strategy produces smallest leftover holes

Worst fit

Allocate the largest hole. The list of free holes is kept sorted according to size in descending order. This strategy produces the largest leftover hole.

BEST FIT ALGORITHM

1- Input memory blocks and processes with sizes.
2- Initialize all memory blocks as free.
3- Start by picking each process and find the
minimum block size that can be assigned to
current process i.e., find min(bockSize[1],
blockSize[2],.....blockSize[n]) >
processSize[current], if found then assign
it to the current process.
5- If not then leave that process and keep checking
the further processes.

WORST FIT ALGORITHM

1- Input memory blocks and processes with sizes.
2- Initialize all memory blocks as free.
3- Start by picking each process and find the

maximum block size that can be assigned to
current process i.e., find max(bockSize[1],
blockSize[2],.....blockSize[n]) >

processSize[current], if found then assign
it to the current process.
5- If not then leave that process and keep checking
the further processes.

FIRST FIT ALGORITHM

1- Input memory blocks with size and processes with size.
2- Initialize all memory blocks as free.
3- Start by picking each process and check if it can
be assigned to current block.
4- If size-of-process <= size-of-block if yes then
assign and check for next process.
5- If not then keep checking the further blocks.

BEST-FIT ALGORITHM

1. Declare structures *hole* and *process* to hold information about set of holes and processes respectively.
2. Get number of holes, say *nh*.
3. Get the size of each hole
4. Get number of processes, say *np*.
5. Get the memory requirements for each process.
6. Allocate processes to holes, by examining each hole as follows:
    a. Sort the holes according to their sizes in ascending order
    b. If hole size > process size then
        i. Mark process as allocated to that hole.
        ii. Decrement hole size by process size.
    c. Otherwise check the next from the set of sorted hole
7. Print the list of process and their allocated holes or unallocated status.
8. Print the list of holes, their actual and current availability.
9. Stop

PROGRAM:

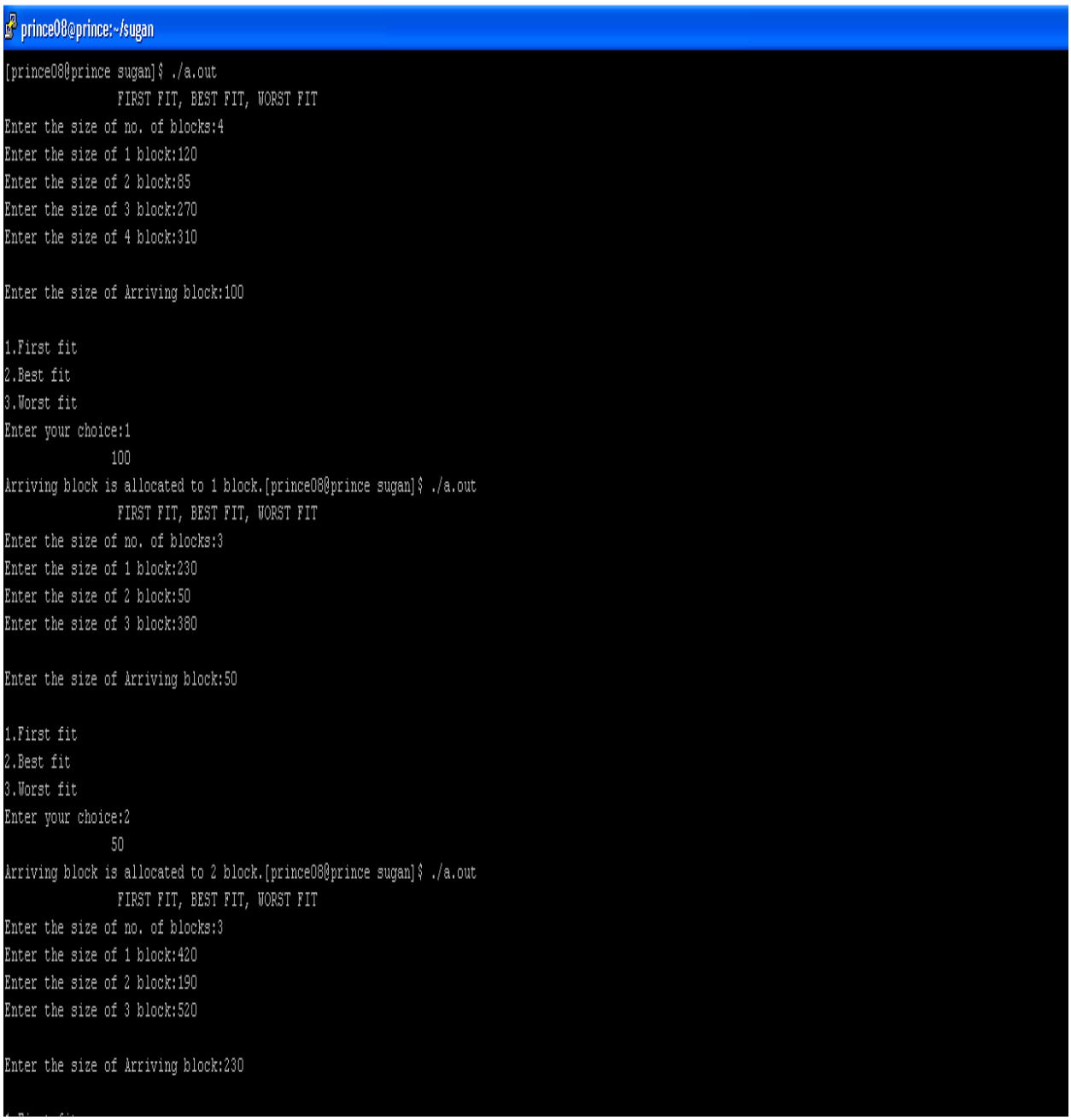
```
#include<stdio.h>

void main()

{

int i,j,temp,b[10],c[10],arr,n,ch,a;

printf("\t\t FIRST FIT, BEST FIT, WORST FIT\n");

printf("Enter the size of no. of blocks:");
```

```
scanf("%d",&n);
for(i=1;i<=n;i++)
{
printf("Enter the size of %d block:",i);
scanf("%d",&b[i]);
c[i]=b[i];
}
printf("\nEnter the size of Arriving block:");
scanf("%d",&arr);
printf("\n1.First fit\n2.Best fit\n3.Worst fit\nEnter your choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:
        for(i=1;i<=n;i++)
        {
        if(b[i]>=arr)
        {
        printf("\t\t%d",arr);
        printf("\nArriving block is allocated to %d block.",i);
        break;
        }
        else
        printf("%d",b[i]);
        continue;
```

```
        }
        break;
case 2:
        for(i=1;i<=n;i++)
        {
        if(b[i]>=b[i+1])
        {
        temp=b[i];
        b[i]=b[i+1];
        b[i+1]=temp;
        }
        }
       for(i=1;i<=n;i++)
        {
        if(b[i]>=arr)
        {
        a=b[i];
        break;
        }
        else
        printf("%d",b[i]);
        continue;
        }
        for(i=1;i<=n;i++)
        {
```

```
            if(c[i]==a)
            {
            printf("\t\t%d",arr);
            printf("\nArriving block is allocated to %d block.",i);
            }
            }
            break;
case 3:
            for(i=1;i<=n;i++)
            {
 if(b[i]>=b[i+1])
            {
            temp=b[i];
            b[i]=b[i+1];
            b[i+1]=temp;
            }
            }
            for(i=1;i<n;i++)
            printf(" %d",b[i]);
            printf("\t%d",arr);
            printf("\n Arriving block is allocated to %d block",i);
            break;
default:
            printf("Enter the valid choice:");
            }}
```

OUTPUT:



RESULT :

Thus the program was executed successfully and hence output verified

Ex.No :12

IMPLEMENTATION OF PAGING TECHNIQUE OF MEMORY MANAGEMENT

DATE:

AIM:

To implement the Paging Technique Of Memory Management

ALGORITHM:

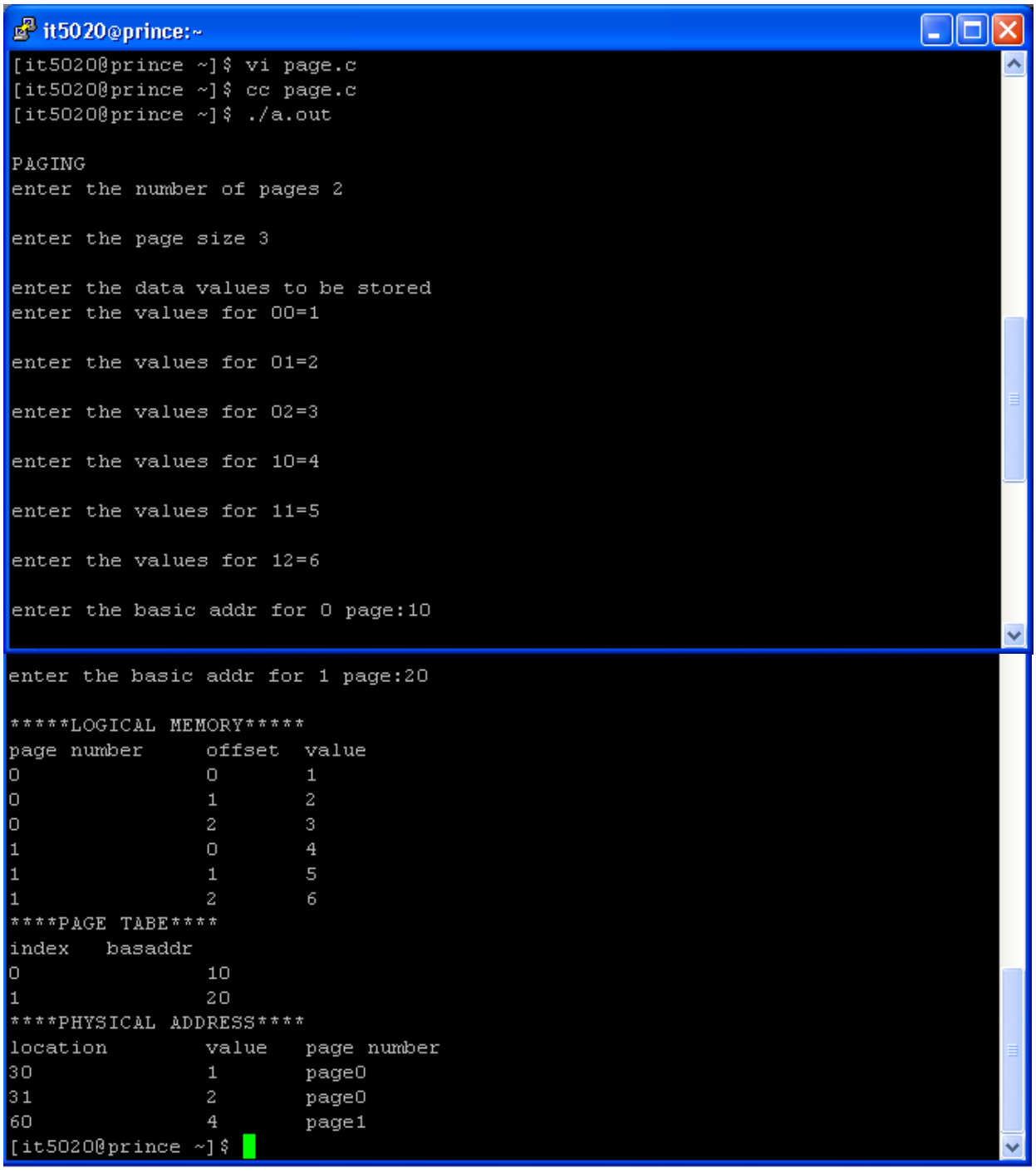
Step 1: Read all the necessary input from the keyboard.
Step 2: Pages - Logical memory is broken into fixed - sized blocks.
Step 3: Frames – Physical memory is broken into fixed – sized blocks.
Step 4: Calculate the physical address using the following
Physical address = (Frame number * Frame size) + offset
Step 5: Display the physical address.
Step 6: Stop the process

PROGRAM:

```
#include<stdio.h>
int main()
{
int lmem[10][10],pmem[10][10],ptabel[10],psize,i,j,n,phyadd;
printf("\nPAGING");
printf("\nenter the number of pages");
scanf("%d",&n);
printf("\nenter the page size");
scanf("%d",&psize);
printf("\nenter the data values to be stored");
for(i=0;i<n;i++)
{
for(j=0;j<psize;j++)
{
printf("\nenter the values for %d%d=",i,j);
scanf("%d",&lmem[i][j]);
}
}
```

```
for(i=0;i<n;i++)
{
printf("\nenter the basic addr for %d page:",i);
scanf("%d",&ptabel[i]);
}
printf("\n*****LOGICAL MEMORY*****");
printf("\npage number\toffset\tvalue");
for(i=0;i<n;i++)
{
for(j=0;j<psize;j++)
{
printf("\n%d\t\t%d\t%d\t",i,j,lmem[i][j]);
}
}
printf("\n****PAGE TABE****");
printf("\nindex\tbasaddr");
for(i=0;i<n;i++)
{
printf("\n%d\t\t%d",i,ptabel[i]);
}
printf("\n****PHYSICAL ADDRESS****");
printf("\nlocation\tvalue\tpage number");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
phyadd=(ptabel[i]*psize)+j;
printf("\n%d\t\t%d\tpage%d",phyadd,lmem[i][j],i);
}
}
}
```

OUTPUT:



RESULT :

Thus the program was executed successfully and hence output verified

Ex.No :13

IMPLEMENTATION OF THE FOLLOWING PAGE REPLACEMENT ALGORITHMS A) FIFO B) LRU C) OPTIMAL

DATE:

AIM:

To Implement The Page Replacement Algorithms A) Fifo B) Lru C) Optimal

ALGORITHM:

Step 1. Start to traverse the pages.
Step 2. If the memory holds fewer pages, then the capacity else goes to step 5.
Step 3. Push pages in the queue one at a time until the queue reaches its maximum capacity or all page requests are fulfilled.
Step 4. If the current page is present in the memory, do nothing.
Step 5. Else, pop the topmost page from the queue as it was inserted first.
Step 6. Replace the topmost page with the current page from the string.
Step 7. Increment the page faults.
Step 8. Stop

PROGRAM:

A] C program for FIFO page replacement algorithm

```
#include <stdio.h>
int main()
{
int incomingStream[] = {4, 1, 2, 4, 5};
int pageFaults = 0;
int frames = 3;
int m, n, s, pages;

pages = sizeof(incomingStream)/sizeof(incomingStream[0]);

printf("Incoming \t Frame 1 \t Frame 2 \t Frame 3");
int temp[frames];
for(m = 0; m < frames; m++)
{
temp[m] = -1;
}

for(m = 0; m < pages; m++)
{
s = 0;

for(n = 0; n < frames; n++)
{
```

```
if(incomingStream[m] == temp[n])
{
s++;
pageFaults--;
}
}
pageFaults++;

if((pageFaults <= frames) && (s == 0))
{
temp[m] = incomingStream[m];
}
else if(s == 0)
{
temp[(pageFaults - 1) % frames] = incomingStream[m];
}

printf("\n");
printf("%d\t\t\t",incomingStream[m]);
for(n = 0; n < frames; n++)
{
if(temp[n] != -1)
printf(" %d\t\t\t", temp[n]);
else
printf(" - \t\t\t");
}
}

printf("\nTotal Page Faults:\t%d\n", pageFaults);
return 0;
}
```

OUTPUT:

```
Incoming Frame 1 Frame 2 Frame 3
4       4       -       -
1       4       1       -
2       4       1       2
4       4       1       2
5       5       1       2
Total Page Faults: 4
```

B] C program for LRU page replacement algorithm

AIM:

To write a c program to implement LRU page replacement algorithm

ALGORITHM :

1. Start the process

2. Declare the size

3. Get the number of pages to be inserted

4. Get the value

5. Declare counter and stack

6. Select the least recently used page by counter value

7. Stack them according the selection.

8. Display the values

9. Stop the process

PROGRAM:

```
#include<stdio.h>
main()
{
        int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];
        printf("Enter no of pages:");
        scanf("%d",&n);
        printf("Enter the reference string:");
        for(i=0;i<n;i++)
scanf("%d",&p[i]);
        printf("Enter no of frames:");
        scanf("%d",&f);
        q[k]=p[k];
        printf("\n\t%d\n",q[k]);
        c++;
        k++;
        for(i=1;i<n;i++)
{
c1=0;
for(j=0;j<f;j++)
```

```
{
if(p[i]!=q[j])
c1++;
}
if(c1==f)
{
c++;
if(k<f)
{
q[k]=p[i];
k++;
for(j=0;j<k;j++)
printf("\t%d",q[j]);
printf("\n");
}
else
{
for(r=0;r<f;r++)
{
c2[r]=0;
for(j=i-1;j<n;j--)
{
if(q[r]!=p[j])
c2[r]++;
else
break;
}
}
for(r=0;r<f;r++)
b[r]=c2[r];
for(r=0;r<f;r++)
{
for(j=r;j<f;j++)
{
if(b[r]<b[j])
{
t=b[r];
b[r]=b[j];
b[j]=t;
}
}
}
for(r=0;r<f;r++)
{
if(c2[r]==b[0])
q[r]=p[i];
```

```
printf("\t%d",q[r]);
}
printf("\n");
}
}
}
printf("\nThe no of page faults is %d",c);
}
```

OUTPUT:

```
Enter no of pages:10
Enter the reference string:7 5 9 4 3 7 9 6 2 1
Enter no of frames:3
7
7       5
7       5       9
4       5       9
4       3       9
4       3       7
9       3       7
9       6       7
9       6       2
1       6       2

The no of page faults is 10
```

C] C program for OPTIMAL page replacement algorithm

```
#include<stdio.h>
#include<conio.h>
main()
{
int fr[5],i,j,k,t[5],p=1,flag=0,page[25],psz,nf,t1,u[5];
clrscr();
printf("enter the number of frames:");
scanf("%d",&nf);
printf("\n enter the page size");
scanf("%d",&psz);

printf("\nenter the page sequence:");
for(i=1; i<=psz; i++)
scanf("%d",&page[i]);

for(i=1; i<=nf; i++)
```

```
fr[i]=-1;
for(i=1; i<=psz; i++)
{
if(full(fr,nf)==1)
break;
else
{
flag=0;
for(j=1; j<=nf; j++)
{
if(page[i]==fr[j])
{
flag=1;
printf("          \t%d:\t",page[i]);
break;
}
}
if(flag==0)
{
fr[p]=page[i];
printf("          \t%d:\t",page[i]);
p++;
}

for(j=1; j<=nf; j++)
printf(" %d  ",fr[j]);
printf("\n");
}
}
p=0;
for(; i<=psz; i++)
{
flag=0;
for(j=1; j<=nf; j++)
{
if(page[i]==fr[j])
{
flag=1;
break;
}
}
if(flag==0)
{
p++;
for(j=1; j<=nf; j++)
{
```

```
for(k=i+1; k<=psz; k++)
{
if(fr[j]==page[k])
{
u[j]=k;
break;
}
else
u[j]=21;
}
}
for(j=1; j<=nf; j++)
t[j]=u[j];
for(j=1; j<=nf; j++)
{
for(k=j+1; k<=nf; k++)
{
if(t[j]<t[k])
{
t1=t[j];
t[j]=t[k];
t[k]=t1;
}
}
}
for(j=1; j<=nf; j++)
{
if(t[1]==u[j])
{
fr[j]=page[i];
u[j]=i;
}
}
printf("page fault\t");
}
else
printf("          \t");
printf("%d:\t",page[i]);
for(j=1; j<=nf; j++)
printf(" %d  ",fr[j]);
printf("\n");
}
printf("\ntotal page faults:  %d",p+3);
// getch();
}
int full(int a[],int n)
```

```
{
int k;
for(k=1; k<=n; k++)
{
if(a[k]==-1)
return 0;
}
return 1;
}
```

OUTPUT:

```
enter the number of frames:5

 enter the page size2

enter the page sequence:1
2
            1:      1   -1   -1   -1   -1
            2:      1   2   -1   -1   -1

total page faults:  3
```

RESULT :

Thus the program was executed successfully and hence output verified

Ex.No :14

IMPLEMENTATION OF THE VARIOUS FILE ORGANIZATION TECHNIQUES

DATE:

AIM:

To Implement The various file organization techniques

Algorithm for Single Level Directory Structure:

Step 1:Start
Step 2: Initialize values gd=DETECT,gm,count,i,j,mid,cir_x;
Initialize character array fname[10][20];
Step 3: Initialize graph function as
Initgraph(& gd, &gm," c:/tc/bgi");
Clear device();
Step 4:set back ground color with setbkcolor();
Step 5:read number of files in variable count.
Step 6:if check i<count
Step 7: for i=0 & i<count
i increment;
Cleardevice();
setbkcolor(GREEN);
read file name;
setfillstyle(1,MAGENTA);
Step 8: mid=640/count;
cir_x=mid/3;
bar3d(270,100,370,150,0,0);
settextstyle(2,0,4);
settextstyle(1,1);
outtextxy(320,125,"rootdirectory");
setcolor(BLUE);
i++;
Step 9:for j=0&&j<=i&&cir_x+=mid
j increment;
line(320,150,cir_x,250);
fillellipse(cir_x,250,30,30);
outtextxy(cir_x,250,fname[i]);
Step 10: End

PROGRAM

1. SINGLE LEVEL DIRECTORY ORGANIZATION

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir;
void main()
{
int i,ch;
char f[30];
clrscr();
dir.fcnt = 0;
printf("\nEnter name of directory -- ");
scanf("%s", dir.dname);
while(1)
{
printf("\n\n 1. Create File\t2. Delete File\t3. Search File \n 4. Display Files\t5. Exit\nEnter your
choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\n Enter the name of the file -- ");
scanf("%s",dir.fname[dir.fcnt]);
dir.fcnt++;
break;
case 2: printf("\n Enter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is deleted ",f);
strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);
break;
}
}
if(i==dir.fcnt)
```

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

OUTPUT:

Enter name of directory -- CSE

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- A

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- B

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- C

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 4

The Files are -- A B C

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 3

Enter the name of the file – ABC

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

Enter the name of the file – B
File B is deleted

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 5

2. TWO LEVEL DIRECTORY ORGANIZATION

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir[10];
void main()
{
int i,ch,dcnt,k;
char f[30], d[30];
clrscr();
dcnt=0;
while(1)
{
printf("\n\n 1. Create Directory\t 2. Create File\t 3. Delete File");
printf("\n 4. Search File \t \t 5. Display \t 6. Exit \t Enter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\n Enter name of directory -- ");
scanf("%s", dir[dcnt].dname);
dir[dcnt].fcnt=0;
dcnt++;
printf("Directory created");
break;
case 2: printf("\n Enter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",dir[i].fname[dir[i].fcnt]);
dir[i].fcnt++;
printf("File created");
break;
}
if(i==dcnt)
printf("Directory %s not found",d);
break;
```

```
case 3: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is deleted ",f);
dir[i].fcnt--;
strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
goto jmp;
}
}
printf("File %s not found",f);
goto jmp;
}
}
printf("Directory %s not found",d);
jmp : break;
case 4: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{

if(strcmp(d,dir[i].dname)==0)
{
printf("Enter the name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is found ",f);
goto jmp1;
}
```

```
}
printf("File %s not found",f);
goto jmp1;
}
}
printf("Directory %s not found",d);
jmp1: break;
case 5: if(dcnt==0)
printf("\nNo Directory's ");
else
{
printf("\nDirectory\tFiles");
for(i=0;i<dcnt;i++)
{
printf("\n%s\t\t",dir[i].dname);
for(k=0;k<dir[i].fcnt;k++)
printf("\t%s",dir[i].fname[k]);
}
}
break;
default:exit(0);
}
}
getch();
}
```

OUTPUT:

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 1

Enter name of directory -- DIR1
Directory created

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 1

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

Enter name of the directory – DIR1
Enter name of the file -- A1
File created

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 2

Enter name of the directory – DIR1
Enter name of the file -- A2
File created

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 2

Enter name of the directory – DIR2
Enter name of the file -- B1
File created

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 5
Directory Files
DIR1 A1 A2
DIR2 B1
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 4

Enter name of the directory – DIR
Directory not found
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 3
Enter name of the directory – DIR1
Enter name of the file -- A2
File A2 is deleted
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice – 6

RESULT :

Thus the program was executed successfully and hence output verified

Ex.No :15

IMPLEMENTATION OF THE FOLLOWING FILE ALLOCATION STRATEGIES A) SEQUENTIAL B) INDEXED C) LINKED

DATE:

AIM:

To write a C program for sequential file for processing the student information.

ALGORITHM:

Step-1: Start the program.
Step-2: Get the number of files user want to store in the system.
Step-3: Using Standard Library function open the file to write the data into the file.
Step-4: Store the entered information in the system.
Step-5: Using file name, starting block, and its size to display.
Step-6: Close the file using fclose() function.
Step-7: Process it and display the result.
Step-8: Stop the program.

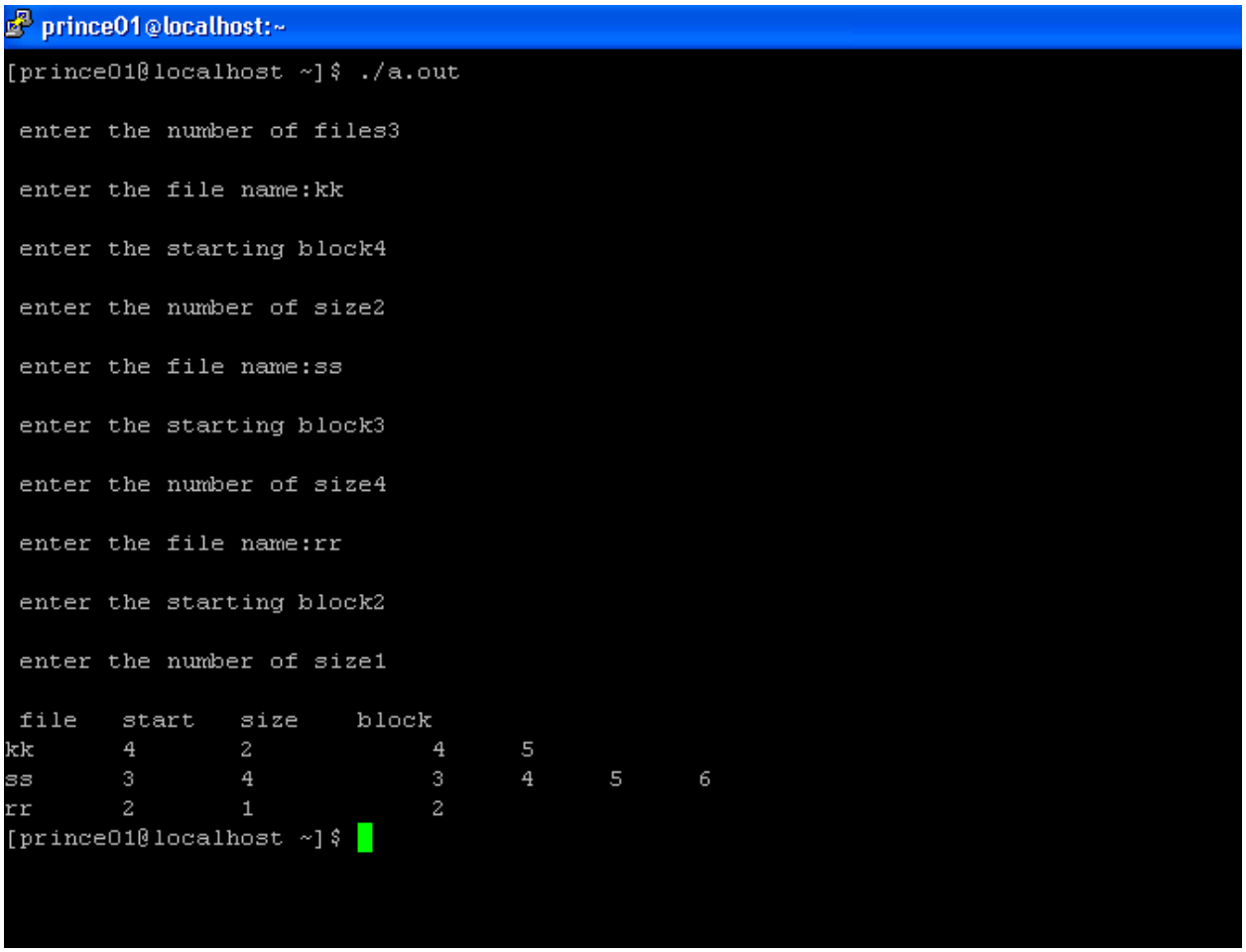
PROGRAM:

A] SEQUENTIAL FILE ALLOCATION

```
#include<stdio.h>
struct file
{
char fname[10];
int start;
int size,block[10];
}f[10];
main()
{
int i,j,n;
printf("\n enter the number of files");
scanf("%d",&n);
for(i=0;i<n;i++)
```

```
{
printf("\n enter the file name:");
scanf("%s",f[i].fname);
printf("\n enter the starting block");
scanf("%d",&f[i].start);
f[i].block[0]=f[i].start;
printf("\n enter the number of size");
scanf("%d",&f[i].size);
}
printf("\n 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;j++)
printf("%6d",f[i].start+(j-1));
printf("\n");}}
```

OUTPUT:



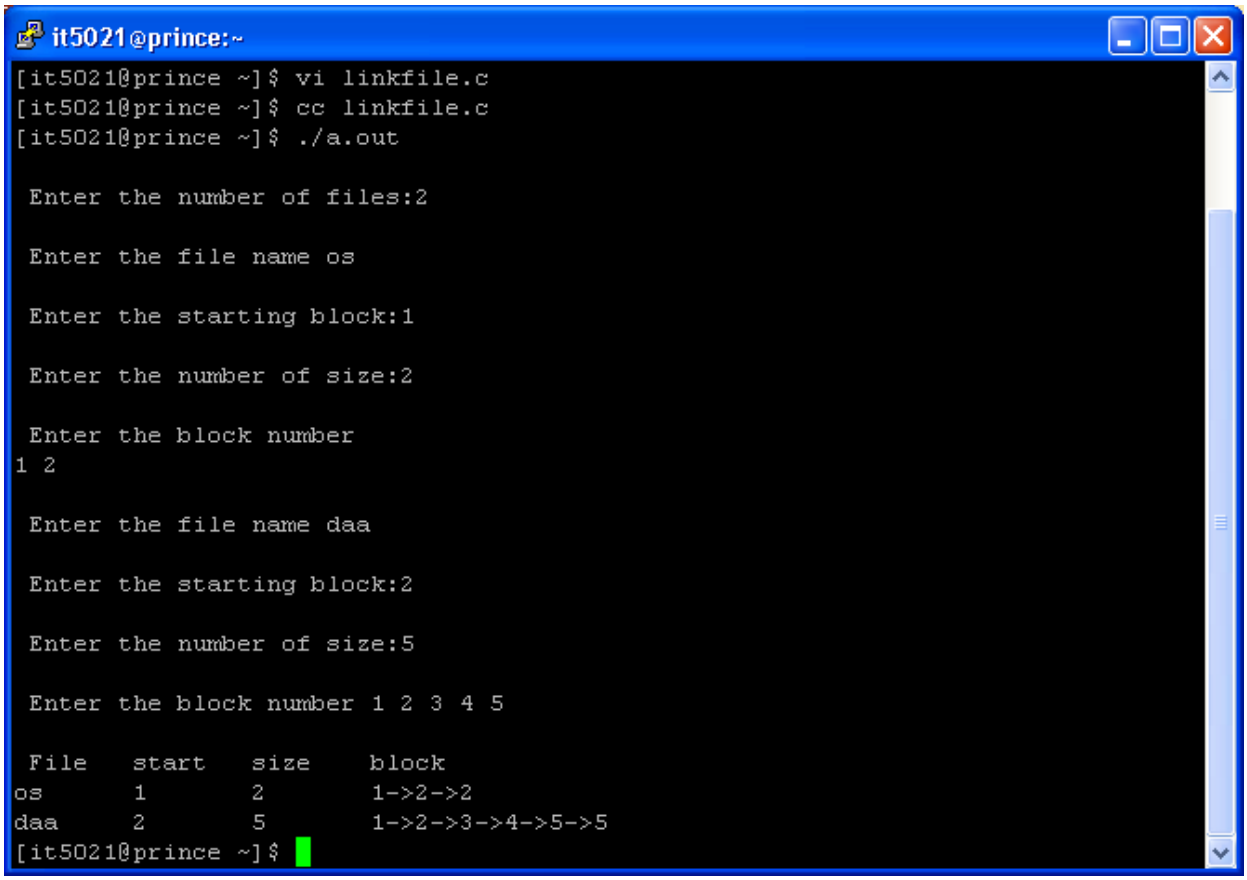
B] LINKED FILE ALLOCATION

PROGRAM:

```
#include<stdio.h>
struct file
{
char fname[10];
int start;
int size;
int block[10];
```

```
}f[10];
main()
{
int i,j,n;
printf("\n Enter the number of files:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\n Enter the file name");
scanf("%s",f[i].fname);
printf("\n Enter the starting block:");
scanf("%d",&f[i].start);
f[i].block[0]=f[i].start;
printf("\n Enter the number of size:");
scanf("%d",&f[i].size);
printf("\n Enter the block number");
for(j=1;j<=f[i].size;j++)
{
scanf("%d",&f[i].block[j]);
}
}
printf("\n 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;j++)
printf("%d->",f[i].block[j]);
printf("%d",f[i].block[j-1]);
printf("\n");
}
}
```

OUTPUT:



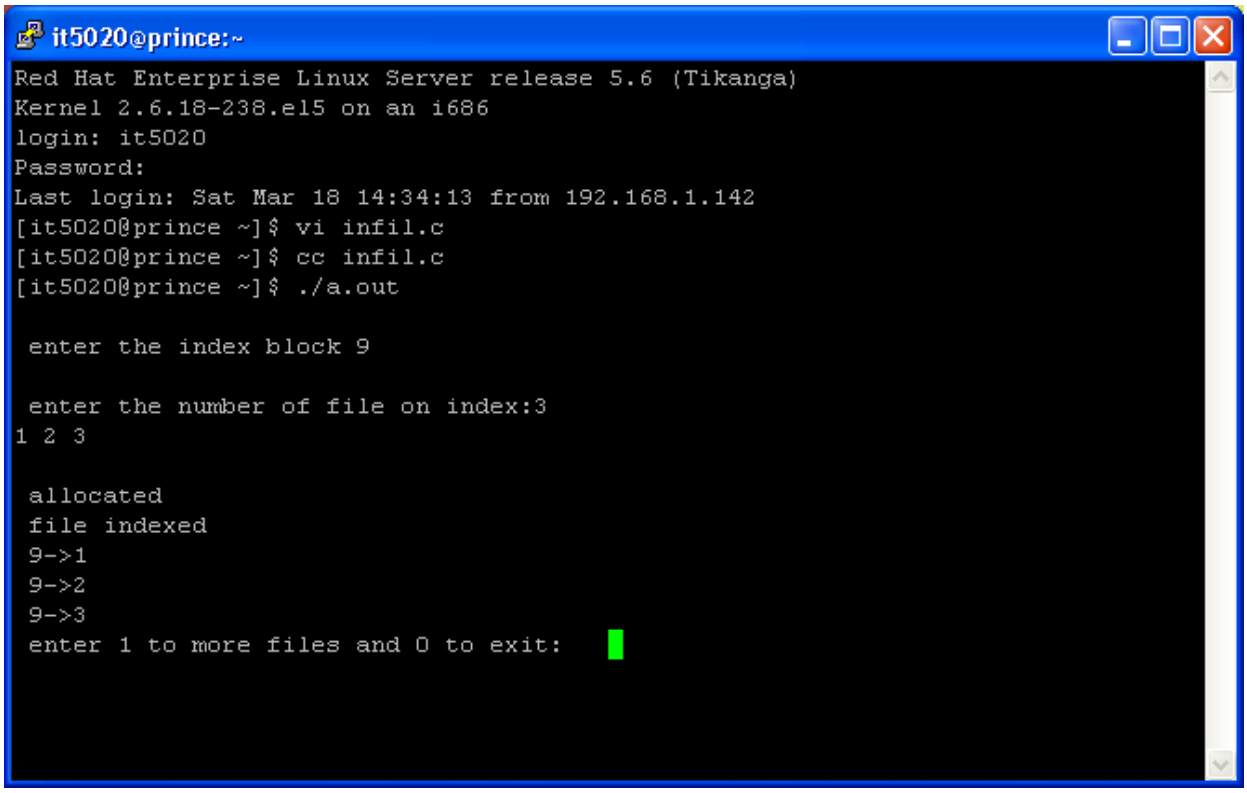
C] INDEX FILE ALLOCATION

PROGRAM:

```
#include<stdio.h>
main()
{
int f[50],i,k,j,inde[50],count=0,n,c,p;
for(i=0;i<50;i++)
f[i]=0;
x:
printf("\n enter the index block");
scanf("%d",&p);
if(f[p]==0)
{
```

```
f[p]=1;
printf("\n enter the number of file on index:");
scanf("%d",&n);
}
else
{
printf("\n enter the number of blocks already allocated");
goto x;
}
for(i=0;i<n;i++)
scanf("%d",&inde[i]);
for(i=0;i<n;i++)
if(f[inde[i]]==1)
{
printf("\n index already allocated");
goto x;
}
for(j=0;j<n;j++)
f[inde[i]]=1;
printf("\n allocated");
printf("\n file indexed");
for(k=0;k<n;k++)
printf("\n %d->%d",p,inde[k]);
printf("\n enter 1 to more files and 0 to exit:\t");
scanf("%d",&c);
if(c==1)
goto x;
else
return(0);
}
```

OUTPUT:



RESULT :

Thus the program was executed successfully and hence output verified