LAB MANUAL

FOR

"18ITL41 – OPERATING SYSTEMS LABORATORY"

R-2018

IV SEMESTER

B.Tech - INFORMATION TECHNOLOGY

(For Private Circulation Only)

DEPARTMENT OF INFORMATION TECHNOLOGY

VELALAR COLLEGE OF ENGINEERING AND TECHNOLOGY

(Autonomous)

Affiliated to Anna University, Chennai

THINDAL, ERODE – 638 012

“18ITL41 – *OPERATING SYSTEMS LABORATORY*”	
Do’s	**Don’ts**
Use RED HAT LINUX 6.0 ,TELNET to execute your program.	**Don’t use pen drives.**
Write observation before entering in to the lab.	**Don’t make noise inside the lab.**
Take print out of your program.	**Don’t use others login area.**
Use afternoon hours and evening lab sessions to complete your program.	**Don’t save anything on the local machine. Use only your login area.**

Contents

Introduction

There are 3 hours allocated to a laboratory session in *DATA COMMUNICATION Lab*. It is a necessary part of the course at which attendance is compulsory.

Here are some guidelines to help you perform the experiments and to submit the reports:

1. Please treat the computer components with care, as they are very expensive.
2. Read the laboratory documentation prior to each lab session.
3. Read the question section before leaving the lab because some of the questions may require observations of the lab equipment.
4. Before leaving the lab, place the stools under the lab bench.
5. Before leaving the lab, turn off the power to all computer components including the printer.
6. Before leaving the lab, turn off the main power switch to the lab bench.
7. Read all instructions carefully and carry them all out.
8. Ask a system administrator if you are unsure of anything.
9. Write up full and suitable conclusions for each experiment.
10. If you have any doubt about the safety of any procedure, contact the system administrator beforehand.
11. **THINK** about what you are doing!

Guidelines for Laboratory Reports

1. Laboratory reports will be due at each lab meeting. Work that was performed the previous lab meeting is to be documented and turned in the following week at the beginning of the lab period.
2. Late reports will have points deducted. A report will be considered one day late if it is handed in after the lab has started.
3. All labs **must** be performed and all lab reports **must** be turned in to pass the course. This includes reports which are so late that they will be counted as an automatic zero.
4. Even while laboratory exercises are performed in pairs, lab reports are to be written on an individual basis. **The lab report turned in by each student must be entirely their own work**.

Minimum Requirement of Hardware and Software:

SOFTWARE REQUIRED	–	RED HAT LINUX 6.0 ,TELNET
OPERATING SYSTEM	–	WINDOWS XP- SP3
COMPUTERS REQUIRED	–	INTEL CORE i3, 500 GB HARD DISK,4 GB RAM

18ITL41 **OPERATING SYSTEMS LABORATORY** **L T P C**

(Common to B.E/B.Tech. – CS & IT Programmes) **0 0 2 1**

Preamble:

The main aim of this course is to implement the concepts of operating system mechanisms and policies. The course makes the students to implement the program that provide the optimal solution for CPU scheduling, synchronization problems and deadlocks. This course will also enable the students to apply the designed module to appropriate memory, file and disk management technique for effective resource utilization.

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

1. Develop the suitable shell commands to establish user interface with UNIX kernel.
2. Develop C Program to implement CPU scheduling algorithms, deadlock avoidance algorithms and page replacement algorithms for a given set of processes considering arrival time, burst time and resources.
3. Develop C program to implement thread, process synchronization and Inter Process Communication for a given set of processes by using semaphore and shared memory mechanisms
4. Construct a C program to implement file allocation and organization techniques for a given set of files by using sequential, indexed and linked file allocation methods.
5. Develop C Program for memory management by using paging technique.

LIST OF EXPERIMENTS

1. Study and execute the basic commands of UNIX operating system for resource management. 1
2. Write a C program to implement FCFS, SJF and Round Robin (time quantum=2) CPU scheduling algorithms for process Scheduling. 3
3. Write a C program to implement the file allocation strategies such as a) Sequential b) Indexed c) Linked for ‘n’ number of files.9
4. Write a C program to implement the process synchronization for producer – consumer problem using semaphore.4
5. Write a C program to implement single and two-level directory structure for effective file organization. 10
6. Write a C program to implement the Bankers Algorithm for deadlock avoidance and detection.6
7. Write a C program to implement the FIFO and LRU page replacement algorithm for the following reference string:1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6. 8
8. Write a C program to implement shared memory between the two-process using Inter Process communication (IPC) primitives.2
9. OS uses a paging system with 2Kbytes pages. A given processes uses a virtual address space of 8Kbytes and it is assigned 8Kbytes of physical memory. Write a C program to create page table and implement Paging Technique to find out physical address for the above scenario.7
10. Write a C program to create 2 threads named Th1 and Th2 and establish synchronization between two threads using mutex lock.5

SOFTWARE

- Linux :Ubuntu / OpenSUSE / Fedora / Red Hat / Debian / Mint OS
- GCC compiler

TOTAL: 45 PERIODS

CF – 03
Rev.0
24.07.15

VELALAR COLLEGE OF ENGINEERING AND TECHNOLOGY, ERODE – 12
(AUTONOMOUS)
DEPARTMENT OF INFORMATION TECHNOLOGY
PO-CO Mapping

Course Code & Name	**: 18ITL41 – Operating Systems Laboratory**
Academic Year	**: 2019 - 2020**
Regulation	**: 2018**

VISION

To provide technological excellence in learning in the field of Information Technology and promote academic and research activities.

MISSION

- Attain technological excellence through quality education and well-designed curriculum adaptable to technological needs.
- Facilitate value added courses and interdisciplinary training to provide highly competent IT professionals and entrepreneurs.
- Produce socially aware and ethically responsible engineers through human value education.
- Impart collaborative research through academic projects and industry institute interactions.

PROGRAMME EDUCATIONAL OBJECTIVES

1. Excel in post graduate studies and career.
2. Solve problems using the engineering fundamentals.
3. Have engineering breadth to achieve solutions for the needs of society.
4. Develop professional and ethical values, effective communication, team work and managerial skills for societal and environmental contexts.
5. Attain professional competence through lifelong learning.

PROGRAMME OUTCOMES

1. Apply the knowledge of mathematics science and engineering fundamentals.
2. Identify, formulate, review research literature and analyze complex problems.
3. Design solutions for complex engineering problems and design system components or process that meet specified needs.
4. Design experiments, analyze, interpret data and synthesis information to provide valid conclusions.
5. Create, select and apply appropriate techniques, resources and modern engineering and IT tools.
6. Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues.

7. Understand the impact of the professional engineering solutions in societal and environmental contexts and need for sustainable development.
8. Apply ethical principles and commit to professional ethics and responsibilities.
9. Function effectively as an individual and as a member or leader in diverse teams.
10. Communicate effectively with engineering community and society.
11. Demonstrate knowledge and understanding of the engineering and management principles to manage projects.
12. Demonstrate the attitude for life-long learning.

PROGRAMME SPECIFIC OUTCOME (PSO)

1. Ability to use and apply current technical concepts and practices in core information technologies of human computer interaction, information management, programming, networking & web systems and technologies.
2. Ability to identify and analyze user needs to effectively integrate IT-based solutions into the user environment.

COURSE OUTCOMES – 18ITT32 – Computer Architecture

After the completion of this course, students will be able to

C216.1. Develop the suitable shell commands to establish user interface with UNIX kernel.

C216.2. Develop C Program to implement CPU scheduling algorithms, deadlock avoidance algorithms and page replacement algorithms for a given set of processes considering arrival time, burst time and resources.

C216.3. Develop C program to implement thread, process synchronization and Inter Process Communication for a given set of processes by using semaphore and shared memory mechanisms.

C216.4. Construct a C program to implement file allocation and organization techniques for a given set of files by using sequential, indexed and linked file allocation methods.

C216.5. Develop C Program for memory management by using paging technique.

Course outcome	**PO 1**	**PO 2**	**PO 3**	**PO 4**	**PO 5**	**PO 6**	**PO 7**	**PO 8**	**PO 9**	**PO 10**	**PO 11**	**PO 12**	**PSO 1**	**PSO 2**
C216.1	-	-	-	3	-	-	-	-	1	-	-	-	-	-
C216.2	-	-	-	3	-	-	-	-	1	-	-	-	-	-
C216.3	-	-	-	3	-	-	-	-	1	-	-	-	-	-
C216.4	-	-	-	3	-	-	-	-	1	-	-	-	-	-
C216.5	-	-	-	3	-	-	-	-	1	-	-	-	-	-

1: Slight (Low) ***2: Moderate (Medium)*** ***3: Substantial (High)*** ***"-" No correlation***

CONTENTS

EX.NO.1

BASICS OF UNIX COMMANDS

AIM :

To study and execute the commands in unix.

COMMANDS:

DATE :

This command is used to display the current data and time.

Syntax :

```
$date
$date +% options
```

Options :

a - Abbrevated weekday.

A - Full weekday.

b - Abbrevated month.

B - Full month.

c - Current day and time.

C - Display the century as a decimal number.

d - Day of the month.

D - Day in „mm/dd/yy‟ format

h - Abbrevated month day.

H - Display the hour.

l - Day of the year.

m - Month of the year.

M - Minute.

P - Display AM or PM

S - Seconds

T - HH:MM:SS format

u - Week of the year.

y - Display the year in 2 digit.

Y - Display the full year.

Z - Time zone .

To change the format :

Syntax : $date “+%H-%M-%S”

CALENDER :

This command is used to display the calendar of the year or the particular month of calendar year.

Syntax :

```
$cal  year
$cal  month  year
```

Here the first syntax gives the entire calendar for given year & the second Syntax gives the calendar of reserved month of that year.

ECHO :

This command is used to print the arguments on the screen .

Syntax : $echo   text

Multi line echo command :

To have the output in the same line , the following commands can be used.

Syntax : $echo text\ text

To have the output in different line, the following command can be used.

Syntax :

```
$echo “text
>line2
>line3”
```

WHO:

It is used to display who are the users connected to our computer currently.

Syntax : $who – options

Options :

H–Display the output with headers.

b–Display the last booting date or time or when the system was lastly rebooted.

WHO AM I:

Display the details of the current working directory.

Syntax : $who am i

TTY :

It will display the terminal name.

Syntax : $tty

BINARY CALCULATOR :

It will change the “$” mode and in the new mode, arithmetic operations such as +,-,*,/,%,n,sqrt(),length(),=, etc can be performed . This command is used to go to the binary calculus mode.

Syntax : $bc

Operations

CLEAR :

It is used to clear the screen.

Syntax : $clear

MAN :

It help us to know about the particular command and its options & working. It is like „help‟ command in windows .

Syntax : $man command name

TPUT :

It is used to manipulate the screen.

Syntax : $tput arguments

Arguments :

Clear – to clear the screen.

Longname – Display the complete name of the terminal.

SMSO – background become white and foreground become black color.

rmso – background become black and foreground becomes white color.

Cop R C – Move to the cursor position to the specified location.

Cols – Display the number of columns in our terminals.

LIST :

It is used to list all the contents in the current working directory.

Syntax : $ ls – options

If the command does not contain any argument means it is working in the Current directory.

Options :

a– used to list all the files including the hidden files.

c– list all the files columnwise.

d- list all the directories.

m- list the files separated by commas.

p- list files include „/‟ to all the directories.

r- list the files in reverse alphabetical order.

f- list the files based on the list modification date.

x-list in column wise sorted order.

PRESENT WORKING DIRECTORY :

To print the complete path of the current working directory.

Syntax : $pwd

MKDIR :

To create or make a new directory in a current directory .

Syntax : $mkdir directory name

CD :

To change or move the directory to the mentioned directory .

Syntax : $cd directory name.

To quitting from a directory.

Syntax : $cd **. .**

RMDIR :

To remove a directory in the current directory & not the current directory itself.

Syntax : $rmdir directory name

CREATE A FILE :

To create a new file in the current directory. The > symbol is redirection operator.

Syntax : $cat >filename

DISPLAY THE CONTENTS OF THE FILE :

To display the content of file.

Syntax : $cat filename

To display the contents of the file in reverse manner.

Syntax : $tac filename

SORTING A FILE:

To sort the contents in alphabetical order.

Syntax : $sort filename

To sort the contents in reverse order.

Syntax : $sort -r filename

COPYING CONTENTS FROM ONE FILE TO ANOTHER :

To copy the contents from source to destination file.

so that both contents are same.

Syntax : $cp source filename destination filename

$cp source filename path destination filename path

MOVE :

To completely move the contents from source file to destination file and to remove the source file.

Syntax : $ mv source filename destination filename

REMOVE :

To permanently remove the file we use this command .

Syntax : $rm filename

WORD COUNT:

To list the content count of no of lines , words, characters .

Syntax : $wc filename

$wc - filename

Options :

c – to display no of characters.

l – to display only the lines.

w – to display the no of words.

LINE PRINTER :

To print the line through the printer.

Syntax : $lp filename

PAGE :

This command is used to display the contents of the file page wise & next page can be viewed by pressing the enter key.

Syntax : $pg filename

DISK FREE:

This command displays the information of device name, total blocks, total disk space, used disk space, available disk space and mount points on a file system.

Syntax : $df

This command displays all file systems and their disk usage in "human readable" formatting.

Syntax : $df -h

DIFF :

This command will compare the two files and print out the differences between.

Syntax: $diff firstfilename secondfilename

UNIQ:

This command reports or filters out repeated lines in a file.

Syntax: $uniq filename

Syntax: $uniq options filename

This command Prefix lines with a number representing how many times they occurred.

Syntax: $uniq -c filename

This command Only print unique lines.

Syntax: $uniq -u filename

LINK:

This command is used to create a link to a file.

Syntax: $ln firstfilename secondfilename

COMPARE:

This command compares the two files.

Syntax: $cmp firstfilename secondfilename

This command compares the two directories.

Syntax: $dircmp firstfilename secondfilename

COMM:

This command Compare two sorted files line-by-line. comm produces three-column output. Column one contains lines unique to FILE1, column two contains lines unique to FILE2, and column three contains lines common to both files. Each of these columns can be suppressed individually with options.

Synatx: $comm firstfilename secondfilename

TOP:

This command is used to know about the system information. The top command provides real-time view of the running system and also the list of tasks currently managed by the kernel. Top is a non-interactive command and provides limited interactive options to the users.

Syntax: $top -options

Options:

b- Starts top command in batch mode. Useful for sending top output to other programs or file.

d- Specify the delay time between the screen updates.

n- Number of iterations, the top should produce before ending.

u- Monitor only the specified user processes.

p- Monitor only the specified processes. Specify the process ID

HEAD:

It is used to display the top ten lines of file.

Syntax: $head filename

$head -linenumber filename

TAIL :

This command is used to display the last ten lines of file.

Syntax: $tail filename

$tail -linenumber filename

PAGE :

This command shows the page by page a screenfull of information is displayed after which the page command displays a prompt and passes for the user to strike the enter key to continue scrolling.

Syntax: $ls –a\p

MORE :

It also displays the file page by page .To continue scrolling with more command , press the space bar key.

Syntax: $more filename

GREP :

This command is used to search and print the specified patterns from the file.

Syntax: $grep search pattern filename

$grep “search pattern” filename

$grep “^ search pattern” filename

EGREP:

This command is used to search the multiple patterns in a file using single command line separated by “|” symbol.

Syntax: $grep “search pattern1 | search pattern2 | search pattern3” filename

FGREP:

This command is used to search the multiple patterns in a file using multiple lines.

Syntax: $grep "search pattern1
>search pattern2
>search pattern3" filename

CUT:

This command selects a list of columns or fields from one or more files.

Syntax: $cut options filename

Options:

-c is for columns

-f for fields

PASTE:

This command merge the lines of one or more files into vertical columns separated by a tab.

Syntax: $cut firstfilename secondfilename >Outputfilename

SORT :

This command is used to sort the data's in ascending order.

Syntax: $sort filename

This command is used to sort the data's in reverse (descending) order.

Syntax: $sort -r filename

COMPRESS:

This command is used to compress a file so that it becomes smaller in size.

Syntax: $gzip filename

DECOMPRESS:

This command is used to decompress a file so that it back to its original size.

Syntax: $gunzip filename

PIPE :

It is a mechanism by which the output of one command can be channeled into the input of another command.

Syntax: $who | wc-l

CHMOD:

This command is used to change the permissions of files or directories permissions defines the permissions for the owner of the file (the "user"), members of the group who owns the file (the "group"), and anyone else ("others").

Permissions:

- The user can read, write, and execute it;
- Members of your group can read and execute it; and
- Others may only read it.

The digits **7**, **5**, and **4** each individually represent the permissions for the user, group, and others, in that order.

Each digit is a combination of the numbers **4**, **2**, **1**, and **0**:

- **4** stands for "read",
- **2** stands for "write",
- **1** stands for "execute", and
- **0** stands for "no permission."

So **7** is the combination of permissions **4**+**2**+**1** (read, write, and execute), **5** is **4**+**0**+**1** (read, no write, and execute), and **4** is **4**+**0**+**0** (read, no write, and no execute).

Syntax:

```
$chmod permissions filename
chmod u=rwx,g=rx,o=r myfile
```

FINGER:

This command is used to print information about logged-in users.

Syntax:

```
$finger
```

MESG:

The message command is used to give permission to other users to send message to your terminal.

Syntax:

```
$mesg y
```

WRITE :

This command is used to communicate with other users, who are logged in at the same time.

Syntax: $write username

WALL :

This command sends message to all users those who are logged in using the unix server.

Syntax: $wall message

MAIL :

It refers to textual information that can be transferred from one user to another

Syntax: $mail user name

REPLY :

It is used to send reply to specified user.

Syntax: $reply user name

<u>OUTPUT</u>

DATE :

```
[student@localhost ~]$ date
Mon Dec  18  03:21:59 IST 2018
[student@localhost ~]$ date +%a
Mon
[student@localhost ~]$ date +%b
Jan
[student@localhost ~]$ date +%A
Monday
[student@localhost ~]$ date +%B
January
[student@localhost ~]$ date +%c
Mon 02 Jan 2017 03:24:49 PM IST
[student@localhost ~]$ date +%C
20
[student@localhost ~]$ date +%d
02
[student@localhost ~]$ date +%D
02/01/17
```

```
[student@localhost ~]$ date +%h
Jan
[student@localhost ~]$ date +%H
03
[student@localhost ~]$ date +%m
01
[student@localhost ~]$ date +%M
24
[student@localhost ~]$ date +%P
pm
[student@localhost ~]$ date +%S
41
[student@localhost ~]$ date +%T
03:24:54
[student@localhost ~]$ date +%u
2
[student@localhost ~]$ date +%y
17
[student@localhost ~]$ date +%Y
2017
[student@localhost ~]$ date +%Z
IST
```

CALENDER :

```
[student@localhost ~]$ cal 01 2018
   January 2018
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
```

ECHO :

```
[student@localhost ~]$ echo Welcome
Welcome
[student@localhost ~]$ echo Welcome to \ OS \ Lab
Welcome  to OS Lab
[student@localhost ~]$ echo " Welcome to
```

```
> OS
> Lab "
Welcome to
OS
Lab
```

WHO:

```
[student@localhost ~]$ who
root          :0            Dec  18 00:16
itb1          pts/1         Dec  18 03:14 (172.16.4.10)
student       pts/2         Dec  18 03:21 (172.16.4.12)
itb4          pts/3         Dec  18 03:21 (172.16.4.39)
itb14         pts/4         Dec  18 03:22 (172.16.4.31)
itb2          pts/5         Dec  18 03:22 (172.16.4.42)
itb18         pts/8         Dec  18 03:23 (172.16.4.209)
itb20         pts/9         Dec  18 03:23 (172.16.4.30)
itb6          pts/11         Dec  1 03:23 (172.16.4.29)
itb9          pts/12        Dec  18 03:23 (172.16.4.43)
itb11         pts/10        Dec  18 03:23 (172.16.4.49)
itb12         pts/13        Dec  18 03:24 (172.16.4.48)
itb7          pts/14        Dec  18 03:24 (172.16.4.27)
itb3          pts/7         Dec  18 03:24 (172.16.4.26)
itb17         pts/15         Dec  18 03:24 (172.16.4.47)
itb13          pts/16        Dec  18 03:25 (172.16.4.46)
itb10         pts/17         Dec  18 03:26 (172.16.4.35)
itb16         pts/20        Dec  18 03:27 (172.16.4.34)
```

WHO AM I:

```
[student@localhost ~]$ who am i
student    pts/2        Dec  18 03:21 (172.16.4.12)

[student@localhost ~]$ who -b
         system boot  Dec  18 00:05
[student@localhost ~]$ who -H
NAME     LINE         TIME         COMMENT
root     :0           Dec  1 00:16
itb1     pts/1        Dec  18 03:14 (172.16.4.10)
student    pts/2        Dec  18 03:21 (172.16.4.12)
itb4     pts/3        Dec  18 03:21 (172.16.4.39)
```

```
itb14    pts/4        Dec  18 03:22 (172.16.4.31)
itb2     pts/5        Dec  18 03:22 (172.16.4.42)
itb18    pts/8        Dec  18 03:23 (172.16.4.209)
itb20    pts/9        Dec  18 03:23 (172.16.4.30)
itb6     pts/11       Dec  18 03:23 (172.16.4.29)
itb9     pts/12       Dec  18 03:23 (172.16.4.43)
itb11    pts/10       Dec  18 03:23 (172.16.4.49)
itb12    pts/13       Dec  18 03:24 (172.16.4.48)
itb7     pts/14       Dec  18 03:24 (172.16.4.27)
itb3     pts/7        Dec  18 03:24 (172.16.4.26)
itb17    pts/15       Dec  18 03:24 (172.16.4.47)
itb13    pts/16       Dec  18 03:25 (172.16.4.46)
itb10    pts/17       Dec  18 03:26 (172.16.4.35)
itb16    pts/20       Dec  18 03:27 (172.16.4.34)
itb8     pts/18       Dec  18 03:27 (172.16.4.33)
itb19    pts/19       Dec  18 03:33 (172.16.4.45)
itb1     pts/6        Dec  18 03:35 (172.16.4.32)
```

TTY :

```
[student@localhost ~]$ tty
/dev/pts/2
```

BINARY CALCULATOR :

```
[student@localhost ~]$ 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+8
13
6-2
4
7*4
28
sqrt(4)
2
Press CTRL+z:
```

CLEAR:

```
[student@localhost ~]$clear
```

MAN :

```
[student@localhost ~]$ man cal
CAL(1)                BSD General Commands Manual                CAL(1)
NAME
     cal - displays a calendar
SYNOPSIS
     cal [-smjy13] [[month] year]
DESCRIPTION
     Cal displays a simple calendar.  If arguments are not specified, the
     current month is displayed.  The options are as follows:
     -1      Display single month output.  (This is the default.)
     -3      Display prev/current/next month output.
     -s      Display Sunday as the first day of the week.  (This is the
             default.)
     -m      Display Monday as the first day of the week.
     -j      Display Julian dates (days one-based, numbered from January 1).
     -y      Display a calendar for the current year.
:          [2]+  Stopped                 man cal
```

LIST :

```
[student@localhost ~]$ ls
a.out  client2.c  greet.c  sclient.c  server2.c  socket.c  sserver.c
```

PRESENT WORKING DIRECTORY :

```
[student@localhost ~]$ pwd
/home/student
```

MKDIR :

```
[student@localhost ~]$ mkdir oslab
[student@localhost ~]$ ls
          oslab
[student@localhost oslab]$ pwd
/home/student/
```

CD :

```
[student@localhost ~]$ cd oslab
[student@localhost oslab]$ pwd
/home/student/oslab
[student@localhost dhanil]$ cd ..
```

```
[student@localhost oslab]$ pwd
/home/student/
```

RMDIR :

```
[student@localhost ~]$ rmdir oslab
[student@localhost ~]$ ls
```

CREATE A FILE :

```
[student@localhost ~]$ cat>osexamples
unix
linux
windows
android
macos
[3]+  Stopped                 cat > osexamples
[student@localhost ~]$ ls
          oslab   osexamples
```

DISPLAY THE CONTENTS OF THE FILE :

```
[student@localhost ~]$ cat osexamples
unix
linux
windows
android
macos
[student@localhost ~]$ tac examplesofos
macos
android
windows
linux
unix
```

SORT:

```
[student@localhost ~]$ sort osexamples
android
linux
macos
unix
windows
```

```
[student@localhost ~]$ sort -r osexamples
windows
unix
macos
linux
android
```

COPYING CONTENTS FROM ONE FILE TO ANOTHER :

```
[student@localhost ~]$ cp osexamples examplesofos
[student@localhost ~]$ cat osexamples
unix
linux
windows
android
macos
[student@localhost ~]$ cat examplesofos
unix
linux
windows
android
macos
```

MOVING THE CONTENTS FROM ONE FILE TO ANOTHER :

```
[student@localhost ~]$ mv osexamples examplesofos
[student@localhost ~]$ cat osexamples
No such file or directory
[student@localhost ~]$ cat examplesofos
unix
linux
windows
android
macos
```

REMOVE :

```
[student@localhost ~]$ cp examplesofos osexamples
[student@localhost ~]$ ls
          oslab   examplesofos  osexamples
 [student@localhost ~]$ rm osexamples
 [student@localhost ~]$ ls
          oslab   examplesofos
```

WORD COUNT:

```
[student@localhost ~]$ wc examplesofos
5        5        28  examplesofos
[student@localhost ~]$ wc -l examplesofos
5 college
[student@localhost ~]$ wc -c examplesofos
28 college
[student@localhost ~]$ wc -w examplesofos
5 college
```

DISK FREE:

```
[student@localhost ~]$ df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
                      33124152   6855516  24586000  22% /
/dev/sda1               101086     12511     83356  14% /boot
none                    517336         0    517336   0% /dev/shm
```

DIFF:

```
[student@localhost ~]$ cat>ostypes
android
unix
windows
[student@localhost ~]$ cat>typesos
linux
windows
[student@localhost ~]$ diff ostypes typesos
1,2c1
< android
< unix
---
> linux
```

COMM:

```
[student@localhost ~]$ comm ostypes typesos
android
                linux
unix
                                windows
```

CMP:

```
[student@localhost ~]$ cmp ostypes typesos
ostypes typesos differ: char 1, line 1
```

UNIQ:

```
[student@localhost ~]$cat>os
linux
linux
unix
windows
android
android
android
[student@localhost ~]$cat os
linux
linux
unix
windows
android
android
android
[student@localhost ~]$uniq os
linux
unix
windows
android
[student@localhost ~]$uniq -c os
      2 linux
      1 unix
      1 windows
      3 android

[student@localhost ~]$uniq -u os
unix
windows
```

TOP:

```
linuxpert@localhost ~]$ top

top - 21:48:14 up 11 min,  2 users,  load average: 0.12, 0.25, 0.30
Tasks: 158 total,   1 running, 157 sleeping,   0 stopped,   0 zombie
Cpu(s):  2.0%us,  3.0%sy,  4.0%ni, 87.7%id,  3.3%wa,  0.0%hi,  0.0%si,  0.0%
```

```
Mem:   1018852k total,   526724k used,   492128k free,    56840k buffers
Swap:  1047544k total,        0k used,  1047544k free,   281676k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3208 linuxper  39  19 46612  16m 9452 S  5.3  1.6   0:01.04 beagled-helper
 3010 linuxper  20   0  178m 9.9m 7664 S  1.7  1.0   0:02.15 lxterminal
 2271 root      20   0 36736  13m 8520 S  1.3  1.4   0:13.82 Xorg
 2610 linuxper  20   0  214m  18m  14m S  0.7  1.9   0:03.16 nautilus
 2639 linuxper  27   7 54120  17m 9.8m S  0.7  1.8   0:01.40 beagled
 2991 linuxper  20   0  227m  25m  16m S  0.3  2.6   0:12.05 abiword
 3220 linuxper  20   0  2700 1120  852 R  0.3  0.1   0:00.02 top
    1 root      20   0  2828 1352 1148 S  0.0  0.1   0:01.11 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    6 root      20   0     0    0    0 S  0.0  0.0   0:00.01 events/0
    7 root      20   0     0    0    0 S  0.0  0.0   0:00.00 cpuset
    8 root      20   0     0    0    0 S  0.0  0.0   0:00.00 khelper
    9 root      20   0     0    0    0 S  0.0  0.0   0:00.00 netns
   10 root      20   0     0    0    0 S  0.0  0.0   0:00.00 async/mgr
```

HEAD:

```
[student@localhost ~]$ head examplesofos
unix
linux
windows
android
macos
[student@localhost ~]$ head -2 examplesofos
unix
linux
```

TAIL:

```
[student@localhost ~]$ tail examplesofos
unix
linux
windows
android
macos
[student@localhost ~]$ tail -1 examplesofos
```

```
macos
```

GREP :

```
[student@localhost ~]$ grep li examplesofos
linux
[student@localhost ~]$ grep "an" examplesofos
android
[student@localhost ~]$ grep "^wi" examplesofos
windows
```

EGREP :

```
[student@localhost ~]$ egrep "un|an|mc" examplesofos
unix
android
macos
```

FGREP : fgrep "un

```
>li
>mc" examplesofos
unix
linux
macos
```

COMPRESS:

```
[student@localhost ~]$ gzip examplesofos
[student@localhost ~]$ ls
        oslab   examplesofos.gz
```

DECOMPRESS:

```
[student@localhost ~]$ gunzip examplesofos
[student@localhost ~]$ ls
        oslab   examplesofos
```

PIPE :

```
[student@localhost ~]$ who | wc-l
15
```

CUT:

```
[student@localhost ~]$ cut  -c1 examplesofos
```

```
u
l
w
a
m
[student@localhost ~]$ cut  -c1,3 examplesofos
ui
ln
wn
ad
mc
```

PASTE:

```
[student@localhost ~]$ cat line
1
2
3
4
5

[student@localhost ~]$ cat osexamp
unix
linux
windows
android
macos

[student@localhost ~]$ paste line osexamp
        1       unix
        2       linux
        3       windows
        4       android
        5       macos
```

CHMOD:

```
[student@localhost ~]$ ls -l examplesofos
-rw-r--r--  1 student users 18 Jan  1 06:12 examplesofos
[student@localhost ~]$ chmod 777 examplesofos
-rwxrwxrwx  1 student users 18 Jan  1 06:12 examplesofos
[student@localhost ~]$ chmod 444 examplesofos
```

```
-rw-rw-rw-  1 student users 18 Jan  1 06:12 examplesofos
[student@localhost ~]$ chmod 111 examplesofos
-r--r--r--  1 student users 18 Jan  1 06:12 examplesofos
[student@localhost ~]$ chmod 000 examplesofos
----------  1 student users 18 Jan  1 06:12 examplesofos
```

LINK:

```
[student@localhost ~]$ ln examplesofos  examplesos
[student@localhost ~]$ cat examplesofos
unix
linux
windows
android
macos
[student@localhost ~]$ cat examplesos
unix
linux
windows
android
macos
```

RESULT:

Thus the basic unix commands has been executed successfully and the output is verified.

EX.NO:2A FIRST COME FIRST SERVED (FCFS) SCHEDULING ALGORITHM

AIM:

To write a C program to implement First Come First Served (FCFS) Scheduling Algorithm.

ALGORITHM:

Step 1: Start the program.
Step 2: Read the total number of processes.
Step 3: Read the process id and burst time for each process.
Step 4: Initially, Waiting time of first process is zero and Total time for the first process is the starting time of that process.
Step 5: Calculate waiting time for remaining process by adding burst time of that process with waiting time of previous process.
Step 6: Calculate Turnaround time for the process by adding waiting time with burst time of that process.
Step 7: Calculate Average waiting time by dividing the total waiting time by total number of process.
Step 8: Calculate Average turnaround time by dividing the total turnaround time by the number of process.
Step 9: Display the result.
Step 10: Stop the program.

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 Time\n");
for(i=0;i<n;i++)
{
printf("P[%d]:",i+1);
scanf("%d",&bt[i]);
}
wt[0]=0;    //waiting time for first process is 0
//calculating waiting time
for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];
}
printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");
//calculating turnaround time
for(i=0;i<n;i++)
{
tat[i]=bt[i]+wt[i];
avwt+=wt[i];
avtat+=tat[i];
printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
}
avwt/=i;
avtat/=i;
printf("\n\nAverage Waiting Time:%d",avwt);
printf("\nAverage Turnaround Time:%d",avtat);
return 0;
}
```

OUTPUT:

```
[it@localhost ~]$ cc fcfs.c
[it@localhost ~]$ ./a.out
Enter total number of processes(maximum 20):3
Enter Process Burst Time
P[1]:24
P[2]:3
P[3]:3
Process         Burst Time      Waiting Time    Turnaround Time
P[1]              24              0                     24
P[2]              3               24                    27
P[3]              3               27                    30
Average Waiting Time:17
Average Turnaround Time:27
```

RESULT:

Thus the C program to implement First Come First Served (FCFS) Scheduling algorithm has been executed successfully and the output is verified.

EX.NO:2B SHORTEST JOB FIRST (SJF) SCHEDULING ALGORITHM

AIM:

To write a C program to implement Shortest Job First scheduling algorithm.

ALGORITHM:

Step 1: Start the program.
Step 2: Read the total number of processes.
Step 3: Read the process id and burst time for each process.
Step 4: Perform selection sort on burst time in ascending order.
Step 5: Initially, Waiting time of first process is zero and Total time for the first process is the starting time of that process.
Step 6: Calculate waiting time for remaining process by adding burst time of that process with waiting time of previous process.
Step 7: Calculate Average waiting time by dividing the total waiting time by total number of process.
Step 8: Calculate Turnaround time for the process by adding waiting time with burst time of that process.
Step 9: Calculate Average turnaround time by dividing the total turnaround time by the number of process.
Step 9: Display the result.
Step 10:Stop the program.

PROGRAM:

```
#include<stdio.h>
int main()
{
int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
float avg_wt,avg_tat;
printf("Enter number of process:");
scanf("%d",&n);
printf("\nEnter Burst Time:\n");
for(i=0;i<n;i++)
{
printf("p%d:",i+1);
scanf("%d",&bt[i]);
p[i]=i+1;           //contains process number
}
//sorting burst time in ascending order using selection sort
for(i=0;i<n;i++)
{
pos=i;
for(j=i+1;j<n;j++)
{
if(bt[j]<bt[pos])
pos=j;
}
temp=bt[i];
bt[i]=bt[pos];
bt[pos]=temp;
temp=p[i];
p[i]=p[pos];
p[pos]=temp;
}
wt[0]=0;            //waiting time for first process will be zero
//calculate waiting time
for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];
total+=wt[i];
}
avg_wt=(float)total/n;      //average waiting time
total=0;
printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
tat[i]=bt[i]+wt[i];     //calculate turnaround time
total+=tat[i];
printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
}
avg_tat=(float)total/n;     //average turnaround time
printf("\n\nAverage Waiting Time=%f",avg_wt);
printf("\nAverage Turnaround Time=%f\n",avg_tat);
return 0;
}
```

OUTPUT:

```
[ita60@localhost ~]$ cc sjf.c
[ita60@localhost ~]$ ./a.out
Enter number of process:4

Enter Burst Time:
p1:4
p2:8
p3:3
p4:7
```

```
Process     Burst Time          Waiting Time    Turnaround Time
p3             3                     0                      3
p1             4                    3                        7
p4             7                    7                       14
p2             8                    14                      22

Average Waiting Time=6.000000
Average Turnaround Time=11.500000
```

RESULT:

Thus the C program to implement Shortest Job First (SJF) Scheduling algorithm has been executed successfully and the output is verified.

EX.NO:2C PRIORITY SCHEDULING ALGORITHM

AIM:

To write a C program to implement Priority Scheduling algorithm.

ALGORITHM:

Step 1: Start the program.

Step 2: Read the total number of processes.

Step 3: Read the process id, burst time and priority for each process.

Step 4: Perform selection sort on process id and burst time based on priority in ascending order.

Step 5: Initially, Waiting time of first process is zero and Total time for the first process is the starting time of that process.

Step 6: Calculate waiting time for remaining process by adding burst time of that process with waiting time of previous process.

Step 7: Calculate Average waiting time by dividing the total waiting time by total number of process.

Step 8: Calculate Turnaround time for the process by adding waiting time with burst time of that process.

Step 9: Calculate Average turnaround time by dividing the total turnaround time by the number of process.

Step 9: Display the result.

Step 10: Stop the program

PROGRAM:

```
#include<stdio.h>
int main()
{
int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
printf("Enter Total Number of Process:");
scanf("%d",&n);
printf("\nEnter Burst Time and Priority\n");
for(i=0;i<n;i++)
{
printf("\nP[%d]\n",i+1);
printf("Burst Time:");
scanf("%d",&bt[i]);
printf("Priority:");
scanf("%d",&pr[i]);
p[i]=i+1;           //contains process number
}
//sorting burst time, priority and process number in ascending order using selection sort
for(i=0;i<n;i++)
{
pos=i;
for(j=i+1;j<n;j++)
{
if(pr[j]<pr[pos])
pos=j;
}
temp=pr[i];
pr[i]=pr[pos];
pr[pos]=temp;
temp=bt[i];
bt[i]=bt[pos];
bt[pos]=temp;
temp=p[i];
p[i]=p[pos];
p[pos]=temp;
}
wt[0]=0;    //waiting time for first process is zero
//calculate waiting time
for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];
total+=wt[i];
}
avg_wt=total/n;      //average waiting time
total=0;
printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
tat[i]=bt[i]+wt[i];     //calculate turnaround time
total+=tat[i];
printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
}
avg_tat=total/n;     //average turnaround time
printf("\n\nAverage Waiting Time=%d",avg_wt);
printf("\nAverage Turnaround Time=%d\n",avg_tat);
return 0;
}
```

OUTPUT:

```
[it@localhost ~]$ cc priority.c
```

```
[it@localhost ~]$ ./a.out
Enter Total Number of Process:4
Enter Burst Time and Priority
P[1]
Burst Time:6
Priority:3
P[2]
Burst Time:2
Priority:2
P[3]
Burst Time:14
Priority:1
P[4]
Burst Time:6
Priority:4
Process     Burst Time          Waiting Time    Turnaround Time
P[3]           14                    0                      14
P[2]           2                    14                      16
P[1]           6                    16                      22
P[4]           6                    22                      28
Average Waiting Time=13
Average Turnaround Time=20
```

RESULT:

Thus the C program to implement Priority Scheduling algorithm has been executed successfully and the output is verified.

EX.NO:2D ROUND ROBIN SCHEDULING ALGORITHM

AIM:

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

ALGORITHM:

Step 1: Start the program.
Step 2: Read the total number of processes.
Step 3: Read the arrival time and burst time for each process.
Step 4: Read the time quantum.
Step 5:Initially, Waiting time of first process is zero and Total time for the first process is the starting time of that process.
Step 6: Calculate waiting time for remaining process by adding burst time of that process with waiting time of previous process.
Step 7: Calculate Average waiting time by dividing the total waiting time by total number of process.
Step 8: Calculate Turnaround time for the process by adding waiting time with burst time of that process.
Step 9: Calculate Average turnaround time by dividing the total turnaround time by the number of process.
Step 9: Display the result.
Step 10: Stop the program

PROGRAM:

```
#include<stdio.h>
int main()
{
int count,j,n,time,remain,flag=0,time_quantum;
int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
printf("Enter Total Process:\t ");
scanf("%d",&n);
remain=n;
for(count=0;count<n;count++)
{
printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);
scanf("%d",&at[count]);
scanf("%d",&bt[count]);
rt[count]=bt[count];
}
printf("Enter Time Quantum:\t");
scanf("%d",&time_quantum);
printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
for(time=0,count=0;remain!=0;)
{
if(rt[count]<=time_quantum && rt[count]>0)
{
```

```
time+=rt[count];
rt[count]=0;
flag=1;
}
else if(rt[count]>0)
{
rt[count]-=time_quantum;
time+=time_quantum;
}
if(rt[count]==0 && flag==1)
{
remain--;
printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);
wait_time+=time-at[count]-bt[count];
turnaround_time+=time-at[count];
flag=0;
}
if(count==n-1)
count=0;
else if(at[count+1]<=time)
count++;
else
count=0;
}
printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
printf("Average Turnaround Time = %f",turnaround_time*1.0/n);
return 0;
}
```

OUTPUT:

```
[it@localhost ~]$ cc roundrobin.c
[it@localhost ~]$ ./a.out
Enter Total Process:     4
Enter Arrival Time and Burst Time for Process Number 1 :   0  9
Enter Arrival Time and Burst Time for Process Number 2 :   1  5
Enter Arrival Time and Burst Time for Process Number 3 :   2  3
Enter Arrival Time and Burst Time for Process Number 4 :   3  4
Enter Time Quantum:     5
Process |Turnaround Time      |Waiting Time

P[2]    |          9                   |       4
P[3]    |          11                  |       8
```

```
P[4]    |           14                  |       10
P[1]    |           21                  |       12

Average Waiting Time= 8.500000
Average Turnaround Time = 13.750000
```

RESULT:

Thus the C program to implement Round Robin Scheduling algorithm has been executed successfully and the output is verified.

EX.NO:3A FILE ALLOCATION STRATEGIES

SEQUENTIAL FILE ALLOCATION METHOD

AIM:

To write a c program to implement file allocation strategies using sequential file allocation method.

ALGORITHM:

```
Step 1 : Start the program.
Step 2 : Read the number of files
Step 3: For each file, read the number of blocks required and the starting block of the file.
 Step 4: Allocate the blocks sequentially to the file from the starting block.
 Step 5: Display the file name, starting block, and the blocks occupied by the file.
Step 6: Stop the program.
```

PROGRAM:

```
#include<stdio.h>
int main()
{
int n,i,j,b[20],sb[20],t[20],x,c[20][20];
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("blocks occupiedare:");
 for(i=0;i<n;i++)
 { printf("fileno%d",i+1);
 for(j=0;j<b[i];j++)
 printf("\t%d",c[i][j]);
 printf("\n");
 }
return 0;
}
```

OUTPUT:

```
Enter no.of files:2
Enter no. of blocks occupied by file14
Enter the starting block of file12
Enter no. of blocks occupied by file26
Enter the starting block of file2 8
Filename        Start block     length
1                    2              4
2                    8                6
b\locks occupied are:
fileno1      2       3       4       5
fileno2      8       9       10      11      12      13
```

RESULT:

Thus the c program to write a c program to implement file allocation strategies using sequential file allocation method has been executed successfully and the output is verified.

EX.NO:3B **INDEXED FILE ALLOCATION METHOD**

AIM:

To write a c program to implement file allocation strategies using indexed file allocation method.

ALGORITHM :

Step 1: Start the program
Step 2: Read the number of files
Step 3: Read the index block for each file.
Step 4: For each file, read the number of blocks occupied and number of blocks of the file.
Step 5: Link all the blocks of the file to the index block.
Step 6: Display the file name, index block, and the blocks occupied by the file.
Step 7: Stop the program.

PROGRAM:

```
#include<stdio.h>
int main()
{
int n,m[20],i,j,ib[20],b[20][20];
printf("Enter no. of files:");
scanf("%d",&n);
for(i=0;i<n;i++)
{ printf("Enter index block :",i+1);
scanf("%d",&ib[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,ib[i],m[i]);
printf("Blocks occupiedare:");
 for(i=0;i<n;i++)
 { printf("fileno%d",i+1);
 for(j=0;j<m[i];j++)
 printf("\t%d--->%d\n",ib[i],b[i][j]);
 printf("\n");
 }
return 0;
}
```

OUTPUT:

```
Enter no. of files:2
 Enter index block 3
Enter blocks occupied by file1: 4
Enter blocks of file1:9
4
 6
 7
Enter index block 5
Enter blocks occupied by file2:2
Enter blocks of file2: 10
8
File    index    length
1         3         4
2         5         2
 Blocks occupied are:
 file1
3--->9
 3--->4
3--->6
3--->7
 file2
5--->10
 5--->8
```

RESULT:

Thus the c program to write a c program to implement file allocation strategies using indexed file allocation method has been executed successfully and the output is verified.

EX.NO:3C LINKED FILE ALLOCATION METHOD

AIM:
To write a c program to implement file allocation strategies using linked file allocation method.
ALGORITHM:

```
Step 1: Start the program.
Step 2: Read the number of files
Step 3: For each file, read the file name, starting block, number of blocks and block numbers of
        the file.
 Step 4: Start from the starting block and link each block of the file to the next block in a linked
        list fashion.
 Step 5: Display the file name, starting block, size of the file, and the blocks occupied by the file
 Step 6: Stop the program.
```

PROGRAM:

```
#include<stdio.h>
struct file
{
char fname[10];
int start,size,block[10];
}f[10];
int main()
{
int i,j,n;
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=0;j<f[i].size;j++)
printf("%d--->",f[i].block[j]);
printf("%d",f[i].block[j]);
printf("\n");
}
return 0;
}
```

```
Enter no. of files:2
Enter file name: Lavanya
Enter starting block:20
 Enter no.of blocks:6
Enter block numbers: 4
 12
15
45
32
25
Enter file name: Priya
Enter starting block:12
Enter no. of blocks:5
Enter block numbers:6
5
 4
3
 2
File    start   size    block
Lavanya         20      6       20--->4--->12--->15--->45--->32--->25
 Priya  12      5       12--->6--->5--->4--->3--->2
```

RESULT:

Thus the c program to write a c program to implement file allocation strategies using linked file allocation method has been executed successfully and the output is verified.

EX.NO:4 IMPLEMENT SEMAPHORE USING PRODUCER CONSUMER PROBLEM

AIM:

To write a c program to implement semaphore using producer consumer problem.

ALGORITHM:

```
Step 1: Start the program.
Step 2: The Semaphore mutex, full & empty are initialized.
Step 3: 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.
Step 4: 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.
Step 5: Print the result.
Step 6: Stop the program.
```

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
int main()
{
int n;
void producer();
void consumer();
int wait(int);
int signal(int);
printf("\n1.Producer\n2.Consumer\n3.Exit");
while(1)
{
printf("\nEnter your choice:");
scanf("%d",&n);
switch(n)
{
case 1:
 if((mutex==1)&&(empty!=0))
producer();
else
printf("Buffer is full!!");
break;
case 2:
if((mutex==1)&&(full!=0))
consumer();
else
printf("Buffer is empty!!");
break;
case 3:
exit(0);
break;
}
}
return 0;
}
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("\nProducer produces the item %d",x);
mutex=signal(mutex);
}
void consumer()
{
mutex=wait(mutex);
full=wait(full);
empty=signal(empty);
printf("\nConsumer consumes item %d",x);
x--;
mutex=signal(mutex);
```

```
}
```

OUTPUT:

```
[ita60@localhost ~]$ cc produc.c
[ita60@localhost ~]$ ./a.out
1.Producer
2.Consumer
3.Exit
Enter your choice:1
Producer produces the item 1
Enter your choice:2
Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:1
Producer produces the item 1
Enter your choice:1
Producer produces the item 2
Enter your choice:1
Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:2
Consumer consumes item 3
Enter your choice:3
```

RESULT:

Thus the c program to implement semaphore using producer consumer problem has been executed successfully and the output is verified.

EX.NO:5A FILE ORGANIZATION TECHNIQUES

SINGLE LEVEL DIRECTORY

AIM:

To write a c program to implement file organization techniques for single level directory.

ALGORITHM:

```
Step 1: Start the program.
Step 2: Read number of directories.
Step 3:.Read the name of directories.
Step 4: Read the size of the directories.
Step 5: Create a file under the directory.
Step 6: Display the single level directory name, size and file.
Step 7: Stop the program.
```

PROGRAM:

```
#include<stdio.h>
int main()
{
int masters,s[20];
char f[20][20][20];
char d[20][20];
int i,j;
printf("Enter number of directories:");
scanf("%d",&masters);
printf("Enter name of directories:");
for(i=0;i<masters;i++)
scanf("%s",&d[i]);
printf("Enter the size of directories:");
for(i=0;i<masters;i++)
scanf("%d",&s[i]);
printf("Enter the file name:");
for(i=0;i<masters;i++)
for(j=0;j<s[i];j++)
scanf("%s",&f[i][j]);
printf("\n");
printf("Directory \t Size \t File Names \n");
printf("\n");
for(i=0;i<masters;i++)
{
printf("%s \t\t%d \t",d[i],s[i]);
for(j=0;j<s[i];j++)
printf("%s\n\t\t\t",f[i][j]);
printf("\n");
}
printf("\t\n");
}
```

OUTPUT:

```
[ita60@localhost ~]$ cc single.c
[ita60@localhost ~]$ ./a.out
Enter number of directories:1
Enter name of directories:X.c
Enter the size of directories:2
Enter the file name:
aaa
bbb
Directory         Size     File names
X.c               2         aaa
                            bbb
```

RESULT:

Thus the c program to implement file organization techniques for single level directory has been executed successfully and the output is verified.

EX.NO:5B **FILE ORGANIZATION TECHNIQUES**

TWO LEVEL DIRECTORY

AIM:

To write a c program to implement file organization techniques for two level directory.

ALGORITHM:

Step 1: Start the program.
Step 2: Read number of directories.
Step 3:.Read the name of directories.
Step 4: Read the size of the directories.
Step 5: Read the subdirectory name and size.
Step 6: Create a file under the subdirectory.
Step 7: Display the two level directory name, size and file.
Step 8: Stop the program

PROGRAM:

```
#include<stdio.h>
struct st
{
char dname[10];
char sdname[10][10];
char fname[10][10][10];
int ds,sds[10];
}dir[10];
int main()
{
int i,j,k,n;
printf("Enter number of directories:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter directory %d names:",i+1);
scanf("%s",&dir[i].dname);
printf("Enter size of directories:");
scanf("%d",&dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("Enter subdirectory name and size:");
scanf("%s",&dir[i].sdname[j]);
scanf("%d",&dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
{
printf("Enter file name:");
scanf("%s",&dir[i].fname[j][k]);
}
}
}
printf("\nDirname\t\tsize\tsubdirname\tsize\tfiles");
printf("\n******************************************************\n");
for(i=0;i<n;i++)
{
printf("%s\t\t%d",dir[i].dname,dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("\t%s\t\t%d\t",dir[i].sdname[j],dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
printf("%s\t",dir[i].fname[j][k]);
printf("\n\t\t");
}
printf("\n");
}
return 0;
 }
```

OUTPUT:

```
[ita60@localhost ~]$ cc two.c
[ita60@localhost ~]$ ./a.out
Enter number of directories:2
Enter directory 1 names:x.c
Enter size of directories:2
Enter subdirectory name and size:aaa 1
Enter file name:a.c
Enter subdirectory name and size:bbb 1
Enter file name:b.c
Enter directory 2 names:Y.c
Enter size of directories:1
Enter subdirectory name and size:ccc 1
Enter file name:d.c

Dirname        Size    Subdirname      Size    Files
******************************************************
X.c             2        aaa             1      a.c
                         bbb             1      b.c

Y.c             1        ccc             1      d.c
```

RESULT:
Thus the c program to implement file organization techniques for two level directory has been executed successfully and the output is verified.

EX.NO:6A IMPLEMENT BANKERS ALGORITHM FOR DEADLOCK DETECTION

AIM:

To write a c program to implement banker's algorithm for deadlock detection..

ALGORITHM

```
Step 1: Start the program.
Step 2: Read the number of processes and   resource instances.
Step 3: Read the Max Matrix, Allocation Matrix and available resources.
Step 4: Calculate Need Matrix using   Need=Max-Allocation
Step 5: If (Finish[i]==False and Need<=available )then
                Available=available + allocation, Finish[i]=true and repeat the step 5.
        Else go to step 6.
Step 6: If Finish[i] ==false for some i,0<=i<n, then print  the system is in deadlocked state and
        then print process Pi is deadlocked else print no deadlock occur.
Step 7: Stop the program.
```

PROGRAM:

```
#include<stdio.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void input();
void show();
void cal();
int main()
{
     int i,j;
     printf("********** Deadlock Detection Algorithm ************\n");
     input();
     show();
     cal();          return 0;
}
void input()
{
          int i,j;
          printf("Enter the no of Processes\t");
          scanf("%d",&n);
          printf("Enter the no of resource instances\t");
          scanf("%d",&r);
          printf("Enter the Max Matrix\n");
          for(i=0;i<n;i++)
```

```
		{
			for(j=0;j<r;j++)
			{
				scanf("%d",&max[i][j]);
			}
		}
		printf("Enter the Allocation Matrix\n");
		for(i=0;i<n;i++)
		{
			for(j=0;j<r;j++)
			{
				scanf("%d",&alloc[i][j]);
			}
		}
		printf("Enter the available Resources\n");
		for(j=0;j<r;j++)
		{
			scanf("%d",&avail[j]);
		}
	}
	void show()
	{
		int i,j;
		printf("Process\t Allocation\t Max\t Available\t");
		for(i=0;i<n;i++)
		{
			printf("\nP%d\t   ",i+1);
			for(j=0;j<r;j++)
			{
				printf("%d ",alloc[i][j]);
			}
			printf("\t");
			for(j=0;j<r;j++)
			{
				printf("%d ",max[i][j]);
			}
			printf("\t");
			if(i==0)
			{
				for(j=0;j<r;j++)
				printf("%d ",avail[j]);
```

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

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

OUTPUT:

```
[it@localhost ~]$ cc detect.c
[it@localhost ~]$ ./a.out
********** Deadlock Detection Algorithm ************
Enter the no of Processes       3
Enter the no of resource instances      3
Enter the Max Matrix
3 6 8
4 3 3
3 4 4
Enter the Allocation Matrix
3 3 3
2 0 3
1 2 4
Enter the available Resources
1 2 0
Process         Allocation          Max     Available
P1                 3 3 3            3 6 8      1 2 0
P2                 2 0 3            4 3 3
P3                 1 2 4            3 4 4
System is in Deadlock and the Deadlock processes are
P0      P1      P2
```

RESULT:

Thus the c program to implement banker's algorithm for deadlock detection has been executed successfully and the output is verified.

EX.NO:6B IMPLEMENT BANKERS ALGORITHM FOR DEADLOCK AVOIDANCE

AIM:

To write a c program to implement banker's algorithm for deadlock avoidance.

ALGORITHM

```
Step 1: Start the program.
Step 2: Read the number of processes and   resource instances.
Step 3: Read the Max Matrix, Allocation Matrix and available resources.
Step 4: Calculate Need Matrix using   Need=Max-Allocation
Step 5: If (Finish[i]==False and Need<=available )then
                Available=available + allocation, Finish[i]=true and repeat the step 5.
        Else go to step 6.
Step 6: If Finish[i] =true for all i,0<=i<n, then print the processes in safe sequence and  the
        system is in safe state  else print the system in unsafe state.
Step 7: Stop the program.
```

PROGRAM:

```
#include<stdio.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void input();
void show();
void cal();
int main()
{
int i,j;
printf("********** Banker's Algorithm ************\n");
input();
show();
cal();
return 0;
}
void input()
{
int i,j;
printf("Enter the no of Processes\t");
scanf("%d",&n);
printf("Enter the no of resources instances\t");
scanf("%d",&r);
printf("Enter the Max Matrix\n");
```

```
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
scanf("%d",&max[i][j]);
}
}
printf("Enter the Allocation Matrix\n");
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
scanf("%d",&alloc[i][j]);
}
}
printf("Enter the available Resources\n");
for(j=0;j<r;j++)
{
scanf("%d",&avail[j]);
}
}
void show()
{
int i,j;
printf("Process\t Allocation\t Max\t Available\t");
for(i=0;i<n;i++)
{
printf("\nP%d\t   ",i+1);
for(j=0;j<r;j++)
{
printf("%d ",alloc[i][j]);
}
printf("\t");
for(j=0;j<r;j++)
{
printf("%d ",max[i][j]);
}
printf("\t");
if(i==0)
{
for(j=0;j<r;j++)
```

```
printf("%d ",avail[j]);
}
}
}
void cal()
{
int finish[100],temp,need[100][100],flag=1,k,c1=0;
int safe[100];
int i,j;
for(i=0;i<n;i++)
{
finish[i]=0;
}
//find need matrix
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
need[i][j]=max[i][j]-alloc[i][j];
}
}
printf("\n");
while(flag)
{
flag=0;
for(i=0;i<n;i++)
{
int c=0;
for(j=0;j<r;j++)
{
if((finish[i]==0)&&(need[i][j]<=avail[j]))
{
c++;
if(c==r)
{
for(k=0;k<r;k++)
{
avail[k]+=alloc[i][j];
finish[i]=1;
flag=1;
}
printf("P%d->",i);
if(finish[i]==1)
{
i=n;
}
}
}
}
}
}
for(i=0;i<n;i++)
{
if(finish[i]==1)
{
c1++;
}
else
{
printf("P%d->",i);
}
}
if(c1==n)
{
printf("\n The system is in safe state");
}
else
{
printf("\n Process are in dead lock");
printf("\n System is in unsafe state");
}
}
```

OUTPUT:

```
[it@localhost ~]$ vi avoidance.c
[it@localhost ~]$ cc avoidance.c
[it@localhost ~]$ ./a.out
********** Banker's Algorithm ************
Enter the no of Processes       5
Enter the no of resources instances     3
Enter the Max Matrix
7  5  3
3  2  2
9  0  2
2  2  2
4  3  3
Enter the Allocation Matrix
0  1  0
2  0  0
3  0  2
2  1  1
0  0  2
Enter the available Resources
3  3  2
Process         Allocation      Max     Available
P1              0 1 0           7 5 3    3 3 2
P2              2 0 0           3 2 2
P3              3 0 2           9 0 2
P4              2 1 1           2 2 2
P5              0 0 2           4 3 3
P1->P3->P4->P2->P0->
 The system is in safe state
```

RESULT:

Thus the c program to implement banker's algorithm for deadlock avoidance has been executed successfully and the output is verified.

EX.NO:7A IMPLEMENT FIFO PAGE REPLACEMENT ALGORITHM

AIM:

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

ALGORITHM

Step 1: Start the program.
Step 2: Read the length of the reference string (number of pages).
Step 3: Read the reference string (page number).
Step 4: Read the no of frames.
Step 5: Initialize the values in frames to -1.
Step 6: Allocate the pages in to frames in First in first out order.
Step 7: Display the number of page faults.
Step 8:Stop the program.

PROGRAM:

```
#include<stdio.h>
int main()
{
int i, j, k, f, pf=0, count=0, rs[25], m[10], n;
printf("\n Enter the length of reference string -- ");
scanf("%d",&n);
printf("\n Enter the reference string -- ");
for(i=0;i<n;i++)
scanf("%d",&rs[i]);
printf("\n Enter no. of frames -- ");
scanf("%d",&f);
for(i=0;i<f;i++)
m[i]=-1;
printf("\n The FIFO Page Replacement Process is -- \n");
for(i=0;i<n;i++)
{
for(k=0;k<f;k++)
{
if(m[k]==rs[i])
break;
} if(k==f)
{
m[count++]=rs[i];
pf++;
}
for(j=0;j<f;j++)
printf("\t%d",m[j]);
if(k==f)
printf("\tPF No. %d",pf);
printf("\n");
if(count==f)
count=0;
}
printf("\n The number of Page Faults using FIFO are %d",pf);
return 0;
}
```

OUTPUT:

```
 [it@localhost ~]$ cc fifo.c
[it@localhost ~]$ ./a.out
 Enter the length of reference string -- 20
 Enter the reference string -- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
 Enter no. of frames -- 3
 The FIFO Page Replacement Process is --
        7       -1      -1      PF No. 1
        7       0       -1      PF No. 2
        7       0       1       PF No. 3
        2       0       1       PF No. 4
        2       0       1
        2       3       1       PF No. 5
        2       3       0       PF No. 6
        4       3       0       PF No. 7
        4       2       0       PF No. 8
        4       2       3       PF No. 9
        0       2       3       PF No. 10
        0       2       3
        0       2       3
        0       1       3       PF No. 11
        0       1       2       PF No. 12
        0       1       2
        0       1       2
        7       1       2       PF No. 13
        7       0       2       PF No. 14
        7       0       1       PF No. 15
The numbers of Page Faults using FIFO are 15
```

RESULT

Thus the c program to implement FIFO page replacement algorithm has been executed successfully and the output is verified.

EX.NO:7B IMPLEMENT LRU PAGE REPLACEMENT ALGORITHM

AIM:

To write a c program to implement LRU page replacement algorithm.

ALGORITHM

Step 1: Start the program.
Step 2: Read the length of the reference string (number of pages).
Step 3: Read the reference string (page number).
Step 4: Read the no of frames.
Step 5: Initialize the values in frames to -1.
Step 6: Allocate the pages in to frames by selecting the page that has not been used for the longest period of time.
Step 7: Display the number of page faults.
Step 8: Stop the program.

PROGRAM:

```
#include<stdio.h>
int main()
{
int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0, next=1;
printf("Enter the length of reference string -- ");
scanf("%d",&n); printf("Enter the reference string -- ");
for(i=0;i<n;i++)
{
scanf("%d",&rs[i]); flag[i]=0;
}
printf("Enter the number of frames -- ");
scanf("%d",&f);
for(i=0;i<f;i++)
{
count[i]=0;
m[i]=-1;
}
printf("\nThe Page Replacement process is -- \n");
for(i=0;i<n;i++)
{
for(j=0;j<f;j++)
{
if(m[j]==rs[i])
{
flag[i]=1;
count[j]=next;
next++;
}
}
if(flag[i]==0)
{
if(i<f)
{
m[i]=rs[i];
count[i]=next;
next++;
}
else
{
min=0;
for(j=1;j<f;j++)
if(count[min] > count[j])
min=j;
m[min]=rs[i];
count[min]=next;
next++;
}
pf++;
}
for(j=0;j<f;j++)
printf("%d\t", m[j]);
if(flag[i]==0)
printf("PF No. -- %d" , pf);
printf("\n");
}
printf("\nThe number of page faults using LRU are %d",pf);
return 0;
}
```

OUTPUT:

```
[it@localhost ~]$ vi lru.c
[it@localhost ~]$ cc lru.c
[it@localhost ~]$ ./a.out
Enter the length of reference string -- 8
Enter the reference string -- 1 2 3 4 5 5 4 3
Enter the number 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
4      5      3      PF No. -- 5
4      5      3
4      5      3
4      5      3
The number of page faults using LRU are 5
```

RESULT:

Thus the c program to implement LRU page replacement algorithm has been executed successfully and the output is verified.

EX.NO:8 INTERPROCESS COMMUNICATION USING SHARED MEMORY

AIM:

To write a C program to implement inter process communication using shared memory.

ALGORITHM:

```
Step 1: Start the program.
Step 2: Create the child process using fork ( ).
Step 3: Create the shared memory for parent process using shmget() system call.
Step 4: Allow the parent process to write in shared memory using shmptr pointer
        which is return type of shmat().
Step 5:  Attach the same shared memory to the child process.
Step 6: The data in the shared memory is read by the child process using the shmptr
        pointer.
Step 7: Detach the shared memory.
Step 8: Stop the program.
```

PROGRAM:

```
#include<stdio.h>
#include<sys/shm.h>
#include<sys/ipc.h>
int main()
{
int child,shmid,i;
char*shmptr;
child=fork();
if(!child)
{
shmid=shmget(2041,32,0666|IPC_CREAT);
shmptr=shmat(shmid,0,0);
printf("\n parent writing \n");
for(i=0;i<10;i++)
{
shmptr[i]='a'+i;
putchar(shmptr[i]);
}
printf("\n\n%s",shmptr);
wait(NULL);
}
else
shmid=shmget(2041,32,0666);
shmptr=shmat(shmid,0,0);
printf("\n child is reading \n");
for(i=0;i<10;i++)
putchar(shmptr[i]);
shmdt(NULL);
}
```

OUTPUT:

```
Parent writing
abcdefghij
abcdefghij
child is reading
abcdefghij
```

RESULT:

Thus the c program to implement inter process communication using shared memory has been executed successfully and the output is verified.

EX.NO:9 IMPLEMENT PAGING TECHNIQUES OF MEMORY MANAGEMENT

AIM:

To write a c program to implement paging techniques of memory management.

ALGORITHM:

```
Step 1: Start the program.
Step 2: Read the number of pages in memory.
Step 3: Read the page size
Step 4: Read the number of frames.
Step 5: Read the frame number for each  page in page table.
Step 6: Read the logical address from the user which consists of page number and offset.
Step 7: If the page table frame   no is -1 then print the required page is not available in main
        memory. Else go to step 8.
Step 8: If the frame is available display the physical address with frame number and offset value.
Step 9: Stop the program.
```

PROGRAM:

```c
#include<stdio.h>
#define MAX 50
int main()
{
int page[MAX],i,n,f,ps,off,pno;
printf("\nEnter the no of pages in memory");
scanf("%d",&n);
printf("\nEnter page size");
scanf("%d",&ps);
printf("\nEnter no of frames");
scanf("%d",&f);
for(i=0;i<n;i++)
page[i]=-1;
printf("\nEnter the page table\n");
printf("(Enter frame no as -1 if that page is not present in any frame)\n\n");
printf("\npageno\tframeno\n-------\t-------");
for(i=0;i<n;i++)
{
printf("\n\n%d\t\t",i);
scanf("%d",&page[i]);
}
printf("\n\nEnter the logical address(i.e,page no & offset):");
scanf("%d%d",&pno,&off);
if(page[pno]==-1)
printf("\n\nThe required page is not available in any of frames");
else
printf("\n\nPhysical address(i.e,frame no & offset):%d,%d",page[pno],off);
return 1;
}
```

OUTPUT:

```
[it@localhost ~]$ vi pagem.c
[it@localhost ~]$ cc pagem.c
[it@localhost ~]$ ./a.out
Enter the no of pages in memory 4
Enter page size 2
Enter no of frames 4
Enter the page table
(Enter frame no as -1 if that page is not present in any frame)
pageno  frameno
------- -------
0         2
1         4
2         1
3         -1
Enter the logical address (i.e,page no & offset):2 1
Physical address(i.e,frame no & offset):1,1
```

RESULT:

Thus the c program to implement paging technique of memory management has been executed successfully and the output is verified.

EX.NO:10 IMPLEMENT THREADING AND SYNCHRONIZATION APPLICATION

AIM:

To write a c program to implement threading and synchronization application.

ALGORITHM:

Step 1: Start the program.

Step 2: Mutex is initialized using pthread_mutex_init() function in main() function.

Step 3: Create Thread1 and Thread 2 each of which executes doSomeThing() function..

Step 4: Mutex is locked in the 'doSomeThing()' function while using the shared resource 'counter' .

Step 5: At the end of the function 'doSomeThing()' the same mutex is unlocked.

Step 6: pthread_join(0 function wait for thread termination. After successful completion of thread execution, the mutex is destroyed using pthread_mutex_destroy() function.

Step 7: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
pthread_t tid[2];
int counter;
pthread_mutex_t lock;
void* doSomeThing(void *arg)
{
   pthread_mutex_lock(&lock);
   unsigned long i = 0;
   counter += 1;
   printf("\n Thread %d started\n", counter);
   for(i=0; i<(0xFFFFFFFF);i++);
   printf("\n Thread %d finished\n", counter);
   pthread_mutex_unlock(&lock);
   return NULL;
}
int main()
{
   int i = 0;
   int err;
   if (pthread_mutex_init(&lock, NULL) != 0)
   {
      printf("\n mutex init failed\n");
      return 1;
   }
```

```
    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);
    pthread_mutex_destroy(&lock);
    return 0;
}
```

OUTPUT:

```
[it@localhost ~]$ cc thread.c -lpthread
[it@localhost ~]$ ./a.out
 Thread 1 started
 Thread 1 finished
 Thread 2 started
 Thread 2 finished
```

RESULT:

Thus the c program to implement threading and synchronization application has been executed successfully and the output is verified.

****