# PRESIDENCY UNIVERSITY

Presidency University Act, 2013 of the Karnataka Act No. 41 of 2013 | Established under Section 2(f) of UGC Act, 1956
Approved by AICTE, New Delhi

## SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

## BACHELOR OF TECHNOLOGY

# Lab Manual

| | |
|---|---|
| **Course Code** | CSE2270 |
| **Course Name** | Operating Systems Lab |
| **Credit Structure** | 0-0-2-1 |
| **Year / Semester** | II/V |
| **Specialization** | B.Tech- Computer Science and Engineering |

| | |
|---|---|
| **Prepared by** | Dr. Jayavadivel Ravi & Dr.P.Sudha |

**VISION OF THE UNIVERSITY**

To be a Value-driven Global University, excelling beyond peers and creating professionals of integrity and character, having concern and care for society.

**MISSION OF THE UNIVERSITY**

- Commit to be an innovative and inclusive institution by seeking excellence in teaching, research and knowledge-transfer.
- Pursue Research and Development and its dissemination to the community, at large.
- Create, sustain and apply learning in an interdisciplinary environment with consideration for ethical, ecological and economic aspects of nation building.
- Provide knowledge-based technological support and services to the industry in its growth and development.
- To impart globally-applicable skill-sets to students through flexible course offerings and support industry's requirement and inculcate a spirit of new-venture creation.

**VISION OF PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

To be a value based, practice-driven School of Computer Science and Engineering, committed to developing globally-competent Engineers, dedicated to developing cutting-edge technology, towards enhancing Quality of Life.

**MISSION OF PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

- Cultivate a practice-driven environment, with computing-based pedagogy, integrating theory and practice.
- Attract and nurture world-class faculty to excel in Teaching and Research, in the realm of Computing Sciences.
- Establish state-of-the-art computing facilities, for effective Teaching and Learning experiences.
- Promote Interdisciplinary Studies to nurture talent for global impact.
- Instil Entrepreneurial and Leadership Skills to address Social, Environmental and Community-needs.

**PROGRAM OUTCOMES**

PO1: Application of Domain Knowledge: Apply the domain knowledge such as mathematics, science and software engineering fundamentals into the Computer Application related professions.

PO2: Problem Solving and Analysis: Identify, Formulate, Analyse and Solve Complex Scenarios related to Computer Applications.

PO3: Design/development of Activities: Conceive, Design and Develop various activities of Computer Applications.

PO4: Conduct Investigations of Events: Carry out Investigation of an event and draw logical conclusions based on critical thinking and analytical reasoning.

PO5: Modern Tool usage: Effectively apply relevant ICT Tools and digital tools to carry out Computer Application Attributes.

PO6: Research: Identify suitable Research Methods and report the findings.

PO7: Profession and Society: Apply the knowledge of the values and beliefs of multicultural society and a global perspective in the profession.

PO8: Ethics: Identify ethical issues and embrace ethical values in conduct of Profession.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Express thoughts and ideas effectively in writing and oral communication.

PO11: Project management and finance: Ability to work independently, identify appropriate resources required for a project, and manage a project through to completion.

PO12: Life-long learning: Recognize the need for,and have the preparation and ability to engage in independent and life-long learning in the broadest context of societal and technological change.

**PROGRAM SPECIFIC OUTCOMES:**

PSO1. [Problem Analysis]: Identify, formulate, research literature, and analyze complex engineering problems related to Software Engineering principles and practices, Programming and Computing technologies reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PSO2. [Design /development of Solutions]: Design solutions for complex engineering problems related to Software Engineering principles and practices, Programming and Computing technologies and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PSO3. [Modern Tool usage]: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities related to Software Engineering principles and practices, Programming and Computing technologies with an understanding of the limitations.

**COURSE DESCRIPTION:**

This laboratory course provides hands-on experience with the core concepts of operating systems through practical assignments, simulations, and case studies. It covers foundational aspects such as system calls, process and thread management, inter-process communication, synchronization, deadlocks, memory management, and file systems. Students will implement and simulate real-time OS components and scheduling algorithms, fostering deeper understanding of OS architecture and design. The lab also introduces modern OS tools, programming interfaces, and the basics of open-source OS environments.

**COURSE OBJECTIVES**

The objective of the course is to familiarize the learners with the concepts of Operating Systems and attain Employability through Problem Solving Methodologies.

**COURSE OUTCOMES: On successful completion of the course the students shall be able to:**

| TABLE 1: COURSE OUTCOMES | | |
|---|---|---|
| **CO Number** | **CO** | **Expected BLOOMS LEVEL** |
| CO1 | Demonstrate system-level programming using system calls and OS structures. | Apply |
| CO2 | Simulate process scheduling and multithreading techniques. | Apply |
| CO3 | Apply various tools to handle synchronization problems using semaphores and shared memory. | Apply |
| CO4 | Demonstrate memory management and file system concepts using simulation or scripting. | Apply |

**MAPPING OF C.O. WITH P.O [H-HIGH, M- MODERATE, L-LOW]**

| TABLE 2a: CO PO Mapping ARTICULATION MATRIX | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **CO. No** | **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO 10** | **PO 11** | **PO 12** |
| **CO1** | H | M | L | - | M | - | - | - | - | M | - | L |
| **CO2** | H | M | L | - | M | - | - | - | - | M | - | L |
| **CO3** | H | M | L | - | M | - | - | - | - | M | - | L |
| **CO4** | H | M | L | - | M | - | - | - | - | M | - | L |

| TABLE 2b: CO PSO Mapping ARTICULATION MATRIX | | | |
|---|---|---|---|
| **CO.No** | **PSO1** | **PSO2** | **PSO3** |
| **CO1** | M | M | L |
| **CO2** | M | - | L |
| **CO3** | M | - | - |
| **CO4** | M | M | - |

## Assessment Component

| TABLE 6 ASSESSMENT SCHEDULE | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Sl. No.** | **Assessment type** | **Contents** | **Course outcome Number** | **Duration (In Hours)** | **Marks** | **Weightage** | **Venue, Date & Time** |
| 1 | Lab Continuous Assessment 1 | Lab sheet 1 - 6 | CO1 & CO2 | - | 40 | 20% | Lab Continuous Assessment 1 |
| 2 | Lab Continuous Assessment 2 | Lab sheet 7 - 12 | CO3 & CO4 | - | 40 | 20% | Lab Continuous Assessment 2 |
| 3 | Record + Observation | All lab sheets | CO1, CO2, CO3 & CO4 | - | 20 | 10% | Record + Observation |

## List of Experiments

| Ex.No | Name of the Experiment |
|---|---|
| 1 | Basics of UNIX commands |
| 2 | Write programs using the following system calls of UNIX operating system fork, exec, getpid, exit, wait, close, stat, opendir, readdir |
| 3 | Write Simple programs using Shell |
| 4 | Write a C program to simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time.<br>a)FCFS b)SJF c)Round Robin(pre-emptive) d)Priority |
| 5 | Write a C program to simulate producer-consumer problem using semaphores. |
| 6 | Write a C Program to implement the Shared memory and IPC |
| 7 | Write a C Program to implement the deadlock avoidance. |
| 8 | Write a C Program to implement the deadlock detection. |
| 9 | Write a C program to simulate the following contiguous memory allocation techniques<br>a) Worst-fit b)Best-fit c)First-fit |
| 10 | Write a C program to simulate paging technique of memory management. |
| 11 | Write a C program to simulate page replacement algorithms<br>a) FIFO b) LRU c) LFU |
| 12 | Write a C program to simulate the following file allocation strategies.<br>a) Sequential b) Indexed c) Linked |
| 13 | Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue. |
| 14 | Write a C program to simulate disk scheduling algorithms<br>a) FCFS b) SCAN c) C-SCAN |
| 15 | Write a C program to simulate the concept of Dining-Philosophers problem. |

## SPECIFIC GUIDELINES TO STUDENTS:

- Maintain Minimum 75% Attendance: Regular attendance is mandatory to ensure continuity in learning and understanding of the experiments.
- Missing lab sessions can lead to gaps in understanding and affect your ability to complete experiments and assignments.
- Carefully follow the instructions provided by the course instructor during both lectures and lab sessions.
- Conduct yourself in a responsible manner at all times in the laboratory. Intentional misconduct will lead to the exclusion from the lab.
- Do not wander around, or distract other students, or interfere with the laboratory experiments of other students.
- Read the handout and procedures before starting the experiments. Follow all written and verbal instructions carefully. If you do not understand the procedures, ask the instructor or teaching assistant.
- The workplace has to be tidy before, during and after the experiment.
- Do not eat food, drink beverages or chew gum in the laboratory.
- Every student should know the location and operating procedures of all Safety equipment including First Aid Kit and Fire extinguisher.

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

**AIM:**

To study and execute Unix commands.

**PROCEDURE:**

Unix is security conscious, and can be used only by those persons who have an account.

Telnet (Telephone Network) is a Terminal emulator program for TCP/IP networks that enables users to log on to remote servers.

To logon, type telnet server_ipaddress in run window.

User has to authenticate himself by providing username and password. Once verified, a greeting and $ prompt appears. The shell is now ready to receive commands from the user. Options suffixed with a hyphen (–) and arguments are separated by space.

**GENERAL COMMANDS**

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

## DIRECTORY COMMANDS

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

## FILE COMMANDS

| Command | Function |
|---|---|
| cat >filename | To create a file with some contents. To end typing press **Ctrl+d**.<br>The >symbol means redirecting output to a file. (<for input) |
| cat filename | Displays the file contents. |
| cat >>filename | Used to append contents to a file |
| cpsrc des | Copy files to given location. If already exists, it will be overwritten |
| cp –i src des | Warns the user prior to overwriting the destination file |
| cp –r src des | Copies the entire directory, all its sub-directories and files. |
| mv old new | To rename an existing file or directory. –i option can also be used |
| mv f1 f2 f3 dir | To move a group of files to a directory. |
| mv –v old new | Display name of each file as it is moved. |
| Rmfile | Used to delete a file or group of files. –i option can also be used |
| rm * | To delete all the files in the directory. |
| rm –r * | Deletes all files and sub-directories |
| rm –f * | To forcibly remove even write-protected files |
| Ls | Lists all files and subdirectories (blue colored) in sorted manner. |
| Lsname | To check whether a file or directory exists. |
| lsname***** | Short-hand notation to list out filenames of a specific pattern. |
| ls –a | Lists all files including hidden files (files beginning with **.**) |
| ls –x dirname | To have specific listing of a directory. |
| ls –R | Recursive listing of all files in the subdirectories |

| | |
|---|---|
| ls –l | Long listing showing file access rights (read/write/execute-**rwx** for user/group/others-**ugo**). |
| cmpfile1 file2 | Used to compare two files. Displays nothing if files are identical. |
| Wcfile | It produces a statistics of lines (**l**), words(**w**), and characters(**c**). |
| chmodperm file | Changes permission for the specified file. (r=4, w=2, x=1)<br>chmod 740 file sets all rights for user, read only for groups<br>and no rights for others |

**OUTPUT**

**GENERAL COMMANDS**

**[student@veccse ~]date**

Sat May 16 06:10:34 UTC 2020

**[student@veccse ~]date +%D**

05/16/20

**[student@veccse ~]date +%T**

10:13:11

**[student@veccse ~]date +%Y**

2020

**[student@veccse ~]date +%H**

10

**[student@veccse ~]cal**

```
May 2020
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

**[student@veccse ~]cal 2020**

```
                                2020
       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  4                      1      1  2  3  4  5  6  7
 5  6  7  8  9 10 11    2  3  4  5  6  7  8      8  9 10 11 12 13 14
12 13 14 15 16 17 18    9 10 11 12 13 14 15     15 16 17 18 19 20 21
19 20 21 22 23 24 25   16 17 18 19 20 21 22     22 23 24 25 26 27 28
```

```
26 27 28 29 30 31      23 24 25 26 27 28 29   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
                       30 31

       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@veccse ~]cal 2020**

```
July 2020
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

**[student@veccse ~]who**

studentpts/1 May 16 10:05 (172.16.1.14)

**[student@veccse ~]who am i**

studentpts/1 May 16 10:05 (172.16.1.14)

**[student@veccse ~]tty**

/dev/pts/1

**[student@veccse ~]uname**

Linux

**[student@veccse ~]echo "hello"**

hello

**[student@veccse ~]echo $HOME**

/home/student

**[student@veccse ~]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'.

**[student@veccse ~]man lp**

lp(1) Easy Software Products lp(1)

NAME

lp - print files cancel - cancel jobs SYNOPSIS

lp [ -E ] [ -c ] [ -d destination ] [ -h server ] [ -m ] [ -n num- copies [ -o option ] [ -q priority ] [ -s ] [ -t title ] [ -H handling

] [ -P page-list ] [ file(s) ]

lp [ -E ] [ -c ] [ -h server ] [ -i job-id ] [ -n num-copies [ -o option ] [ -q priority ] [ -t title ] [ -H handling ] [ -P page-list ] cancel [ -a ] [ -h server ] [ -u username ] [ id ] [ destination ] [ destination-id ]

DESCRIPTION

lpsubmits files for printing or alters a pending job. Use a filename of "-" to force printing from the standard input.

cancelcancels existing print jobs. The -a option will remove all jobs from the specified destination.

OPTIONS

The following options are recognized by lp:

**[student@veccse ~]history**

1 date

2 date +%D

3 date +%T

4 date +%Y

5 date +%H

6 cal

7 cal 2020

8 cal 7 2020

10 who

11 who am i

12 tty

13 uname

14 uname -r

15 uname -n

16 echo "helloi"

17 echo $HOME

18 bc

19 man lp

20 history



**DIRECTORY COMMANDS**

**[student@veccse]$ pwd**

/home/student

**[student@veccse ~]mkdir san**

**[student@veccse ~]mkdir s1 s2**

**[student@veccse ~]ls**

s1 s2 san

**[student@veccse ~]cd s1**

**[student@veccse s1]$ cd /**

**[student@veccse /]$ cd . .**

**[student@veccse /]$ rmdir s1**

**[student@veccse ~]$ ls**

s2 san

**FILE COMMANDS**

**[student@vecit ~]$ cat>test**

hi welcome operating systems lab

**[student@vecit ~]$ cat test**

hi welcome operating systems lab

**[student@vecit ~]$ cat>>test fourth semester**

**[student@vecit ~]$ cat test**

hi welcome operating systems lab fourth semester

**[student@vecit ~]$ cat>test1**

**[student@vecit ~]$ cp test test1**

**[student@vecit ~]$ cat test1**

hi welcome operating systems lab fourth semester

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

**[student@vecit ~]$ cp -r test test1**

**[student@vecit ~]$ ls**

s s2 san swap.sh temp.sh test TEST test1



**[student@vecit ~]$ mv san san1**

**[student@vecit ~]$ ls**

s s2 san1 swap.sh temp.sh test TEST test1

**[student@vecit ~]$ mv test test1 san1**

**[student@vecit ~]$ mv -v san1 sannew**

`san1' -> `sannew'

**[student@vecit ~]$ ls**

s s2 sannew swap.sh temp.sh TEST

**[student@vecit ~]$ cmp test test1**

cmp: test: No such file or directory

**RESULT**

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

**VIVA QUESTIONS**

1. What is the use of cat commands?
2. Define Operating Systems?
3. What is the use of filter/grep/pipe commands?
4. How is unix different from windows
5. What is unix?
6. What is the file structure of unix?
7. What is a kernel?
8. What is the difference between multi-user and multi-tasking?
9. Differentiate relative path from absolute path.
10. What are the differences among a system call, a library function, and a UNIX command.

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

**AIM:**

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

**ALGORITHM:**

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



**PROGRAM:**

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

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

**OUTPUT:**

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



**RESULT:**

Thus the program for process management was written and successfully executed

## EX.NO.2B: IMPLEMENTATION OF OPENDIR AND READDIR SYSTEM CALLS

**AIM:**

To write a program for implementing Directory management using the following system calls of UNIX operating system: opendir, readdir.

**ALGORITHM:**

1. Start the program.
2. Open the directory at specified in command line input.
3. Display the directory contents.
4. Stop the program.

**PROGRAM:**

```
#include<sys/types.h>
#include<dirent.h>
#include<stdio.h>
main(int c, char* arg[])
{
        DIR *d;
        struct dirent *r; int i=0;
        d=opendir(arg[1]);
        printf("\n\t NAME OF ITEM \n");
        while((r=readdir(d)) != NULL)
        {
                printf("\t %s \n",r->d_name); i=i+1;
        }
        printf("\n TOTAL NUMBER OF ITEM IN THAT DIRECTORY IS %d \n",i);
}
```



**OUTPUT:**

```
[root@localhost ~]# cc dr.c
[root@localhost ~]# ./a.out lab_print
NAME OF ITEM pri_output.doc sjf_output.doc fcfs_output.doc rr_output.doc
ipc_pipe_output.doc
```

pro_con_prob_output.doc

TOTAL NUMBER OF ITEM IN THAT DIRECTORY IS 8

**RESULT:**

Thus the program for directory management was written and successfully executed.

**VIVA QUESTIONS:**

1. What is the purpose of system calls?
2. What system calls have to be executed by a command interpreter or shell in order to start a new process?
3. When a process creates a new process using the fork() operation, which of the following state is shared between the parent process and the child process?
4. What is the use of exec system call?
5. What system call is used for closing a file?
6. What is the value return by close system call?
7. What is the system call is used for writing to a file.
8, what are system calls used for creating and removing directories?

**EX. NO: 3A: SIMPLE SHELL PROGRAMS**

**AIM:**

To write simple shell scripts using shell programming fundamentals.

**DESCRIPTION:**

The activities of a shell are not restricted to command interpretation alone. The shell also has Rudimentary programming features. When a group of commands has to be executed regularly, they are stored in a file (with extension .sh). All such files are called shell scripts or shell programs. Shell programs run in interpretive mode.

The original UNIX came with the Bourne shell (sh) and it is universal even today. Then came a plethora of shells offering new features. Two of them, C shell (csh) and Korn shell (ksh) has been well accepted by the UNIX fraternity. Linux offers Bash shell (bash) as a superior alternative to Bourne shell.

**Preliminaries**

1. Comments in shell script start with #. It can be placed anywhere in a line; the shell ignores contents to its right. Comments are recommended but not mandatory
2. Shell variables are loosely typed i.e. not declared. Their type depends on the value assigned. Variables when used in an expression or output must be prefixed by $.
3. The read statement is shell's internal tool for making scripts interactive.
4. Output is displayed using echo statement. Any text should be within quotes. Escape sequence should be used with –e option.
5. Commands are always enclosed with ` ` (back quotes).
6. Expressions are computed using the expr command. Arithmetic operators are + - * / %. Meta characters * ( ) should be escaped with a \.
7. Multiple statements can be written in a single line separated by ;
8. The shell scripts are executed using the sh command (sh filename).

**<u>Swapping values of two variables</u>**

**Algorithm**

Step 1 : Start

Step 2 : Read the values of a and b

Step 3 : Interchange the values of a and b using another variable t as follows: t = a a = b b = t

Step 4 : Print a and b

Step 5 : Stop

**Program (swap.sh) # Swapping values**

```
echo -n "Enter value for A : " read a
echo -n "Enter value for B : " read b
t=$a a=$b b=$t
echo "Values after Swapping" echo "A Value is $a"
echo "B Value is $b"
```

**Output**

```
[student@vecit ~]$ sh swap.sh Enter Value for A:5
Enter Value for B:6 Values after Swapping A value is 6
B values is 5 [student@vecit ~]$
```

**Farenheit to Centigrade Conversion**

**Algorithm**

Step 1: Start

Step 2: Read Fahrenheit value

Step 3: Convert Fahrenheit to centigrade using the formulae:

(Fahrenheit – 32) × 5/9

Step 4: Print centigrade

Step 5: Stop

**Program**

```
# Degree conversion
echo -n "Enter Fahrenheit : " read f
c=`expr\( $f - 32 \) \* 5 / 9`
echo "Centigrade is : $c"
```

**Output**

```
[student@vecit ~]$ sh temp.sh Enter Fahrenheit:4 Centrigrade is: -15
[student@vecit ~]$
```

**RESULT**

**EX.NO.4A: IMPLEMENTATION OF FCFS SCHEDULING ALGORITHM**

**AIM**

To write a C program to implement First Come First Serve scheduling algorithm.

**DESRIPTION:**

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.

.**ALGORITHM:**

Step 1: Start the program.

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

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

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

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

Step 6: Print the details about all the processes.

Step 7: Stop the program.



**PROGRAM**

PROGRAM:

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

```
        scanf("%s%d",p[i],&bt[i]);
printf("\nEnter the arrival-time:");
for(i=0;i<n;i++)
        scanf("%d",&at[i]);
wt[0]=0;
for(i=1;i<=n;i++)
        wt[i]=wt[i-1]+bt[i-1];
ft[0]=bt[0];
for(i=1;i<=n;i++)
        ft[i]=ft[i-1]+bt[i];
printf("\n\n\t\t\tGANTT CHART\n");
printf("\n      \n");
for(i=0;i<n;i++)
        printf("|\t%s\t",p[i]);
printf("|\t\n");
printf("\n      \n");
printf("\n");
for(i=0;i<n;i++)
        printf("%d\t\t",wt[i]);
printf("%d",wt[n]+bt[n]);
printf("\n      \n");
printf("\n");
for(i=0;i<n;i++)
        wat[i]=wt[i]-at[i];
for(i=0;i<n;i++)
        tat[i]=wat[i]-at[i];
printf("\n FIRST COME FIRST SERVE\n");
printf("\n Process Burst-time Arrival-time Waiting-time Finish-time Turnaround-
time\n");
for(i=0;i<n;i++)
        printf("\n\n  %d%s  \t  %d\t\t  %d  \t\t  %d\t\t  %d  \t\t
        %d",i+1,p[i],bt[i],at[i],wat[i],ft[i],tat[i]);
for(i=0;i<n;i++)
        sum=sum+wat[i];
awt=sum/n;
```

```
        for(i=0;i<n;i++)
                sum1=sum1+bt[i]+wt[i];
        att=sum1/n;
        printf("\n\nAverage waiting time:%f",awt);
        printf("\n\nAverage turnaround time:%f",att);
}
```

**OUTPUT:**

enter the number of process 3

Enter the process name and burst-time:

p1 2

p2 3

p3 4

Enter the arrival-time:0 1 2

GANTT CHART

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

0 2 5 9



FIRST COME FIRST SERVE

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



Average waiting time:1.333333

Average turnaround time:5.333333

**RESULT:**

The FCFS scheduling algorithm has been implemented in C.

**EX.NO.4B : IMPLEMENTATION OF SJF SCHEDULING ALGORITHM**

**AIM**

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

**DESCRIPTION:**

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.

**ALGORITHM:**

Step 1: Start the program.

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

Step 3: Sort the processes based on burst time.

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

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

Step 6: Print the details about all the processes.

Step 7: Stop the program.



**PROGRAM:**

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

```
for(i=1;i<=n;i++)
{
        scanf("%d\t %d\t %d",&p[i],&bt[i],&at[i]);
}
printf("\nPROCESS\tBURSTTIME\tARRIVALTIME");
for(i=1;i<=n;i++)
{
        printf("\np%d\t%d\t\t%d",p[i],bt[i],at[i]);
}
for(i=1;i<=n;i++)
{
        for(j=i+1;j<=n;j++)
        {
                if(bt[i]>bt[j])
                {
                        temp=bt[i];
                        bt[i]=bt[j];
                        bt[j]=temp;
                        temp1=p[i];
                        p[i]=p[j];
                        p[j]=temp1;
                }
        }
        if(i==1)
        {
                st[1]=0;
                ft[1]=bt[1]; wt[1]=0;
        }
        else
        {
                st[i]=ft[i-1];
                ft[i]=st[i]+bt[i];
```

```
                wt[i]=st[i];
        }
}
printf("\n\n\t\t\tGANTT CHART\n");
printf("\n     \n");
for(i=1;i<=n;i++)
        printf("|\tp%d\t",p[i]);
printf("|\t\n");
printf("\n     \n");
printf("\n");
for(i=1;i<=n;i++)
        printf("%d \t\t",wt[i]);
printf("%d",wt[n]+bt[n]);
printf("\n     \n");
 for(i=2;i<=n;i++)
        wat[i]=wt[i]-at[i];
for(i=1;i<=n;i++)
         tt[i]=wat[i]+bt[i]-at[i];
printf("\nPROCESS\tBURSTTIME\tARRIVALTIME\tWAITINGTIME\tT
URNAROUNDTI ME\n");
for(i=1;i<=n;i++)
{
        printf("\np%d %5d %15d %15d %15d",p[i],bt[i],at[i],wat[i],tt[i]);
}
for(i=1,tot=0;i<=n;i++)
        tot+=wt[i];
awt=(float)tot/n;
printf("\n\n\n AVERAGE WAITING TIME=%f",awt);
for(i=1,tot=0;i<=n;i++)
        tot+=tt[i];
att=(float)tot/n;
printf("\n\n AVERAGE TURNAROUND TIME=%f",att);
```

}

**OUTPUT:**

enter the no.of process3

enter the process number,burst time and arrival time

1 8 1

2 5 1

3 3 1

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

AVERAGE WAITING TIME=3.666667

AVERAGE TURNAROUND TIME=7.333333

**RESULT:**

The SJF scheduling algorithm has been implemented in C.

## EX.NO.4C: IMPLEMENTATION OF ROUND ROBINSCHEDULING ALGORITHM

**AIM:**

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

**DESCRIPTION:**

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.

**ALGORITHM:**

Step 1: Start the program.

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

Step 3: Sort the processes based on priority.

Step 4: Compute the waiting time and turnaround time for each process. Step 5: Calculate the average waiting time and average turnaround time. Step 6: Print the details about all the processes.

Step 7: Stop the program.



**PROGRAM:**

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

```
for(i=0;i<n;i++)
{
        scanf("%s",p[i]);
        scanf("%d",&bt[i]);
        scanf("%d",&at[i]); wt[i]=t[i]=0;
        b[i]=bt[i];
}
printf("\n\t\tGANTT CHART");
printf("\n     \n");
for(i=0;i<n;i++)
{
        if(max<bt[i])
                max=bt[i];
}
while(max!=0)
{
        for(i=0;i<n;i++)
        {
                if(bt[i]>0)
                {
                        if(ct==0)
                                wt[i]=wt[i]+cwt;
                        else
                                wt[i]=wt[i]+(cwt-t[i]);
                }
                if(bt[i]==0)
                cwt=cwt+0;
                else if(bt[i]==max)
                {
                        if(bt[i]>tq)
                        {
                                cwt=cwt+tq;
```

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

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

**OUTPUT:**

enter the no of process:3

enter the time quantum2

enter the process name, bursttime, arrival time

p1 2 0

p2 3 1

p3 4 2

**GANTT CHART**

| | p1 | p2 | p3 | p2 | p3 |
|---|---|---|---|---|---|
| 0 | 2 | 4 | 6 | 7 | 9 |

**ROUND ROBIN**

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

Avg Waiting Time=2.000000

Avg Turnaround Time=4.000000



**RESULT**

The Round Robin scheduling algorithm has been implemented in C.

**EX.NO.4D: IMPLEMENTATION OF PRIORITY SCHEDULING ALGORITHM**

**AIM**

To write a C program to implement Priority Scheduling algorithm.

**DESCRIPTION:**

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**.**

**ALGORITHM:**

Step 1: Start the program.

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

Step 3: Sort the processes based on priority.

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

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

Step 6: Print the details about all the processes.

Step 7: Stop the program.



**PROGRAM:**

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

```
 for(i=0;i<np;i++)
          scanf("%s%d%d",p[i],&bt[i],&pr[i]);
printf("\nEnter the arrival-time:");
for(i=0;i<np;i++)
        scanf("%d",&at[i]);
 for(i=0;i<np;i++)
        for(j=i+1;j<np;j++)
        {
                if(pr[i]>pr[j])
                {
                        x=pr[j];
                         pr[j]=pr[i];
                        pr[i]=x;
                        strcpy(y,p[j]);
                        strcpy(p[j],p[i]);
                        strcpy(p[i],y);
                         z=bt[j]; b
                        t[j]=bt[i];
                        bt[i]=z;
                }
        }
wt[0]=0;
for(i=1;i<=np;i++)
        wt[i]=wt[i-1]+bt[i-1];
ft[0]=bt[0];
 for(i=1;i<np;i++)
          ft[i]=ft[i-1]+bt[i];
printf("\n\n\t\tGANTT CHART\n");
printf("\n     \n");
for(i=0;i<np;i++)
        printf("|\t%s\t",p[i]);
```

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



**OUTPUT:**

Enter the number of process3

Enter the process, burst-time and priority:

p1 3 3

p2 4 2

p3 5 1

Enter the arrival-time: 0 1 2

GANTT CHART

| p3 | p2 | p1 |

0 5 9 12

**PRIORITY SCHEDULING:**

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

Average waiting time: 3.666667

Average turnaround time is: 2.666667

**RESULT**

The Priority scheduling algorithm has been implemented in C.

**VIVA QUESTIONS:**

1. Define operating system?
2. What are the different types of operating systems?
3. Define a process?
4. What is CPU Scheduling?
5. Define arrival time, burst time, waiting time, turnaround time?
6. What is the advantage of round robin CPU scheduling algorithm?
7. Which CPU scheduling algorithm is for real-time operating system?
8. In general, which CPU scheduling algorithm works with highest waiting time?
9. Is it possible to use optimal CPU scheduling algorithm in practice?
10. What is the real difficulty with the SJF CPU scheduling algorithm?

**ASSIGNMENT QUESTIONS**

1. Write a C program to implement round robin CPU scheduling algorithm for the following given scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Consider the time quantum size for the system processes and user processes to be 5 msec and 2 msec respectively.

2. Write a C program to simulate pre-emptive SJF CPU scheduling algorithm.

**EX.NO:5 PRODUCER CONSUMER PROBLEM USING SEMAPHORE**

**AIM:**

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

**DESCRIPTION:**

Producer-consumer problem, is a common paradigm for cooperating processes. A producer process produces information that is consumed by a consumer process. One solution to the producer-consumer problem uses shared memory. To allow producer and consumer processes to run concurrently, there must be available a buffer of items that can be filled by the producer and emptied by the consumer. This buffer will reside in a region of memory that is shared by the producer and consumer processes. A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

**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 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 3: In the case of consumer process

i) It should wait if the buffer is empty

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

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

iv) Consume the item.

Step 4: Print the result

**PROGRAM :**

```
#define BUFFERSIZE 10
int mutex,n,empty,full=0,item,item1;
 int buffer[20];
int in=0,out=0,mutex=1;
void wait(int s)
{
        while(s<0)
        {
                printf("\nCannot add an item\n");
                 exit(0);
        }
        s--;
}
void signal(int s)
{
        s++;
}
void producer()
{
        do
        {
                wait (empty);
                wait(mutex);
                printf("\nEnter an item:");
                scanf("%d",&item);
                buffer[in]=item;
                in=in+1;
                signal(mutex);
                signal(full);
```

```
        }
        while(in<n);
}
void consumer()
{
        do
        {
                wait(full);
                wait(mutex);
                item1=buffer[out];
                printf("\nConsumed item =%d",item1);
                 out=out+1;
                signal(mutex);
                signal(empty);
        }
        while(out<n);
}
void main()
{
        printf("Enter the value of n:");
        scanf("%d ",&n);
        empty=n;
        while(in<n)
                producer();
        while(in!=out)
                 consumer();
}
```



**OUTPUT:**

```
$ cc prco.c
$ a.out
Enter the value of n :3
Enter the item:2
```

Enter the item:5

Enter the item:9

consumed item=2

consumed item=5

consumed item=9

$

**RESULT:**

Thus the program for solving producer and consumer problem using semaphore was executed successfully.

**VIVA QUESTIONS**

1. Define Semaphore?
2. What is use of wait and signal functions?
3. What is mutual exclusion?
4. Define producer consumer problem?
5. What is the need for process synchronization?
6. Discuss the consequences of considering bounded and unbounded buffers in producer-consumer problem?
7. Can producer and consumer processes access the shared memory concurrently? If not which technique provides such a benefit?



**ASSIGNMENT QUESTION:**

1.Write a C program to simulate producer-consumer problem using message-passing system.

**EX.NO:6 IMPLEMENTATION OF SHARED MEMORY AND IPC**

**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. Data is read.
5. Stop the program.

**SHARED MEMORY FOR WRITER PROCESS**



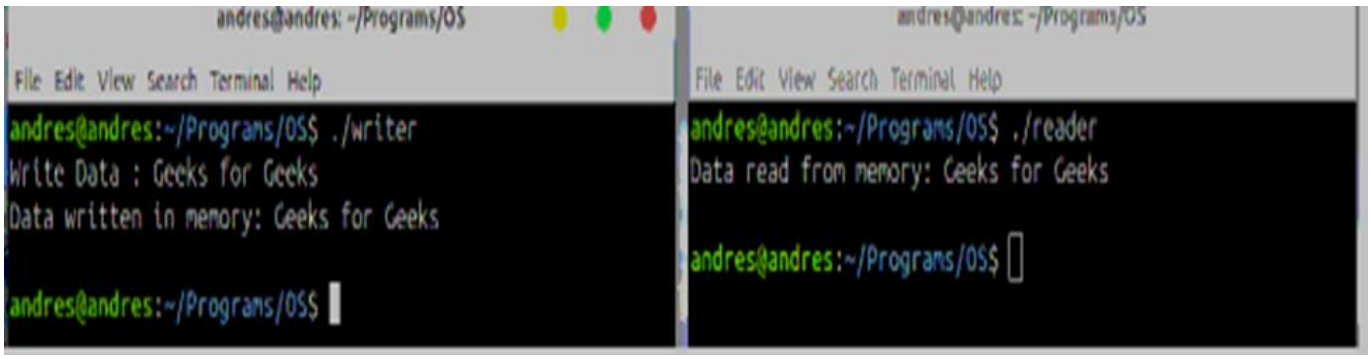
```
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
 #include <stdio.h>
using namespace std;
int main()
{
        // ftok to generate unique key key_t
        key = ftok("shmfile",65);
        // shmget returns an identifier in shmid
        int shmid = shmget(key,1024,0666|IPC_CREAT);
        // shmat to attach to shared memory
        char *str = (char*) shmat(shmid,(void*)0,0);
        printf("Write Data : ");
        gets(str);
        printf("Data written in memory: %s\n",str);
        //detach from shared memory
        shmdt(str);
        return 0;
```

}

**SHARED MEMORY FOR READER PROCESS**

```
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
using namespace std;
int main()
{
        // ftok to generate unique key key_t
         key = ftok("shmfile",65);
        // shmget returns an identifier in shmid
        int shmid = shmget(key,1024,0666|IPC_CREAT);
        // shmat to attach to shared memory
        char *str = (char*) shmat(shmid,(void*)0,0);
        printf("Data read from memory: %s\n",str);
        //detach from shared memory
        shmdt(str);
        // destroy the shared memory
        shmctl(shmid,IPC_RMID,NULL);
        return 0;
}
```

OUTPUT:



**RESULT:**

Thus the program was executed successfully.

**VIVA QUESTIONS**

1. What is IPC?
2. What is the use shared memory?
3. List commands used for shared memory communication?
4. What is the function of shmget function?
5. What is the use of shmctl fuction?

**EX.NO: 7** **DEADLOCK AVOIDANCE**

**AIM:**

To Simulate Algorithm for Deadlock avoidance

**DESCRIPTION:**

In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock. Deadlock avoidance is one of the techniques for handling deadlocks. This approach requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process. Banker's algorithm is a deadlock avoidance algorithm that is applicable to a system with multiple instances of each resource type



**ALGORITHM:**

Step 1: Start the Program

Step 2: Get the values of resources and processes.

Step 3: Get the avail value.

Step 4: After allocation find the need value.

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

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

Step 7: Stop the execution

10.Stop the program

**PROGRAM:**

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

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

```
{
        avail[j]=0;
       total=0;
       for(i=1;i<= pno;i++)
       {
              total+=allocated[i][j];
        }
       avail[j]=tres[j]-total;
       work[j]=avail[j];
       printf("     %d \t",work[j]);
}
do
{
       for(i=1;i<= pno;i++)
       {
              for(j=1;j<= rno;j++)
              {
                     need[i][j]=max[i][j]-allocated[i][j];
              }
       }
       printf("\n Allocated matrix       Max      need");
       for(i=1;i<= pno;i++)
       {
              printf("\n");
              for(j=1;j<= rno;j++)
              {
                     printf("%4d",allocated[i][j]);
              }
              printf("|");
              for(j=1;j<= rno;j++)
              {
                     printf("%4d",max[i][j]);
```

```
                }
                printf("|");
                for(j=1;j<= rno;j++)
                {
                        printf("%4d",need[i][j]);
                }
        }
 prc=0;
 for(i=1;i<= pno;i++)
{
        if(flag[i]==0)
        {
                prc=i;
                for(j=1;j<= rno;j++)
                {
                        if(work[j]< need[i][j])
                        {
                                prc=0;
                                break;
                        }
                }
        }
         if(prc!=0)
         break;
}

 if(prc!=0)
{
        printf("\n Process %d completed",i);
        count++;
         printf("\n Available matrix:");
         for(j=1;j<= rno;j++)
```

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

**OUTPUT**



Enter number of process:5

Enter number of resources:3

Enter total numbers of each resources:10 5 7

Enter Max resources for each process:

for process 1: 7 5 3

for process 2: 3 2 2

for process 3: 9 0 2

for process 4: 2 2 2

for process 5: 4 3 3

Enter allocated resources for each process:

for process 1: 0 1 0

for process 2: 3 0 2

for process 3: 3 0 2

for process 4: 2 1 1

for process 5: 0 0 2

available resources:

2 3 0

Allocated matrix Max need

0 1 0| 7 5 3| 7 4 3

3 0 2| 3 2 2| 0 2 0

3 0 2| 9 0 2| 6 0 0

2 1 1| 2 2 2| 0 1 1

0 0 2| 4 3 3| 4 3 1

Process 2 completed

Available matrix: 5 3 2

Allocated matrix Max need

0 1 0| 7 5 3| 7 4 3

0 0 0| 0 0 0| 0 0 0

3 0 2| 9 0 2| 6 0 0

2 1 1| 2 2 2| 0 1 1

0 0 2| 4 3 3| 4 3 1

Process 4 completed

Available matrix: 7 4 3

Allocated matrix Max need

0 1 0| 7 5 3| 7 4 3

0 0 0| 0 0 0| 0 0 0

3 0 2| 9 0 2| 6 0 0

0 0 0| 0 0 0| 0 0 0

0 0 2| 4 3 3| 4 3 1

Process 1 completed

Available matrix: 7 5 3

Allocated matrix Max need

0 0 0| 0 0 0| 0 0 0

0 0 0| 0 0 0| 0 0 0

3 0 2| 9 0 2| 6 0 0

0 0 0| 0 0 0| 0 0 0

0 0 2| 4 3 3| 4 3 1

Process 3 completed

Available matrix: 10 5 5

Allocated matrix Max need

0 0 0| 0 0 0| 0 0 0

0 0 0| 0 0 0| 0 0 0

0 0 0| 0 0 0| 0 0 0

0 0 0| 0 0 0| 0 0 0

0 0 2| 4 3 3| 4 3 1

Process 5 completed

Available matrix: 10 5 7

The system is in a safe state!!

**RESULT:**

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



**VIVA QUESTIONS:**

1. How to recover once the Deadlock has been detected?
2. List the steps to illustrate the Deadlock Detection Algorithm.
3. What are the advantages to check each resource request?
4. .How to fill the Allocation matrix?
5. How to identify whether the process exist in deadlock or not?

**EX.NO:8 DEADLOCK DETECTION ALGORITHM**

**AIM:**

To Simulate Algorithm for Deadlock detection

**ALGORITHM**:

Step 1: Start the Program

Step 2: Get the values of resources and processes.

Step 3: Get the avail value..

Step 4: After allocation find the need value.

Step 5: Check whether its possible to allocate.

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

Step 7: Stop the execution



**PROGRAM**

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

```
}

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

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

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

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



**OUTPUT:**

Enter the no. Of processes 3

Enter the no of resources instances 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 deadlock process are

P1 P2 P3

**RESULT:**

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



**VIVA QUESTIONS:**

1. Define Deadlock Prevention.
2. List the difference Between Starvation and Deadlock.
3. Give the advantages of Deadlock.
4. List the disadvantages of Deadlock method.
5. Define resource. Give examples.
6. What are the conditions to be satisfied for the deadlock to occur?
7. How can be the resource allocation graph used to identify a deadlock situation?
8. How is Banker's algorithm useful over resource allocation graph technique?
9. Differentiate between deadlock avoidance and deadlock prevention?

**ASSIGNMENT**

1. Write a C program to implement deadlock detection technique for the following scenarios?

a. Single instance of each resource type

b. Multiple instances of each resource type

**Ex.NO: 9 IMPLEMENTATION OF MEMORY ALLOCATION TECHNIQUES**

**AIM:**

To write a C program to implement Memory Management concept using the technique best fit, worst fit and first fit algorithms.

**ALGORITHM:**

1. Get the number of process.
2. Get the number of blocks and size of process.
3. Get the choices from the user and call the corresponding switch cases.
4. First fit -allocate the process to the available free block match with the size of the process
5. Worst fit –allocate the process to the largest block size available in the list
6. Best fit-allocate the process to the optimum size block available in the list
7. Display the result with allocations

**PROGRAM:**



```
#include<stdio.h>
main()
{
        int p[10],np,b[10],nb,ch,c[10],d[10],alloc[10],flag[10],i,j;
        printf("\nEnter the no of process:");
        scanf("%d",&np);
        printf("\nEnter the no of blocks:");
        scanf("%d",&nb);
        printf("\nEnter the size of each process:");
        for(i=0;i<np;i++)
        {
                printf("\nProcess %d:",i);
                scanf("%d",&p[i]);
        }
```

```
printf("\nEnter the block sizes:");
for(j=0;j<nb;j++)
{
        printf("\nBlock %d:",j);
        scanf("%d",&b[j]);c[j]=b[j];d[j]=b[j];
}
if(np<=nb)
{
        printf("\n1.First fit 2.Best fit 3.Worst fit");
        do
        {
                printf("\nEnter your choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: printf("\nFirst Fit\n");
                                for(i=0;i<np;i++)
                                {
                                        for(j=0;j<nb;j++)
                                        {
                                                if(p[i]<=b[j])
                                                {
                                                alloc[j]=p[i];printf("\n\nAlloc[%d]",all
                                                oc[j]);
                                                printf("\n\nProcess %d of size %d is
                                        allocated in block:%d of size:%d",i,p[i],j,b[j]);
                                                flag[i]=0,b[j]=0;break;
                                                }
                                                else
                                                        flag[i]=1;
```

```
                    }
            }
            for(i=0;i<np;i++)
            {
                    if(flag[i]!=0)
                    printf("\n\nProcess %d of size %d is not
            allocated",i,p[i]);
            }
            break;

case 2: printf("\nBest Fit\n");
            for(i=0;i<nb;i++)
            {
                    for(j=i+1;j<nb;j++)
                    {
                            if(c[i]>c[j])
                            {
                                    int temp=c[i];
                                    c[i]=c[j];
                                     c[j]=temp;
                            }
                    }
            }
            printf("\nAfter sorting block sizes:");
            for(i=0;i<nb;i++)
            printf("\nBlock %d:%d",i,c[i]);
            for(i=0;i<np;i++)
            {
                    for(j=0;j<nb;j++)
                    {
```

```
                if(p[i]<=c[j])
                {
                        alloc[j]=p[i];printf("\n\nAlloc[%d]",all
                        oc[j]);
                        printf("\n\nProcess %d of size %d is
                        allocated in block %d of size
                        %d",i,p[i],j,c[j]);
                        flag[i]=0,c[j]=0;break;
                }
                else
                        flag[i]=1;
        }
}
for(i=0;i<np;i++)
{
        if(flag[i]!=0)
        printf("\n\nProcess %d of size %d is not
        allocated",i,p[i]);
}
break;
case 3: printf("\nWorst Fit\n");
        for(i=0;i<nb;i++)
        {
                for(j=i+1;j<nb;j++)
                {
                        if(d[i]<d[j])
                        {
                                int temp=d[i];
                                d[i]=d[j];
                                d[j]=temp;
                        }
```

```
        }
}
printf("\nAfter sorting block sizes:");
for(i=0;i<nb;i++)
printf("\nBlock %d:%d",i,d[i]);
for(i=0;i<np;i++)
{
        for(j=0;j<nb;j++)
        {
                if(p[i]<=d[j])
                {
                alloc[j]=p[i];
                printf("\n\nAlloc[%d]",alloc[j]);
                printf("\n\nProcess %d of size %d is
                allocated in block %d of size
                %d",i,p[i],j,d[j]
                flag[i]=0,d[j]=0;break;
                }
                else
                        flag[i]=1;
        }
}
for(i=0;i<np;i++)
{
        if(flag[i]!=0)
                        printf("\n\nProcess %d of size
                        %d is not allocated",i,p[i]);
}
break;
default:        printf("Invalid Choice…!");break;
```

```
                                    }
                }while(ch<=3);
        }
}
```

**OUTPUT**

```
Enter the no of process:3
Enter the no of blocks:3
Enter the size of each process:
Process 0:100
Process 1:150
Process 2:200
Enter the block sizes:
Block 0:300
Block 1:350
Block 2:200
1.First fit 2.Best fit 3.Worst fit
Enter your choice:1
Alloc[100]
Process 0 of size 100 is allocated in block 0 of size 300
Alloc[150]
Process 1 of size 150 is allocated in block 1 of size 350
Alloc[200]
Process 2 of size 200 is allocated in block 2 of size 200
Enter your choice:2
Best Fit
After sorting block sizes are:
Block 0:200
Block 1:300
Block 2:350
```

Alloc[100]

Process 0 of size 100 is allocated in block:0 of size:200

Alloc[150]

Process 1 of size 150 is allocated in block:1 of size:300

Alloc[200]

Process 2 of size 200 is allocated in block:2 of size:350

enter your choice:3

Worst Fit

After sorting block sizes are:

Block 0:350

Block 1:300

Block 2:200

Alloc[100]

Process 0 of size 100 is allocated in block 0 of size 350

Alloc[150]

Process 1 of size 150 is allocated in block 1 of size 300

Alloc[200]

Process 2 of size 200 is allocated in block 2 of size 200

Enter your choice:6

Invalid Choice…!

**RESULT:**

Thus a UNIX C program to implement memory management scheme using Best fit worst fit and first fit were executed successfully.

**VIVA QUESTIONS**

1. What is Memory Management?
2. Why Use Memory Management?
3. List the memory allocation techniques
4. Define Best fit and its advantage?
5. What is the use of First fit and worst fir methods?

**ASSIGNMENT :**

1. Write a C program to implement compaction technique.

## EX.NO:10 IMPLEMENTATION OF PAGING TECHNIQUE OF MEMORY MANAGEMENT

**AIM**:

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

**ALGORIHTM:**

Step 1: Start the program.

Step 2: Enter the logical memory address.

Step 3: Enter the page table which has offset and page frame. Step 4: The corresponding physical address can be calculate by, PA = [ pageframe* No. of page size ] + Page offset.

Step 5: Print the physical address for the corresponding logical address. Step 6: Terminate the program.



**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
main()
{
        int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;
        int s[10], fno[10][20];
        clrscr();
        printf("\nEnter the memory size -- ");
         scanf("%d",&ms);
        printf("\nEnter the page size -- ");
        scanf("%d",&ps);
        nop = ms/ps;
        printf("\nThe no. of pages available in memory are -- %d ",nop);
        printf("\nEnter number of processes -- ");
        scanf("%d",&np);
        rempages = nop;
         for(i=1;i<=np;i++)
```

```
{

printf("\nEnter no. of pages required for p[%d]-- ",i);
scanf("%d",&s[i]);
if(s[i] >rempages)
{
        printf("\nMemory is Full"); break;
}
rempages = rempages - s[i];
printf("\nEnter pagetable for p[%d] --- ",i);
for(j=0;j<s[i];j++)
        scanf("%d",&fno[i][j]);
}

printf("\nEnter Logical Address to find Physical Address ");
printf("\nEnter process no. and pagenumber and offset -- ");
scanf("%d %d %d",&x,&y, &offset);
if(x>np || y>=s[i] || offset>=ps)
        printf("\nInvalid Process or Page Number or offset");
else
{
        pa=fno[x][y]*ps+offset;
        printf("\nThe Physical Address is -- %d",pa);
}

getch();
}
```

**OUTPUT**

Enter the memory size – 1000

Enter the page size --  100

The no. of pages available in memory are  10

Enter number of processes -- 3

Enter no. of pages required for p[1]-- 4

Enter pagetable for p[1] --- 8 6 9 5

Enter no. of pages required for p[2]-- 5

Enter pagetable for p[2] --- 1 4 5 7 3

Enter no. of pages required for p[3]-- 5

Memory is Full

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

3

60

The Physical Address is -- 760

**RESULT:**

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

**VIVA QUESTIONS**

1. What is the use Memory Management?
2. Define Memory Management Techniques
3. What is Swapping?
4. What is paging?
5. What are the advantages of non-contiguous memory allocation schemes?
6. What is the process of mapping a logical address to physical address with respect to the paging memory management technique?
7. Define the terms – base address, offset?
8. Differentiate between paging and segmentation memory allocation techniques?
9. What is the purpose of page table?
10. Whether the paging memory management technique suffers with internal or external fragmentation problem. Why?

**ASSIGNMENT**

1. Write a C program to simulate two-level paging technique.
2. Write a C program to simulate segmentation memory management technique.

## EX.NO:11A IMPLEMENTATION OF THE FIFO PAGE REPLACEMENT ALGORITHMS

**AIM:**

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

**DESCRIPTION** :

The FIFO Page Replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen . There is not strictly necessary to record the time when a page is brought in. By creating a FIFO queue to hold all pages in memory and by replacing the page at the head of the queue. When a page is brought into memory, insert it at the tail of the queue.

**ALGORITHM:**

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

**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
 main()
{
        int i, j, k, f, pf=0, count=0, rs[25], m[10], n;
        clrscr();
        printf("\n Enter the length of reference string -- ");
```

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

**OUTPUT**

Enter the length of reference string – 20

Enter the reference string -- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter no. of frames -- 3

The Page Replacement Process is –

| | | | |
|---|---|---|---|
| 7 | -1 | -1 | PF No. 1 |
| 7 | 0 | -1 | PF No. 2 |
| 7 | 0 | 1 | PF No. 3 |
| 2 | 0 | 1 | PF No. 4 |
| 2 | 0 | 1 | |
| 2 | 3 | 1 | PF No. 5 |
| 2 | 3 | 0 | PF No. 6 |
| 4 | 3 | 0 | PF No. 7 |
| 4 | 2 | 0 | PF No. 8 |
| 4 | 2 | 3 | PF No. 9 |
| 0 | 2 | 3 | PF No. 10 |
| 0 | 2 | 3 | |
| 0 | 2 | 3 | |
| 0 | 1 | 3 | PF No. 11 |
| 0 | 1 | 2 | PF No. 12 |
| 0 | 1 | 2 | |
| 0 | 1 | 2 | |
| 7 | 1 | 2 | PF No. 13 |
| 7 | 0 | 2 | PF No. 14 |
| 7 | 0 | 1 | PF No. 15 |



The number of Page Faults using FIFO are 15

**RESULT:**

Thus a UNIX C program to implement FIFO page replacement is executed successfully.

**EX.NO:11B IMPLEMENTATION OF LRU PAGE REPLACEMENT ALGORITHM**

**AIM:**

To write UNIX C program a program to implement LRU page replacement algorithm.

**DESCRIPTION:**

The Least Recently Used replacement policy chooses to replace the page which has not been referenced for the longest time. This policy assumes the recent past will approximate the immediate future. The operating system keeps track of when each page was referenced by recording the time of reference or by maintaining a stack of references.

**ALGORITHM:**

1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. Declare counter and stack
6. Select the least recently used page by counter value
7. Stack them according the selection.
8. Display the values
9. Stop the process

**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
main()
{
        int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0, next=1;
         clrscr();
        printf("Enter the length of reference string -- ");
```

```
scanf("%d",&n);
printf("Enter the reference string -- ");
for(i=0;i<n;i++)
{
        scanf("%d",&rs[i]); flag[i]=0;
}
printf("Enter the number of frames -- ");
scanf("%d",&f);
for(i=0;i<f;i++)
{
        count[i]=0;
        m[i]=-1;
}
printf("\nThe Page Replacement process is -- \n");
for(i=0;i<n;i++)
{
        for(j=0;j<f;j++)
        {
                if(m[j]==rs[i])
                {
                        flag[i]=1;
                        count[j]=next;
                         next++;
                }
        }
        if(flag[i]==0)
        {
                if(i<f)
                {
                        m[i]=rs[i];
                         count[i]=next;
                        next++;
```

```
                }
                else
                {
                        min=0;
                        for(j=1;j<f;j++)
                                if(count[min] > count[j])
                                        min=j;
                        m[min]=rs[i];
                        count[min]=next;
                        next++;
                }
                pf++;
        }
        for(j=0;j<f;j++)
                printf("%d\t", m[j]);
        if(flag[i]==0)
                        printf("PF No. -- %d" , pf);
        printf("\n");
    }
    printf("\nThe number of page faults using LRU are %d",pf);
    getch();
}
```



**OUTPUT**

Enter the length of reference string -- 20

Enter the reference string -- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter the number of frames -- 3

The Page Replacement process is --

7 -1 -1 PF No. -- 1

7 0 -1 PF No. -- 2

7 0 1 PF No. -- 3

2 0 1 PF No. -- 4

| | | | |
|---|---|---|---|
| 2 | 0 | 1 | |
| 2 | 0 | 3 | PF No. -- 5 |
| 2 | 0 | 3 | |
| 4 | 0 | 3 | PF No. -- 6 |
| 4 | 0 | 2 | PF No. -- 7 |
| 4 | 3 | 2 | PF No. -- 8 |
| 0 | 3 | 2 | PF No. -- 9 |
| 0 | 3 | 2 | |
| 0 | 3 | 2 | |
| 1 | 3 | 2 | PF No. -- 10 |
| 1 | 3 | 2 | |
| 1 | 0 | 2 | PF No. -- 11 |
| 1 | 0 | 2 | |
| 1 | 0 | 7 | PF No. -- 12 |
| 1 | 0 | 7 | |
| 1 | 0 | 7 | |

The number of page faults using LRU are 12



**RESULT:**

Thus a UNIX C program to implement LRU page replacement is executed successfully.

**EX.NO:11C IMPLEMENTATION OF LFU PAGE REPLACEMENT ALGORITHM**

**AIM**:

To write a program in C to implement LFU page replacement algorithm.

**ALGORITHM**

Step1: Start the program

Step2: Declare the required variables and initialize it.

Step3; Get the frame size and reference string from the user

Step4: Keep track of entered data elements

Step5: Accommodate a new element look for the element that is not to be used in frequently replace.

Step6: Count the number of page fault and display the value

Step7: Terminate the program

**PROGRAM**



```
#include<stdio.h>
 #include<conio.h>
main()
{
        int rs[50], i, j, k, m, f, cntr[20], a[20], min, pf=0;
        clrscr();
        printf("\nEnter number of page references -- ");
         scanf("%d",&m);
        printf("\nEnter the reference string -- ");
        for(i=0;i<m;i++)
                scanf("%d",&rs[i]);
        printf("\nEnter the available no. of frames -- ");
         scanf("%d",&f);
        for(i=0;i<f;i++)
        {
```

```
        cntr[i]=0;
        a[i]=-1;
    }
    Printf("\nThe Page Replacement Process is – \n");
    for(i=0;i<m;i++)
    {
        for(j=0;j<f;j++)
            if(rs[i]==a[j])
            {
                cntr[j]++;
                 break;
            }
        if(j==f)
        {
            min = 0;
             for(k=1;k<f;k++)
                if(cntr[k]<cntr[min])
                    min=k;
            a[min]=rs[i];
            cntr[min]=1;
            pf++;
        }
        printf("\n");
        for(j=0;j<f;j++)
            printf("\t%d",a[j]);
        if(j==f)
            printf("\tPF No. %d",pf);

    }
    printf("\n\n Total number of page faults -- %d",pf);
    getch();
}
```

**OUTPUT**

Enter number of page references -- 10

Enter the reference string -- 1 2 3 4 5 2 5 2 5 1 4 3

Enter the available no. of frames 3

The Page Replacement Process is –

| | | | |
|---|---|---|---|
| 1 | -1 | -1 | PF No. 1 |
| 1 | 2 | -1 | PF No. 2 |
| 1 | 2 | 3 | PF No. 3 |
| 4 | 2 | 3 | PF No. 4 |
| 5 | 2 | 3 | PF No. 5 |
| 5 | 2 | 3 | |
| 5 | 2 | 3 | |
| 5 | 2 | 1 | PF No. 6 |
| 5 | 2 | 4 | PF No. 7 |
| 5 | 2 | 3 | PF No. 8 |

Total number of page faults -- 8



**RESULT:**

Thus the C programs to implement LFU page replacement algorithm was executed successfully.

**VIVA QUESTIONS:**

1. What is the purpose of page replacement?
2. Define page fault?
3. Which replacement algorithms suffers from Belady's anomaly?
4. Reference bit is used in which page replacement algorithm?
5. What is LRU page replacement?
6. Define optimal page replacement.
7. Define the concept of thrashing? What is the scenario that leads to the situation of thrashing?

**ASSIGNMENT:**

1.Write a C program to simulate LRU-approximation page replacement algorithm?

a. Additional-Reference bits algorithm

b. Second-chance algorithm

**EX.NO:12A SEQUENTIAL FILE ALLOCATION**

**AIM**:

To implement sequential file allocation technique.

**ALGORITHM:**

Step 1: Start the program.

Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations to each in sequential order.

a). Randomly select a location from available location s1= random(100); b). Check whether the required locations are free from the selected location. c). Allocate and set flag=1 to the allocated locations.

Step 5: Print the results fileno, length , Blocks allocated. Step 6: Stop the progr

**PROGRAM**

```
#include<stdio.h>
main()
{
        int f[50],i,st,j,len,c,k;
        clrscr();
        for(i=0;i<50;i++)
                f[i]=0;
        X:
        printf("\n Enter the starting block & length of file");
        scanf("%d%d",&st,&len);
        for(j=st;j<(st+len);j++)
                if(f[j]==0)
                {
                        f[j]=1;
                        printf("\n%d->%d",j,f[j]);
                }
                else
```

```
                {
                        printf("Block already allocated");
                         break;
                }
        if(j==(st+len))
                printf("\n the file is allocated to disk");
        printf("\n if u want to enter more files?(y-1/n-0)");
        scanf("%d",&c);
         if(c==1)
                goto X;
        else
                exit();
getch();
}
```

**OUTPUT**

Output: Enter the starting block & length of file 4 10

4->1

5->1

6->1

7->1

8->1

9->1

10->1

11->1

12->1

13->1

The file is allocated to disk

If you want to enter more files? (Y-1/N-0)

**RESULT :**

Thus the program to implement the Sequential file allocation was executed successfully.

**EX.NO:12B LINKED FILE ALLOCATION**

**AIM:**

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

**ALGORITHM:**

Step 1: Start the Program

Step 2: Get the number of files.

Step 3: Allocate the required locations by selecting a location randomly

Step 4: Check whether the selected location is free.

Step 5: If the location is free allocate and set flag =1 to the allocated locations.

Step 6: Print the results file no, length, blocks allocated.

Step 7: Stop the execution

**PROGRAM:**

```
#include<stdio.h>
 main()
{
        int f[50],p,i,j,k,a,st,len,n,c;
        clrscr();
        for(i=0;i<50;i++)
                f[i]=0;
        printf("Enter how many blocks that are already allocated");
        scanf("%d",&p);
        printf("\nEnter the blocks no.s that are already allocated");
         for(i=0;i<p;i++)
        {
                scanf("%d",&a);
                f[a]=1;
        }
        X: printf("Enter the starting index block & length");
        scanf("%d%d",&st,&len);
```

```
        k=len;
        for(j=st;j<(k+st);j++)
        {
                if(f[j]==0)
                 {
                        f[j]=1;
                        printf("\n%d->%d",j,f[j]);
                }
                else
                {
                        printf("\n %d->file is already allocated",j); k++;
                }
        }
        printf("\n If u want to enter one more file? (yes-1/no-0)");
        scanf("%d",&c);
        if(c==1)
                goto X;
        else
                exit();
getch( );
}
```



**OUTPUT:**

```
Enter how many blocks are already allocated 3
Enter the blocks no's that are already allocated 4 7 9
 Enter the starting index block & length 3 7
3-> 1
4-> File is already allocated 5->1
6->1
7-> File is already allocated 8->1
9-> File is already allocated 10->1
11->1
12->1
If u want to enter one more file? (yes-1/no-0)
```

**RESULT:**

Thus the program to implement the linked file allocation was executed successfully

**EX.NO:12C INDEXED FILE ALLOCATION**

**AIM**:

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

**ALGORITHM:**

Step 1: Start the Program

Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations by selecting a location randomly. Step 5: Print the results file no,length, blocks allocated.

Step 6: Stop the execution.

**PROGRAM**

```
#include<stdio.h>
int f[50],i,k,j,inde[50],n,c,count=0,p;
 main()
{
        clrscr();
        for(i=0;i<50;i++)
                f[i]=0;
        x: printf("enter index block\t");
        scanf("%d",&p);
        if(f[p]==0)
        {
                f[p]=1;
                printf("enter no of files on index\t"); s
                canf("%d",&n);
        }
        else
        {
                printf("Block already allocated\n"); goto x;
```

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



**OUTPUT:**

Enter index block 9

Enter no of files on index 3 1 2 3

Allocated

File indexed 9-> 1:1

9-> 2:1

9->3:1

Enter 1 to enter more files and 0 to exit.

**RESULT :**

Thus the program to implement the indexed file allocation was executed successfully

**VIVA QUESTIONS:**

1. Define file?
2. What are the different kinds of files?
3. What is the purpose of file allocation strategies?
4. Identify ideal scenarios where sequential, indexed and linked file allocation strategies are most appropriate?
5. What are the disadvantages of sequential file allocation strategy?
6. What is an index block?
7. What is the file allocation strategy used in UNIX?

**ASSIGNMENT:**

1. Write a C program to simulate a two-level index scheme for file allocation?

**EX.NO: 13 MULTI-LEVEL QUEUE SCHEDULING**

**AIM:**

Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. The priority of each process ranges from 1 to 3. Use fixed priority scheduling for all the processes.

**DESCRIPTION:**

Multi-level queue scheduling algorithm is used in scenarios where the processes can be classified into groups based on property like process type, CPU time, IO access, memory size, etc. In a multi-level queue scheduling algorithm, there will be 'n' number of queues, where 'n' is the number of groups the processes are classified into. Each queue will be assigned a priority and will have its own scheduling algorithm like round-robin scheduling or FCFS. For the process in a queue to execute, all the queues of priority higher than it should be empty, meaning the process in those high priority queues should have completed its execution. In this scheduling algorithm, once assigned to a queue, the process will not move to any other queues.

**PROGRAM:**



```
main()
{
        int p[20],bt[20], su[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 of Process %d --- ", i);
                scanf("%d",&bt[i]);
                printf("System/User Process (0/1) ? --- ");
                scanf("%d", &su[i]);
```

```
        }
        for(i=0;i<n;i++)
                for(k=i+1;k<n;k++)
                        if(su[i] > su[k])
                        {
                                temp=p[i]; p[i]=p[k]; p[k]=temp;
                                temp=bt[i]; bt[i]=bt[k]; bt[k]=temp;
                                temp=su[i]; su[i]=su[k]; su[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\t SYSTEM/USER PROCESS \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],su[i],bt[i],wt[i],tat[i]);
        printf("\nAverage Waiting Time is --- %f",wtavg/n);
        printf("\nAverage Turnaround Time is --- %f",tatavg/n);
        getch();
}
```



**OUTPUT**

```
Enter the number of processes --- 4

Enter the Burst Time of Process 0 --- 3
```

System/User Process (0/1) ? --- 1

Enter the Burst Time of Process 1 --- 2

System/User Process (0/1) ? --- 0

Enter the Burst Time of Process 2 --- 5

System/User Process (0/1) ? --- 1

Enter the Burst Time of Process 3 --- 1

System/User Process (0/1) ? --- 0

| PROCESS | USER PROCESS | BURST TIME | WAITING TIME | TURNAROUND TIME |
|---|---|---|---|---|
| 1 | 0 | 2 | 0 | 2 |
| 3 | 0 | 1 | 2 | 3 |
| 2 | 1 | 5 | 3 | 8 |
| 0 | 1 | 3 | 8 | 11 |

Average Waiting Time is --- 3.250000

Average Turnaround Time is --- 6.000000

**RESULT:**

The Multilevel Scheduling algorithm has been implemented in C.



**VIVA QUESTIONS**

1. What is multi-level queue CPU Scheduling?

2. Differentiate between the general CPU scheduling algorithms like FCFS, SJF etc and multi-level queue CPU Scheduling?

3. What are CPU-bound I/O-bound processes?

4. What are the parameters to be considered for designing a multilevel feedback queue scheduler?

5. Differentiate multi-level queue and multi-level feedback queue CPU scheduling algorithms?

6. What are the advantages of multi-level queue and multi-level feedback queue CPU scheduling algorithms?

**ASSIGNMENT:**

1.Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher

priority than user processes. Consider each process priority to be from 1 to 3. Use priority scheduling for the processes in each queue

**EX.NO: 14 DISK SCHEDULING ALGORITHMS**

**AIM:**

Write a C program to simulate disk scheduling algorithms

a) FCFS b) SCAN c) C-SCAN

**DESCRIPTION**

One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth. Both the access time and the bandwidth can be improved by managing the order in which disk I/O requests are serviced which is called as disk scheduling. The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service. In the SCAN algorithm, the disk arm starts at one end, and moves towards the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk. C-SCAN is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip



**PROGRAM**

**FCFS DISK SCHEDULING ALGORITHM**

```
#include<stdio.h>
 main()
{
        int t[20], n, I, j, tohm[20], tot=0;
        float avhm;
        clrscr();
        printf("enter the no.of tracks");
        scanf("%d",&n);
        printf("enter the tracks to be traversed");
        for(i=2;i<n+2;i++)
```

```
        scanf("%d",&t*i+);
    for(i=1;i<n+1;i++)
    {
        tohm[i]=t[i+1]-t[i];
        if(tohm[i]<0)
            tohm[i]=tohm[i]*(-1);
    }
    for(i=1;i<n+1;i++)
        tot+=tohm[i];
    avhm=(float)tot/n;
    printf("Tracks traversed\tDifference between tracks\n"); for(i=1;i<n+1;i++)
        printf("%d\t\t\t%d\n",t*i+,tohm*i+);
     printf("\nAverage header movements:%f",avhm);
    getch();
}
```

**OUTPUT**



Enter no.of tracks:9

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

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

Average header movements:30.888889

**SCAN DISK SCHEDULING ALGORITHM**

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

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



**OUTPUT**

Enter no.of tracks:9

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

Tracks traversed Difference between tracks

150 50

160 10

1841 24

90 94

70 20

60 10

58 2

55 3

18 37

Average header movements: 27.77

**C-SCAN DISK SCHEDULING ALGORITHM**

```
#include<stdio.h>
main()
{
        int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;
        clrscr();
        printf("enter the no of tracks to be traveresed");
        scanf("%d'",&n);
        printf("enter the position of head");
        scanf("%d",&h);
        t[0]=0;t[1]=h;
        printf("enter total tracks");
        scanf("%d",&tot);
        t[2]=tot-1;
        printf("enter the tracks");
        for(i=3;i<=n+2;i++)
                scanf("%d",&t[i]);
        for(i=0;i<=n+2;i++)
                for(j=0;j<=(n+2)-i-1;j++)
                        if(t[j]>t[j+1])
                        {
                                temp=t[j];
                                t[j]=t[j+1];
```

```
                        t[j+1]=temp;
                }
for(i=0;i<=n+2;i++)
        if(t[i]==h)
                j=i;break;
 p=0;
while(t[j]!=tot-1)
{
        atr[p]=t[j];
        j++;
        p++;
}
atr[p]=t[j];
 p++;
i=0;
while(p!=(n+3) && t[i]!=t[h])
{
        atr[p]=t[i];
         i++;
        p++;
}
for(j=0;j<n+2;j++)
{
if(atr[j]>atr[j+1])
        d[j]=atr[j]-atr[j+1];
else
        d[j]=atr[j+1]-atr[j];
sum+=d[j];
}
printf("total header movements%d",sum);
```

```
        printf("avg is %f",(float)sum/n);
        getch();
}
```

**OUTPUT**

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

Enter starting position : 100

| Tracks traversed | Difference Between tracks |
|---|---|
| 150 | 50 |
| 160 | 10 |
| 184 | 24 |
| 18 | 240 |
| 55 | 37 |
| 58 | 3 |
| 60 | 2 |
| 70 | 10 |
| 90 | 29 |

Average seek time : 35.7777779



**RESULT:**

Thus the program to implement disk Scheduling algorithm has been executed and verified

**VIVA QUESTIONS:**

1. What is disk scheduling?
2. List the different disk scheduling algorithms?
3. Define the terms – disk seek time, disk access time and rotational latency?
4. Define sequential file allocation?
5. What is the use of indexed file allocation?
6. What is the advantages if linked allocation?
7. What is the advantage of C-SCAN algorithm over SCAN algorithm?
8. Which disk scheduling algorithm has highest rotational latency? Why?

**ASSIGNMENT:**

1. Write a C program to implement SSTF disk scheduling algorithm?

## EX.NO.15 DINING-PHILOSOPHERS PROBLEM.

**AIM**:

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

**DESCRIPTION**

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



**PROGRAM**

```
int tph, philname[20], status[20], howhung, hu[20], cho;

main()

{

        int i;

        clrscr();

        printf("\n\nDINING PHILOSOPHER PROBLEM");

        printf("\nEnter the total no. of philosophers: ");

        scanf("%d",&tph);

        for(i=0;i<tph;i++)

        {

                philname[i] = (i+1);
```

```
        status[i]=1;
}
printf("How many are hungry : ");
 scanf("%d", &howhung);
 if(howhung==tph)
{
        printf("\nAll are hungry..\nDead lock stage will occur");
        printf("\nExiting..");
}
else
{
        for(i=0;i<howhung;i++)
        {
                printf("Enter philosopher %d position: ",(i+1));
                scanf("%d", &hu[i]);
                status[hu[i]]=2;
        }
        do
        {
        printf("1.One can eat at a time\t2.Two can eat at a
        time\t3.Exit\nEnter your choice:");
        scanf("%d", &cho);
        switch(cho)
        {
                case 1:  one();
                         break;
                case 2:  two();
                         break;
                case 3: exit(0);
                default: printf("\nInvalid option..");
```

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

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

**OUTPUT**

DINING PHILOSOPHER PROBLEM

Enter the total no. of philosophers: 5

How many are hungry : 3

Enter philosopher 1 position: 2

Enter philosopher 2 position: 4

Enter philosopher 3 position: 5

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

Enter your choice: 1

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

P 3 is waiting P 5 is waiting P 0 is waiting

P 5 is granted to eat P 5 is waiting

P 0 is waiting

P 0 is granted to eat P 0 is waiting

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

Allow two philosophers to eat at same time combination 1

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

combination 2

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

combination 3

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

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

Enter your choice: 3

**RESULT:**

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

**VIVA QUESTIONS**:

1. Differentiate between a monitor, semaphore and a binary semaphore?

2. Define clearly the dining-philosophers problem?

3. Identify the scenarios in the dining-philosophers problem that leads to the deadlock situations?

**ASSIGNMENT:**

1. Write a C program to simulate readers-writers problem using monitors?