OPERATING SYSTEM LABORATORY

(V semester of B.Tech)

As per the curriculam and syllabus

Of

Bharath Institute of Higher Education & Research

PREPARED BY

DR.A.KUMARAVEL

NEW EDITION

Department of information technology

Bharath
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Declared as Deemed - to - be - University under section 3 of UGC Act 1956)
ACCREDITED WITH 'A' GRADE BY NAAC

OPERATING SYSTEM

(V semester of B.Tech)

As per the curricullam and syllabus

Of

Bharath Institute of Higher Education & Research

PREPARED BY
DR.A.KUMARAVEL

Department of information technology

Bharath
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Declared as Deemed - to - be - University under section 3 of UGC Act 1956)
ACCREDITED WITH 'A' GRADE BY NAAC

SCHOOL OF COMPUTING

DEPARTMENT OF INFORMATION TECHNOLOGY

LAB MANUAL

SUBJECT NAME: **OPERATING SYSTEM LABORATORY**

SUBJECT CODE: **U20ITCJ07**

Regulation - 2020

VISION AND MISSION OF THE INSTITUTE

VISION

"Bharath Institute of Higher Education & Research (BIHER) envisions and constantly strives to provide an excellent academic and research ambience for students and members of the faculties to inherit professional competence along with human dignity and transformation of community to keep pace with the global challenges so as to achieve holistic development."

MISSION

- To develop as a Premier University for Teaching, Learning, Research and Innovation on par with leading global universities.
- To impart education and training to students for creating a better society with ethics and morals.
- To foster an interdisciplinary approach in education, research and innovation by supporting lifelong professional development, enriching knowledge banks through scientific research, promoting best practices and innovation, industry driven and institute-oriented cooperation, globalization and international initiatives.
- To develop as a multi-dimensional institution contributing immensely to the cause of societal advancement through spread of literacy, an ambience that provides the best of international exposures, provide health care, enrich rural development and most importantly impart value-based education.
- To establish benchmark standards in professional practice in the fields of innovative and emerging areas in engineering, management, medicine, dentistry, nursing, physiotherapy and allied sciences.
- To imbibe human dignity and values through personality development and social service activities.

VISION AND MISSION OF THE DEPARTMENT

VISION

To be an excellence in education and research in Information Technology producing global scholars for improvement of the society

MISSION

- To provide sound fundamentals, and advances in Information Technology, Software Engineering, data Communications and Computer Applications by offering world class curriculum.
- To create ethically strong leaders and expert for next generation IT.
- To nurture the desire among faculty and students from across the globe to perform outstanding and impactful research for the benefit of humanity and, to achieve meritorious and significant growth.

PROGRAM EDUCATIONAL OBJECTIVES (PEO)

The Program Educational Objectives (PEOs) of Information technology are listed below: The graduate after 3-5 years of programme completion will

PEO1: PREPARATION
To provide students with sound fundamental in Mathematical, Scientific and Engineering fundamentals necessary to formulate, analyse, and comprehend the fundamental concepts essential to articulate, solve and assess engineering problems and to prepare them for research & development and higher learning.

PEO2: CORE COMPETENCE
To apply critical reasoning, quantitative, qualitative, designing and programming skills, to identify, solve problems and to analyze the experimental evaluations, and finally making appropriate decisions along with knowledge of computing principles and applications and be able to integrate this knowledge in a variety of industry and inter-disciplinary setting.

PEO3: PROFESSIONALISM
To broaden knowledge to establish themselves as creative practicing professionals, locally and globally, in fields such as design, development, problem solving to production support in software industries and R&D sectors.

PEO4: SKILL
To provide better opportunity to become a future researchers / scientist with good communication skills so that they may be both good team-members and leaders with innovative ideas for a sustainable development.

PEO5: ETHICS
To be ethically and socially responsible solution providers and entrepreneurs in Computer Science and other engineering discipline.

PROGRAMME OUTCOMES

PO 1	**ngineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	**oblem Analysis:** Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
PO 3	**sign/Development of Solutions:** Design solutions for complex engineering problems 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.
PO 4	**nduct Investigations of Complex Problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions for complex problems.
PO 5	**odern Tool Usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
PO 6	**e Engineer and Society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO 7	**vironment and Sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO 8	**hics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO 9	**dividual and Team Work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO 10	**mmunication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO 11	**oject Management and Finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO 12	**fe-long Learning:** Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

PROGRAMME SPECIFIC OUTCOME

PSO 1	**Programming Design :** Design and develop algorithm for real life problems using latest technologies and solve it by using computer programming languages and database technologies .
PSO 2	**IT Business Scalable Design :** Analyze and recommend computing infrastructures and operations requirements and Simulate and implement information networks using configurations, algorithms, suitable protocol and security for valid and optimal connectivity.
PSO 3	**Intelligent Agents Design :** Design and execute projects for the development of data modeling, data analytics and knowledge representation in various domain.

U20ITCJ07 – OPERATING SYSTEMS

PART-A INTRODUCTION OF THE COURSE

Course Code	U20ITCJ07	L	T	P	C
		2	0	2	3
Course Title	OPERATINGSYSTEMS				
Course Category	Professional Core (C)	Contact Hrs	45		
Pre-requisite	U20CSCT01	Co- Requisite			
Name of the Course Co-ordinator	Ms. C. ANURADHA				
Course offering Dept/School	CSE				

Course Objective and Summary

- To understand how an operating system controls the computing resources and provide services to the users
- To understand the operating system functions, design and implementation

Course Outcomes (COs)

CO1	Illustrate the basic concepts, functionalities and structure of Operating System
CO2	Describe the concepts of process, threads, process scheduling and to clarify interprocess communication, memory management, file and disk management methods.
CO3	Solve the process synchronization, mutual exclusion, deadlock and memory management problems
CO4	Implement the algorithms for process and disk scheduling and memory management.
CO5	Analyze algorithms of process and disk scheduling and memory management.
CO6	Evaluate process synchronization, process scheduling, memory management and disk scheduling algorithms.

Mapping / Alignment of COs with PO & PSO

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3														
CO2	3													3	
CO3	3														
CO4	3														
CO5		3													
CO6				3											

(Tick mark or level of correlation: 3-High, 2-Medium, 1-Low)

Index

S. No.	Date	Name of the Experiment	Pg No.	Signature
1	28/4/22	WORKING WITH BASIC UNIX/ LINUX COMMANDS		
2	6/5/22	SHELL PROGRAMMING a) BIGGEST OF THREE NUMBERS b) FACTORIAL OF A NUMBER c) MULTIPLICATION TABLE d) SIMPLE ADD FUNCTION e) FIBONACCI SERIES f) SUM OF n NUMBERS		
3	20/5/22	SYSTEM CALLS a) PROGRAM USING fork() b) PROGRAM USING getpid(), getppid() c) PROGRAM USING opendir() readdir() closedir() d) PROGRAM USING exec() e) PROGRAM USING wait() exec() f) PROGRAM USING open() read() write()		
4	27/5/22	NON PRE-EMPTIVE CPU SCHEDULING ALGORITHMS a) FIRST COME FIRST SERVED b) SHORTEST JOB FIRST c) PRIORITY SCHEDULING d) ROUND ROBIN SCHEDULING		
5	3/6/22	IMPLEMENTATION OF PROCESS SYNCHRONIZATION USING SEMAPHORE a) PRODUCER CONSUMER PROBLEM b) DINING PHILOSOPHER PROBLEM		
6	10/6/22	DEADLOCK AVOIDANCE		
7	17/6/22	MEMORY MANAGEMENT TECHNIQUES a) MFT b) MVT		
8	24/6/22	MEMORY ALLOCATION TECHNIQUES a) Worst-Fit b) Best – fit c) First Fit		

9	1/7/22	VIRTUAL MEMORY MANAGEMENT TECHNIQUES		

10	8/7/22	FILE ALLOCATION TECHNIQUES a) SEQUENTIAL FILE ALLOCATION b) LINKED FILE ALLOCATION c) INDEXED FILE ALLOCATION		
11	8/7/22	DISK SCHEDULING ALGORITHM a) FCFS b) SCAN		

Exp-1	WORKING WITH BASIC UNIX/ LINUX COMMANDS
Date - 28/4/22	

AIM : To work with basic UNIX commands.

COMMANDS :

1. DATE COMMANDS

SYNTAX : date
USES : Used to display system date and time.
OUTPUT : Thu May 4 15:52:02 IST 2006

OPTIONS :

SYNTAX : date+%m
USES : Used to print the month in number.
OUTPUT : 05

SYNTAX : date+%h
USES : Used to print the month in name.
OUTPUT : May

SYNTAX : date+%y
USES : Used to print the last 2 digits of year.
OUTPUT : 06

SYNTAX : date+%M
USES : Used to displays the time in minute.
OUTPUT : 53

SYNTAX : date+%H
USES : Used to displays the time in hour.
OUTPUT : 15

SYNTAX : date+%a
USES : Used to displays the aggregated form of the day.
OUTPUT : Thu

SYNTAX : date+%r
USES : Used to displays the time in AM or PM.
OUTPUT : 03:54:43PM

SYNTAX : date+%T
USES : Used to displays the full time in the format of HH:MM:SS.
OUTPUT : 15:55:58

2. CALENDER :

SYNTAX : cal
USES : It displays the current month calendar.
OUTPUT : May 2006

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			

SYNTAX : cal1998
USES : It displays the mentioned year calendar.
OUTPUT : It will displays the given year calendar.

3. ECHO

SYNTAX : echo " UNIX"
USES : Used to displaying the given text.
OUTPUT : UNIX

TYPING CONTENT IN MORE THAN ONE LINE

SYNTAX : echo " This command
> line exceeds
> a single line"
OUTPUT : This command line exceeds a single line
USES : Used to display the multiple line as we entered while executing the command.

SYNTAX : echo This command \
> line exceeds \
> a single line"
OUTPUT : This command line exceeds a single line.
USES : The Backslash character at the end of each line (followed by pressing the ENTER key) informs the shell that the user wants to continue the command on the next line. It is used to display the multiple lines into single line.

4. BC

SYNTAX : bc
USES : Used to perform simple mathematical calculations. By default this command accepts decimal numbers and also perform mathematical calculations on octal and hexadecimal numbers.
OUTPUT : 1) 5+3
8
2) 1.1+2.2
3.3
3) 3.3-1.2

2.1

4) 5.6*4.1
22.9

5) 8.4/3.2
2

6) 3-1
2

7) 4*5
20

5. WHO

SYNTAX : who
USES : Used to display the login details for all users using the UNIX systems.
OUTPUT : student pts/1 May 4 15:50(192.168.2.3) student pts/11 May 4 10:47(192.168.2.19)
root :0 May 4 14:52(console)

6. WHO AM I

SYNTAX : whoami
USES : Used to displays the login details of the current users of the system, who invokes the command.
OUTPUT : student pts/1 May 4 15:50(192.168.2.3)

7. TTY

SYNTAX : tty (tele type)
USES : It gives the file name of the terminal that you are using , tell you the device name and the name of the terminal that you currently working in.
OUTPUT : /dev/pts/1

8. MAN

SYNTAX : man any command
USES : It offers the online help facilities and it gives all details for the particular commands that which user can type file and the directory commands.

DIRECTORY COMMANDS

9. MAKE DIRECTORY

SYNTAX : mkdir directoryname
USES : Used to create a new directory.
OUTPUT : mkdir jeeva

10. CHANGE DIRECTORY

SYNTAX : cd directoryname
USES : Used to change from one working directory to another directory specified.
OUTPUT : cd jeeva

11. REMOVE DIRECTORY

SYNTAX : rmdir
USES : Used to remove the directory.
OUTPUT : rmdir jeeva

12. PWD

SYNTAX : pwd
USES : Used to display the path that we are working.
OUTPUT : /home/student/vs

FILE & REDIRECTION COMMANDS

13. CAT

SYNTAX : cat > filename
USES : Used to create a new file & insert a content into that file. Press CTRL + D to exit from that file.
OUTPUT : cat>sample
flower
jasmin
e rose
Press ctrl + d

14.

SYNTAX : cat filename
USES : Used to display the content of that file.
OUTPUT : cat sample
flower
jasmin
e rose

15.

SYNTAX : cat < filename1 >filename2
USES : It will copy the content of filename1 to filename2 .the content of filename to is erased.
OUTPUT : cat sample1
a
b
c

cat <sample > sample1
flower
jasmine
rose
rose d

cat sample1
flower
jasmine

16. **SYNTAX** : cat filename1 filename2 >filename3

USES : The output of filename1 and filename2 are concatenated and stored in the filename3 , and it is not displayed in the terminal. Here filename3 is already exits. To view the contents of the file use cat command.

OUTPUT :

catsample1	catsample2	catsample3
flower	ab	flower
jasmine	bc	jasmine rose
rose	d	
	ef	
a		
b		
bc		
df		

17. **SYNTAX** : any command > filename

USES : The output of any command is stored in the file(filename) and it not displayed in the command prompt. To view the output use cat command.

EXAMPLE :

SYNTAX : who >sample

USES : The result of who am i is stored in sample.to view the contents use cat sample.

OUTPUT : who> sample4
cat sample4

student pts/1	May 4 15:50(192.168.2.3)
student pts/11	May 4 10:47(192.168.2.19)
root :0	May 4 14:52(console)

APPEND THE FILE CONTENTS

18. **SYNTAX** : cat filename1 >>filename2

USES : Used to append the contents of filename1 to filename2.

OUTPUT :

catsample1	catsample2	cat sample1 >> sample2
flower	ab	cat
jasmine	bc	sample3
rose	d ef	ab
		b
		c
		d
		ef
		flower
		jasmin
		e
		rose

19. **SYNTAX** : cat filename1 filename2 >>filename3
USES : The output of filename1 and filename2 are append in the

filename3 , and it is not displayed in the terminal. Here filename3 is not already exits. It is a new file .so contents are not over written. To view the contents of the file use cat command.

RENAMING THE FILE

20. **SYNTAX** : mv source filename destination filename
USES : Used to rename the file.

LIST OUT COMMANDS

21. **SYNTAX** : ls
USES : Used to list out all files in a directory.

22. **SYNTAX** : ls-a
USES : Used to list out all files including hidden files (Files that begin with (. , ..)

23. **SYNTAX** : ls-i
USES : Used to list out all files with its i-node number in the first column.

24. **SYNTAX** : ls -r
USES : Used to list out all files in reverse alphabetical order.

25. **SYNTAX** : ls -t
USES : Used to list out all files in the order of their last modification time.

26. **SYNTAX** : ls -u
USES : Used to list out all files in the order of their last access time.

27. **SYNTAX** : ls-l
USES : Used to list out all files in long format (one entry per line) ,giving its mode, number of links, owner, group , size in bytes, the time that each file was last modified.

28. **SYNTAX** : ls -lt
USES : Used to list out all files in long format with their last modification time.

FILTER COMMANDS

29. **SYNTAX** : head -n filename
USES : Used to display the Top ‘n’ lines of the file.

30. **SYNTAX** : tail -n filename
USES : Used to display the Bottom ‘n’ lines of the file.

31. **SYNTAX** : more filename
USES : To see the content of the filename on the screen one page at a time.

WORD COUNT COMMAND

32. **SYNTAX :** wc filename
USES : It will display the no of lines ,no of words and used to display no of characters.

33. **SYNTAX** : wc -l filename
USES : Used to display no of lines in a filename.

34. **SYNTAX** : wc -w filename
USES : Used to display no of words in a filename.

35. **SYNTAX** : wc -c filename
USES : Used to display no of Characters in a filename.

COMMON COMMAND

36. **SYNTAX** : comm. filename1filenmae2
USES : This command each line of first file with its corresponding line in the second file. Output contains 3 columns. First column contains lines unique to filename1 . Second column contains lines unique to filename2.Third column contains lines common to both.

EXAMPLE :

$cat n1	$cat nithi
sridhar	sridhar
ramamoorthy	vikky
mahalashmi	vimal
jegan	prabhu

OUTPUT :

```
$comm n1 nithi
sridhar
vikky
vima
l
```

OPTIONS :

37. **SYNTAX** : comm -1filename
USES : It doesn't include the first column in output.

38. **SYNTAX** : comm -2filename
USES : It doesn't include the second column inoutput.

39. **SYNTAX** : comm -3filename
USES : It doesn't include the third column inoutput.

COMPARE COMMAND

40. **SYNTAX** : cmp filename1filename2
USES : Two files are compared byte by byte and the location of the first mismatch is echoed in
OUTPUT : thescreen.
cmp sample1 sample5
sample1 sample5 differ: byte 7, line 1

41. **SYNTAX** : cmp –l file1file2
USES : This option gives the detailed list of the byte number and the differing bytes in octal for each character that differ in bothfiles.
OUTPUT : cat sample1 catsample2
flower flower
jasmine carrot
rose papaya
banana

cmp -l	sample1	sample5
7	12	40
8	152	12
9	141	143
10	163	141
11	155	162
12	151	162
13	156	157
14	145	164
16	162	160
17	157	141

cmp: EOF on sample1

CUT COMMAND

42. **SYNTAX** : cut -d "any character " -f2filename.
USES : Used to cut the character from first to the specified character from the file.
OUTPUT : catsample1
flower
jasmin
e rose
cut -d "w" -f2 sample1
er
jasmin
e rose

43. **SYNTAX** : cut –c 1-3filename
USES : It will command the character which are defined in numbers and it will displays the character up to that specified number.
OUTPUT : catsample5 cut -c 4sample5

Flower	x
Carrot	r
Papaya	a
Banana	a

44. COPY COMMAND :

SYNTAX : cp source filename destination filename
USES : Used to copy the source file contents to destination file contents.

45. PASTE COMMAND :

SYNTAX : paste file1file2
USES : This command merges the contents of 2 files in into a single file. It reads a line from file in the file list specified and combines them into a single file.
OUTPUT : paste sample1 sample2

flower	ab
jasmine	bc
rose	d
	ef
	flower
	jasmin
	e rose

UNIQUE COMMAND

46. SYNTAX : uniq filename
USES : Used to display the unique lines present in thefile.
OUTPUT :

$ cat n1	$ uniq n1
Sridhar	Sridhar
Ramamoorth	Ramamoorth
y	y
Mahalashmi	Mahalashmi
Jegan	jegan
Prhb	prhbu
u	
prhb	
u	

OPTIONS :

47. SYNTAX : uniq - ufilename
USES : Used to display only unique lines present in
OUTPUT : thefile.

$ uniq –u n1
sridhar
ramamoorthy
mahalashmi
jegan

48. SYNTAX : uniq - dfilename
USES : Used to display the duplicate lines in thefile.

OUTPUT : $ uniq –d n1
prhbu

49. **SYNTAX** : uniq - cfilename
USES : Used to count the duplicate lines in thefile.
OUTPUT : $ uniq –c n1
sridhar
ramamoorthy
mahalashmi
jegan

GREP COMMAND

SYNTAX : $grep ` ` filename
USES : This command is used to search for a particular pattern from a file or from the standard input and display those lines on the standard output.
OPTIONS :

50. **SYNTAX** : $grep –v ` `filename
USES : displays only those lines that do not match the patternspecified.
OUTPUT : name : Bahirathi

51. **SYNTAX** : $grep –c ` `filename
USES : displays only the count of those lines, which match the pattern, specified.
OUTPUT : $grep –c ` `alamu
3

52. **SYNTAX** : $grep –n ` `filename
USES : displays those lines, which match the pattern, specified along with line number at the beginning of the line
OUTPUT : $grep –n ` ` alamu
1:rathi
3:rgi
5:vrs

53. **SYNTAX** : $grep i ` `filename
USES : displays those lines, which match the pattern specified ignoring the case distinction.
OUTPUT : $grep –i ` ` alamu
rathi
raji
vrs

54. PIPE COMMAND : A pipe is a mechanism, which takes the output of the command as its input for the next command .

SYNTAX : Command |command
OUTPUT : $ who |wc -l

55. TEE COMMAND :

SYNTAX : command|tee
USES : used to join pipes and make copies of input.
OUTPUT : $who |tee rs | wc -l

WILD CARDS

56. *

SYNTAX : $ls*
USES : used in representing any number of characters when used in the prefix or suffix.

57. ?

SYNTAX : $ls ?
USES : this character is use to represent one character wither in prefix or in suffix of the filename.

SORTING COMMANDS

58. SORT :

SYNTAX : $ sort filename
USES : Sort filter arranger input from standard input in alphabetical
OUTPUT : order
$Sort alamu

OPTIONS :

59.
SYNTAX : $ sort –rfilename
USES : sort command when used with this option will display input taken from keyboard in reverse alphabetical order .
OUTPUT : sort –r alamu
343
56
34
12

60.
SYNTAX : $ sort –nfilename
USES : This option will arrange the input according to numerical and display it .
OUTPUT : sort –n alamu
12
24
56
343

61. **SYNTAX** : $ sort –ffilename
USES : digits alphabets and other characters taken as input are converted to ASCII value . Sort arranges them according to their ASCII value.
OUTPUT : sort –f alamu
*
#
1
2
24
56
343

62. **SYNTAX** : $ sort –ufilename
USES : this option will remove duplicate lines from input and displays as output
OUTPUT : sort –u alamu
malathi
raji
rathi
vassanthi
vrs

63. CHMOD :

SYNTAX : $ chmod -w dir name
USES : This command is used to set the three permissions for all the three categories of users of the file . Only the owner of the file can use it.
OUTPUT : $ chmod –w cs 2 3
Ls –l cs 2 3
Total 8
Drwx rwx r-x 2 linux 4096 jun 24 14.46 cs 23

CP COMMAND

64. **SYNTAX** : $ cp –I filename1 filename2
USES : This command helps us to create duplicate copies of ordinary file.
OPTIONS : $ cp –i filename1filename2
OUTPUT : $cp rathi :rs
Cat rs
Name : bahirathi
Roll no: 01 cs 23
3.7.1982

65. **SYNTAX** : $cp –I filename1filename2
USES : The –I (interactive option , originally warns the users before over writing the destination file.
OUTPUT : $cp –I rs rathi

cp : overwrite ‘rathi’? n

66. **SYNTAX** : $cp –r filename1filename2
USES : it is now possible to copy an entire directory structures with –r (recursive) option
OUTPUT : $ cp –r rs rathi

67. THE PATH

SYNTAX : $echo PATH
USES : It specifies the current path of the operating system.
OUTPUT : /bin:/usv/bin:/usr/local/bin

68. CHANGE THE PASSWORD :

SYNTAX : $passwd
USE : password can be changed by using passwd command
OUTPUT : $ passwd
UX: passwd: INFO: change password for
local Old password: *****
New password: ******
Re-enter new password: ******
$

RESULT:

Exp-2	SHELL PROGRAMMING
Date - 6/5/22	

2.A)BIGGEST AMONG THREE NUMBERS

AIM : To write a shell script to find the biggest of three numbers.

ALGORITHM :

Step 1 : Start The Problem
Step 2 : Take Three Inputs From The User
Step 3 : In If-Else Condition, Check Which Is The
Greatest Step 4 : Also Check With The Third
Number
Step 5 : Find The
Result Step 6 : Print
The Result Step 7 :
Stop The Program

PROGRAM :

```
echo “Enter three Numbers :”
read a b c
if [ $a -gt $b ] && [ $a -gt $c ]
then
        echo “$a is Greatest”
elif [ $b -gt $c ] && [ $b -gt $a
] then
        echo “$b is
els
        Greatest” echo “$c
e fi
        is Greatest!”
```

OUTPUT :

```
Enter three number
: 23
67
40
B is big.
```

RESULT :

2.B) FACTORIAL OF A GIVEN NUMBER

AIM : To write a shell script to print the factorial of a given number.

ALGORITHM :

Step 1 : Get a number
Step 2 : Use do-while loop to compute the factorial by using the below formula Step 3 : fact(n) = n * n-1 * n-2 * .. 1
Step 4 : Display the result.

PROGRAM :

```
#!/bin/bash
echo "Enter a number : "
read num
fact=1
while [ $num -gt 1 ]
do
fact=$((fact * num))
num=$((num - 1)) done
echo Factorial=$fact
```

OUTPUT :

```
Enter a number
: 4
24
```

RESULT :

2.C) MULTIPLICATION TABLE OF A NUMBER

AIM : To write a shell script to print the multiplication table of given number up to a given range.

ALGORITHM :

Step 1 : Get the numbers to be displayed for multiplication and limit of the multiplication table.
Step 2 : Check using while
loop. Step 3 : Calculate the
result.
Step 4 : Increment the variable and repeat.
Step 5 : Display the value and terminate the program.

PROGRAM :

```
#!/bin/bash
echo "Enter the Number : "
read a
echo Multiplication Table of $a
for((i=1;i<=10;i++))
do
echo
$i*$a=$(($a*$i))
done
```

OUTPUT :

```
Enter the Number : 5

Multiplication Table of 5
1*5=5
2*5=10
3*5=15
4*5=20
5*5=25
6*5=30
7*5=35
8*5=40
9*5=45
10*5=50
```

RESULT :

2.D)SIMPLE ADD FUNCTION

AIM : To write a shell program to call the function with arguments, add the argument values after the function name.

ALGORITHM :

Step 1 : Initialize two variables.
Step 2 : Declare and implement the addition function.
Step 3 : Call the add function with two arguments.

PROGRAM :

```
function add()
{
sum=$(($1 + $2))
echo "Sum = $sum"
}
a=1
0
b=2
0
#call the add function and pass the
values add $a $b
```

OUTPUT :

Sum = 30

RESULT :

2.E) FIBONACCI SERIES

AIM : To write a shell program to print the Fibonacci series of a given number.

ALGORITHM :

Step 1: Start
Step 2: Declare variable x, y, z, n, i
Step 3: Initialize variable x=0, y=1 and
i=2 Step 4: Read n from user
Step 5: Print x and y
Step 6: Repeat until i<=n :
    z=x+y
    print z
    x=y,
    y=z
    i=i+1
Step 7: Stop

PROGRAM :

```
clear
echo "Program to Find Fibonacci Series"
echo "How many number of terms to be generated ?"
read n
x=
0
y=
1
i=2
echo "Fibonacci Series up to $n terms
:" echo "$x"
echo "$y"
while [ $i -lt $n ]
do
i=`expr $i + 1 `
z=`expr $x + $y
` echo "$z"
x=$y
y=$z
don
e
```

OUTPUT :

```
Program to Find Fibonacci Series
How many number of terms to be generated ? 7
Fibonacci Series up to 7 terms :
0 1 1 2 3 5 8
```

RESULT :

2.F) SUM OF N NATURAL NUMBERS

AIM : To write a shell program to find the sum of n natural numbers.

ALGORITHM :

Step 1 : Start
Step 2 : Display “Enter the number of
N” Step 3 : read n
Step 4 : sum ←0
Step 5 : Repeat Step 6
    i ←0
    While i ≤n
Step 6 : sum ←sum + i
Step 7 : Display “the sum is”
sum Step 8 : Stop

PROGRAM :

```
echo "Enter the number of N : "
read n
sum=0
for (( i=0; i<=n; i++
)) do
sum=$((sum + i))
done
echo -e "The sum of first N number is \t $sum"
```

OUTPUT :

```
Enter the number of N : 5
The sum of first N number is 15
```

RESULT :

Exp-3	SYSTEM CALLS
Date - 20/5/22	

3.A)PROGRAM USING SYSTEM CALL fork( )

AIM : To write a C program to implement fork() system call.

DESCRIPTION OF SYSTEM CALLS USED :

1. **fork() -** Used to create new processes. The new process consists of a copy of the address space of the original process. The value of process id for the child process is zero, whereas the value of process id for the parent is an integer value greater than zero.
   Syntax : fork()
2. **execlp() -** Used after the fork() system call by one of the two processes to replace the process‟memory space with a new program. It loads a binary file into memory destroying the memory image of the program containing the execlp system call and starts its execution. The child process overlays its address space with the UNIX command /bin/ls using the execlp system call.
   Syntax : execlp( )
3. **wait() -** The parent waits for the child process to complete using the wait system call. The wait system call returns the process identifier of a terminated child, so that the parent can tell which of its possibly many children has terminated.
   Syntax : wait(NULL)
4. **exit() -** A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit system call. At that point, the process may return data (output) to its parent process (via the wait system call). **Syntax:** exit(0)

PROGRAM :

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

```
    wait(NULL);
    exit(0);
  }
}
```

OUTPUT :

```
[cse6@localhost Pgm]$ cc
prog4a.c [cse6@localhost Pgm]$
./a.out Process id is 156234
```

RESULT :

3.B) PROGRAM USING SYSTEM CALL getpid( ) & getppid( )

AIM : To write a C program to implement getpid() and getppid() system calls.

DESCRIPTION OF SYSTEM CALLS USED :

1. **getpid() -** Each process is identified by its id value**.** This function is used to get the id value of a particular process.
2. **getppid() -** Used to get particular process parent's id value.
3. **perror() -** Indicate the process error.

PROGRAM :

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

OUTPUT :

```
Parent process is under execution
Process id of the parent process is
5194
Process id of the child process in parent is 5194
Process id of the parent of parent is 5193
```

RESULT :

3.C) PROGRAM USING SYSTEM CALL opendir( ) readdir( ) & closedir()

AIM : To write a C program to implement opendir(), readdir() and closedir() system calls.

SYSTEM CALLS USED :

1. **opendir() -** Open a directory.
2. **readdir() -** Read a directory.
3. **closedir() -** Close a directory

PROGRAM :

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

OUTPUT :

```
[cse6@localhost Pgm]$ cc proga.c
[cse6@localhost Pgm]$ ./a.out mkdir cse
[cse6@localhost Pgm]$ cd cse
[cse6@localhost Pgm]cse$ cat>file1
[cse6@localhost Pgm]cse$ cat>file2
[cse6@localhost Pgm]cse$ cd
[cse6@localhost Pgm]$ ./a.out cse
File1
File 2
```

RESULT :

3.D)PROGRAM USING SYSTEM CALL exec( )

AIM : To write a C program to implement exec() system call.

SYSTEM CALLS USED :

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

PROGRAM :

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

OUTPUT :

Wed 04 May 2022 03:36:34 AM UTC

RESULT :

3.E) PROGRAM USING SYSTEM CALL wait( ) & exit( )

AIM : To write a C program to implement wait() and exit() system calls.

SYSTEM CALLS USED :

1. **fork ( ) -** Used to create new process. The new process consists of a copy of the address space of the original process. The value of process id for the child process is zero, whereas the value of process id for the parentis an int value greater than zero. **Syntax:** fork ( )
2. **wait () -** The parent waits for the child process to complete using the wait system call. The wait system call returns the process identifier of a terminated child, so that the parent can tell which of its possibly many children has terminated.
   Syntax: wait (NULL)
3. **exit () -** A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit system call. At that point, the process may return data (output) to its parent process (via the wait system call). **Syntax:** exit(0)

PROGRAM :

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

OUTPUT :

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

RESULT :

3.F) PROGRAM USING SYSTEM CALL open( ) read( ) & close( )

AIM : To write a C program to implement open(), read() and close() system calls.

SYSTEM CALLS USED :

1. open()
2. read()
3. write()
4. close()
5. gets()
6. lseek()

PROGRAM :

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

OUTPUT :

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

RESULT :

Exp-4	NON PRE-EMPTIVE CPU SCHEDULING ALGORITHMS
Date - 27/5/22	

OBJECTIVE : To simulate the following non pre-emptive CPU scheduling algorithms to find turnaround time and waiting time

a) FCFS
b) SJF
c) Round Robin
d) Priority

ENVIRONMENT :

LINUX **LANGUAGE :**

C

DESCRIPTION : Assume all the processes arrive at the same time.

FCFS CPU SCHEDULING ALGORITHM

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

SJF CPU SCHEDULING ALGORITHM

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

ROUND ROBIN CPU SCHEDULING ALGORITHM

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

PRIORITY CPU SCHEDULING ALGORITHM

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

Calculate the waiting time and turnaround time of each of the processes accordingly.

PROGRAM :

(a) FCFS

```
#include
<stdio.h> int
main()
{ int bt[20], wt[20], tat[20], i, n;
  float wtavg, tatavg;
  printf("\nEnter the number of processes -- ");
  scanf("%d", &n);
  for(i=0;i<n;i++)
   { printf("\nEnter Burst Time for Process %d -- ", i);
     scanf("%d", &bt[i]);
   }
  wt[0]=wtavg=0;
  tat[0]=tatavg=bt[0];
  for(i=1;i<n;i++)
   { wt[i] = wt[i - 1] + bt[i - 1];
     tat[i] = tat[i - 1] + bt[i];
     wtavg = wtavg + wt[i];
     tatavg   =   tatavg   +
     tat[i];
   }
  printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
  for(i=0;i<n;i++)
  printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]);
  printf("\nAverage Waiting Time -- %f", wtavg / n);
  printf("\nAverage Turnaround Time -- %f", tatavg / n);
  return 0;
}
```

OUTPUT :

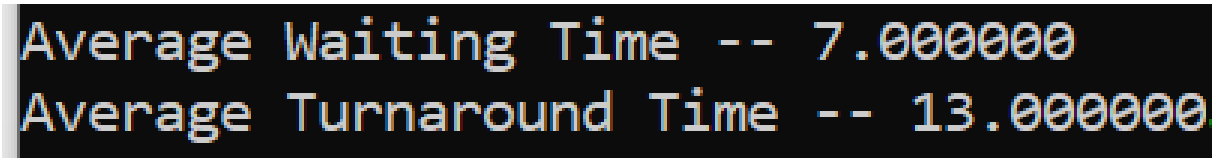
```
Enter the number of processes --  3

Enter Burst Time for Process 0 -- 24

Enter Burst Time for Process 1 -- 3

Enter Burst Time for Process 2 -- 3
        PROCESS         BURST TIME       WAITING TIME    TURNAROUND TIME

        P0              24               0               24
        P1              3                24              27
        P2              3                27              30
```

```
Average Waiting Time -- 17.000000
Average Turnaround Time -- 27.000000
```

(b) SJF

```
#include
<stdio.h> int
main()
{ int p[20],bt[20],wt[20],tat[20],i,k,n,temp;
  float wtavg, tatavg;
  printf("\nEnter the number of processes -- ");
  scanf("%d", &n);
  for(i=0;i<n;i++)
   { p[i]=i;
     printf("Enter Burst Time for Process %d -- ", i);
     scanf("%d", &bt[i]);
   }
  for(i=0;i<n;i++)
   for(k=i+1;k<n;k++)
   if(bt[i]>bt[k])
     { temp=bt[i];
       bt[i]=bt[k];
       bt[k]=temp;
     }
    wt[0]=wtavg=0;
    tat[0]=tatavg=bt[0];
    for(i=1;i<n;i++)
     { temp=p[i];
       p[i]=p[k];
       p[k]=temp;
       wt[i]=wt[i-1]+bt[i-1];
       tat[i]=tat[i-1]+bt[i];
       wtavg=wtavg+wt[i];
       tatavg=tatavg+tat[i]
       ;
     }
    printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
    for(i=0;i<n;i++)
    printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);
    printf("\nAverage Waiting Time -- %f", wtavg / n);
    printf("\nAverage Turnaround Time -- %f", tatavg / n);
    return 0;
}
```

OUTPUT:



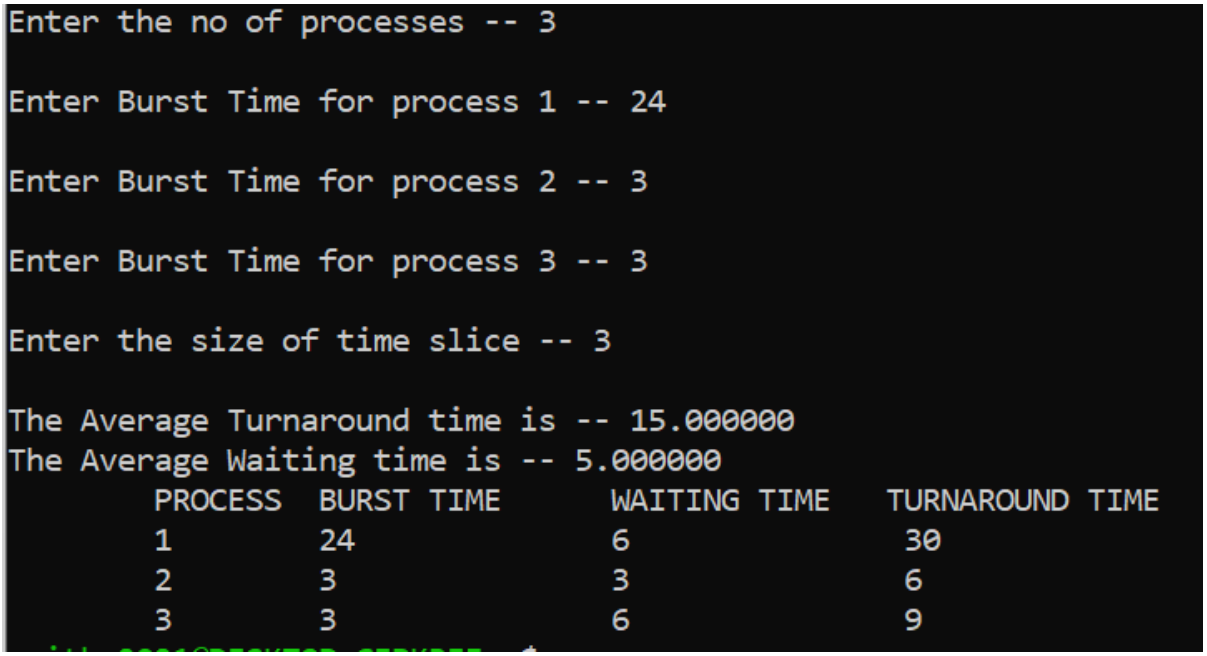
```
Enter the number of processes --  4
Enter Burst Time for Process 0 -- 6
Enter Burst Time for Process 1 -- 8
Enter Burst Time for Process 2 -- 7
Enter Burst Time for Process 3 -- 3

        PROCESS        BURST TIME       WAITING TIME    TURNAROUND TIME

        P0              3               0               3
        P0              6               3               9
        P1              7               9               16
        P2              8               16              24
```

(c)ROUND ROBIN

```
#include
<stdio.h> int
main()
{ int i, j, n, bu[10], wa[10], tat[10], t, ct[10], max;
  float awt = 0, att = 0, temp = 0;
  printf("Enter the no of processes -- ");
  scanf("%d", &n);
  for(i=0;i<n;i++)
   { printf("\nEnter Burst Time for process %d -- ", i +
     1); scanf("%d", &bu[i]);
     ct[i]=bu[i];
   }
  printf("\nEnter the size of time slice -- ");
  scanf("%d",&t);
  max=bu[0];
  for(i=1;i<n;i++)
  if(max<bu[i])
  max=bu[i];
  for(j=0;j<(max/t)+1;j+
  +) for(i=0;i<n;i++)
  if(bu[i]!=0)
     if(bu[i]<=t)
      { tat[i]=temp+bu[i];
        temp=temp+bu[i]
        ; bu[i]=0;
      }
     else
      { bu[i]=bu[i]-t;
        temp=temp+t;
      }
  for(i=0;i<n;i++)
   { wa[i]=tat[i]-ct[i];
     att+=tat[i];
     awt+=wa[i];
```

```
  }
 printf("\nThe Average Turnaround time is -- %f", att / n);
 printf("\nThe Average Waiting time is -- %f ", awt / n);
 printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");
 for (i = 0; i < n; i++)
  printf("\t%d \t %d \t\t %d \t\t %d \n", i + 1, ct[i], wa[i], tat[i]);
 return 0;
}
```

OUTPUT :



(d) PRIORITY

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

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

OUTPUT :

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

PROCESS         PRIORITY        BURST TIME      WAITING TIME    TURNAROUND TIME
1                1               1               0               1
4                2               5               1               6
0                3               10              6               16
2                4               2               16              18
3                5               1               18              19
```

RESULT :

Exp-5	IMPLEMENTATION OF PROCESS SYNCHRONIZATION USING SEMAPHORE
Date - 3/6/22	

OBJECTIVE :

(a) To simulate producer-consumer problem using semaphores.
(b) To simulate the concept of Dining-Philosophers problem.

ENVIRONMENT :

LINUX **LANGUAGE :** C

DESCRIPTION :

(a) 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.

(b) 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 centre 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 neighbours). 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 neighbour. 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.

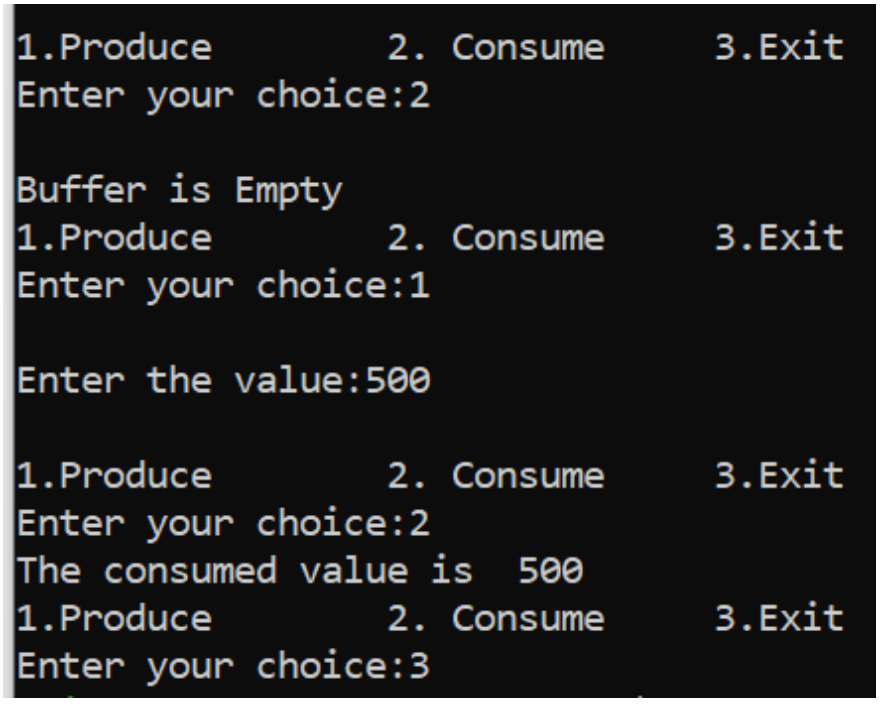
PROGRAM :

(a) PRODUCER CONSUMER PROBLEM

```
#include
<stdio.h> void
main()
{ int buffer[10],bufsize,in,out,produce,consume,choice =
  0; in=0;
```

```
 out=0;
 bufsize=10;
 while(choice!=3
 )
  { printf("\n1.Produce \t 2. Consume \t3.Exit");
    printf("\nEnter your choice:");
    scanf("%d",
    &choice);
    switch(choice)
     { case 1:
            if((in+1)%bufsize==out)
            printf("Buffer is Full");
            else
             { printf("\nEnter the value:");
               scanf("%d", &produce);
               buffer[in]=produce;
               in=(in+1)%bufsize;
             }
            break;
       case 2: if(in==out)
            printf("\nBuffer is Empty");
            else
             { consume = buffer[out];
               printf("The consumed value is % d",
               consume); out = (out + 1) % bufsize;
             }
            break;
     }
  }
}
```

OUTPUT:



(b) DINING PHILOSOPHER PROBLEM

```
#include
<stdio.h>
#include
<stdlib.h>
int
tph,philname[20],status[20],howhung,hu[20],cho;
void 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]]);
   }
}
void 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]]);
           }
        }
      }
   }
}
void main()
{ int i;
  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("\n1.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);
   }
}
```

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 :

Exp-6	DEADLOCK AVOIDANCE
Date - 10/6/22	

OBJECTIVE : To simulate Bankers algorithm for the purpose of Deadlock avoidance

ENVIRONMENT :

LINUX **LANGUAGE :** C

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.

PROGRAM :

```
#include
<stdio.h> struct
file
{ int all[10],max[10],need[10],flag;
};
void main()
{ struct file f[10];
  int fl,i,j,k,p,b,n,r,g,cnt = 0,id,newr,avail[10], seq[10];
  printf("Enter number of processes -- ");
  scanf("%d", &n);
  printf("Enter number of resources -- ");
  scanf("%d", &r);
  for(i=0;i<n;i++)
   { printf("Enter details for P%d", i);
     printf("\nEnter allocation\t -- \t");
     for(j=0;j<r;j++)
     scanf("%d", &f[i].all[j]);
     printf("Enter Max\t\t -- \t");
     for(j=0;j<r;j++)
     scanf("%d", &f[i].max[j]);
```

```
  f[i].flag = 0;
 }
printf("\nEnter Available Resources\t --
\t"); for(i=0;i<r;i++)
 scanf("%d", &avail[i]);
 printf("\nEnter New Request Details -- ");
 printf("\nEnter pid \t -- \t");
 scanf("%d", &id);
 printf("Enter Request for Resources \t -- \t");
for(i=0;i<r;i++)
 { scanf("%d",&newr);
  f[id].all[i]+=newr;
  avail[i]=avail[i]-
  newr;
 }
for(i=0;i<n;i++)
 { for(j=0;j<r;j++)
   { }
 }
cnt = 0;
fl = 0;
f[i].need[j]=f[i].max[j]-f[i].all[j];
if(f[i].need[j]<0)
 f[i].need[j]=0;
while(cnt!=n)
 { g=0;
  for(j=0;j<n;j++)
   { if(f[j].flag==0)
     { b=0;
       for(p=0;p<r;p++)
        {
          if(avail[p]>=f[j].need[p
          ]) b=b+1;
         else
          b=b-
          1;
        }
       if(b==r)
        { printf("\nP%d is visited", j);
          seq[fl++]=j;
          f[j].flag=1;
          for(k=0;k<r;k++)
          avail[k]=avail[k]+f[j].all[k
          ]; cnt=cnt+1;
           printf("(");
          for(k=0;k<r;k++)
          printf("%3d", avail[k]);
           printf(")");
```

```
            g = 1;
          }
        }
      }
     if(g==0)
      { printf("\n REQUEST NOT GRANTED -- DEADLOCK
        OCCURRED"); printf("\n SYSTEM IS IN UNSAFE STATE");
        goto y;
      }
   }
  printf("\nSYSTEM IS IN SAFE STATE");
  printf("\nThe Safe Sequence is -- (");
  for(i=0;i<fl;i++)
   printf("P%d ", seq[i]);
   printf(")");
  y:
  printf("\nProcess\t\tAllocation\t\tMax\t\t\tNeed\n");
  for(i=0;i<n;i++)
   { printf("P%d\t", i);
     for(j=0;j<r;j++)
     printf("%6d",f[i].all[j]);
     for (j=0;j<r;j++)
     printf("%6d",f[i].max[j]);
     for (j=0;j<r;j++)
     printf("%6d",f[i].need[j])
     ; printf("\n");
   }
}
```

OUTPUT:

```
Enter number of processes -- 5   3
Enter number of resources -- Enter details for P0
Enter allocation          --     0 1 0
Enter Max                 --     7 5 3
Enter details for P1
Enter allocation          --     2 0 0
Enter Max                 --     3 2 2
Enter details for P2
Enter allocation          --     3 0 2
Enter Max                 --     9 0 2
Enter details for P3
Enter allocation          --     2 1 1
Enter Max                 --     2 2 2
Enter details for P4
Enter allocation          --     0 0 2
Enter Max                 --     4 3 3

Enter Available Resources         --     3 3 2

Enter New Request Details --
Enter pid         --     1
Enter Request for Resources       --     1 0 2

 REQUEST NOT GRANTED -- DEADLOCK OCCURRED
 SYSTEM IS IN UNSAFE STATE
Process         Allocation              Max                     Need
P0          0     1     0     7     5     3     0     01736415839
P1          3     0     2     3     2     2 325991148406248 32599
P2          3     0     2     9     0     21148292044 325991148401465
P3          2     1     1     2     2     2 32767     0     0
P4          0     0     2     4     3     3    42     0     0
```

RESULT :

Exp-7	MEMORY MANAGEMENT TECHNIQUES
Date - 17/6/22	

OBJECTIVE : To simulate the (a) MFT and (b) MVT memory management techniques

ENVIRONMENT :

LINUX **LANGUAGE :** C

DESCRIPTION :

MFT (Multiprogramming with a Fixed number of Tasks) is one of the old memory management techniques in which the memory is partitioned into fixed size partitions and each job is assigned to a partition. The memory assigned to a partition does not change. MVT (Multiprogramming with a Variable number of Tasks) is the memory management technique in which each job gets just the amount of memory it needs. That is, the partitioning of memory is dynamic and changes as jobs enter and leave the system. MVT is a more ``efficient'' user of resources. MFT suffers with the problem of internal fragmentation and MVT suffers with external fragmentation.

PROGRAM :

(a) MFT:

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

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

OUTPUT :

```
Enter the total memory available (in Bytes) -- 1000
Enter the block size (in Bytes) -- 300

Enter the number of processes -- 5
Enter memory required for process 1 (in Bytes)-- 275
Enter memory required for process 2 (in Bytes)-- 400
Enter memory required for process 3 (in Bytes)-- 290
Enter memory required for process 4 (in Bytes)-- 293
Enter memory required for process 5 (in Bytes)-- 100

No. of Blocks available in memory -- 3

PROCESS MEMORY REQUIRED  ALLOCATED      INTERNAL FRAGMENTATION
  1             275             YES     25
  2             400             NO              ---
  3             290             YES     10
  4             293             YES     7
Memory is Full, Remaining Processes cannot be accomodated

Total Internal Fragmentation is 42
```

RESULT :

(b)MVT:

```
#include
<stdio.h> void
main()
{ int ms, mp[10], i, temp, n = 0;
  char ch = 'y';
  printf("\nEnter the total memory available (in Bytes)-- ");
  scanf("%d", &ms);
  temp = ms;
  for (i = 0; ch == 'y'; i++, n++)
```

```
  { printf("\nEnter memory required for process %d (in Bytes) -- ", i + 1);
    scanf("%d", &mp[i]);
    if (mp[i] <= temp)
     { printf("\nMemory is allocated for Process %d ", i +
       1); temp = temp - mp[i];
     }
    else
     { printf("\nMemory is Full");
       break;
     }
    printf("\nDo you want to continue(y/n) -- ");
    scanf(" %c", &ch);
  }
 printf("\n\nTotal Memory Available -- %d", ms);
 printf("\n\n\tPROCESS\t\t MEMORY
 ALLOCATED "); for (i = 0; i < n; i++)
  printf("\n \t%d\t\t%d", i + 1, mp[i]);
  printf("\n\nTotal Memory Allocated is %d", ms -
  temp); printf("\nTotal External Fragmentation is
  %d", temp);
}
```

OUTPUT:

```
Enter the total memory available (in Bytes)-- 1000

Enter memory required for process 1 (in Bytes) -- 400

Memory is allocated for Process 1
Do you want to continue(y/n) -- y

Enter memory required for process 2 (in Bytes) -- 275

Memory is allocated for Process 2
Do you want to continue(y/n) -- y

Enter memory required for process 3 (in Bytes) -- 550

Memory is Full

Total Memory Available -- 1000

        PROCESS          MEMORY ALLOCATED
        1               400
        2               275
```

RESULT :

Exp-8 Date - 24/6/22	MEMORY ALLOCATION TECHNIQUES

OBJECTIVE : To simulate the following contiguous memory allocation techniques

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

fit **ENVIRONMENT :** LINUX

LANGUAGE : C

DESCRIPTION :

One of the simplest methods for memory allocation is to divide memory into several fixed- sized partitions. Each partition may contain exactly one process. In this multiple-partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process. The operating system keeps a table indicating which parts of memory are available and which are occupied. Finally, when a process arrives and needs memory, a memory section large enough for this process is provided. When it is time to load or swap a process into main memory, and if there is more than one free block of memory of sufficient size, then the operating system must decide which free block to allocate. Best fit strategy chooses the block that is closest in size to the request. First-fit chooses the first available block that is large enough. Worst-fit chooses the largest available block

PROGRAM

(a) WORST-FIT

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

```
  for (i = 1; i <= nf; i++)
   { for (j = 1; j <= nb; j++)
     { if (bf[j] != 1)
       { temp = b[j] - f[i];
         if (temp >= 0)
          if (highest < temp)
           { ff[i] = j;
             highest = temp;
           }
       }
     }
    frag[i] = highest;
    bf[ff[i]] = 1;
    highest = 0;
   }
  printf("\nFile_no:\tFile_size
  :\tBlock_no:\tBlock_size:\tFragement"); for (i = 1; i <= nf; i++)
  printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}
```

OUTPUT :

```
        Memory Management Scheme - Worst Fit
Enter the number of blocks:3
Enter the number of files:2

Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:1
File 2:4

File_no:        File_size :     Block_no:       Block_size:     Fragement
1               1               3               7               6
```

(b) BEST –FIT

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

```
 printf("\nEnter the size of the blocks:-\n");
 for (i = 1; i <= nb; i++)
  { printf("Block %d:", i);
    scanf("%d", &b[i]);
  }
 printf("Enter the size of the files :-\n");
 for (i = 1; i <= nf; i++)
  { printf("File %d:", i);
    scanf("%d", &f[i]);
  }
 for (i = 1; i <= nf; i++)
  { for (j = 1; j <= nb; j++)
     { if (bf[j] != 1)
       { temp = b[j] - f[i];
         if (temp >= 0)
          if (lowest > temp)
           { ff[i] = j;
             lowest = temp;
           }
       }
     }
    frag[i] = lowest;
    bf[ff[i]] = 1;
    lowest = 10000;
  }
 printf("\nFile No\tFile Size \tBlock No\tBlock
 Size\tFragment"); for (i = 1; i <= nf && ff[i] != 0; i++)
  printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}
```

OUTPUT :

```
        Memory Management Scheme - Best Fit
Enter the number of blocks:3
Enter the number of files:2

Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:1
File 2:4

File No File Size       Block No        Block Size      Fragment
1               1               2               2               1
```

(c) FIRST FIT

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

OUTPUT :

```
       Memory Management Scheme - First Fit
Enter the number of blocks:3
Enter the number of files:2

Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:1
File 2:4

File_no:        File_size :     Block_no:       Block_size:     Fragement
1               1               1               5               4
```

RESULT :

Exp-9	VIRTUAL MEMORY MANAGEMENT TECHNIQUES
Date - 1/7/22	

OBJECTIVE : To simulate paging technique of memory management.

ENVIRONMENT :

LINUX **LANGUAGE :** C

DESCRIPTION :

In computer operating systems, paging is one of the memory management schemes by which a computer stores and retrieves data from the secondary storage for use in main memory. In the paging memory-management scheme, the operating system retrieves data from secondary storage in same-size blocks called pages. Paging is a memory-management scheme that permits the physical address space a process to be non-contiguous. The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocks of the same size called pages. When a process is to be executed, its pages are loaded into any available memory frames from their source.

PROGRAM :

```
#include
<stdio.h> void
main()
{
int ms, ps, nop, np, rempages, i, j, x, y, pa,
offset; int s[10], fno[10][20];
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);
}
}
```

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 :

Exp-10	FILE ALLOCATION TECHNIQUES
Date - 8/7/22	

OBJECTIVE : Write a C program to simulate the following file allocation strategies. a) Sequential b) Linked c)

Indexed **ENVIRONMENT :**

LINUX **LANGUAGE :** C

DESCRIPTION :

A file is a collection of data, usually stored on disk. As a logical entity, a file enables to divide data into meaningful groups. As a physical entity, a file should be considered in terms of its organization. The term "file organization" refers to the way in which data is stored in a file and, consequently, the method(s) by which it can be accessed.

SEQUENTIAL FILE ALLOCATION : In this file organization, the records of the file are stored one after another both physically and logically. That is, record with sequence number 16 is located just after the 15th record. A record of a sequential file can only be accessed by reading all the previous records.

LINKED FILE ALLOCATION : With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block.

INDEXED FILE ALLOCATION : Indexed file allocation strategy brings all the pointers together into one location: an index block. Each file has its own index block, which is an array of disk- block addresses. The ith entry in the index block points to the ith block of the file. The directory contains the address of the index block. To find and read the ith block, the pointer in the ith index-block entry is used.

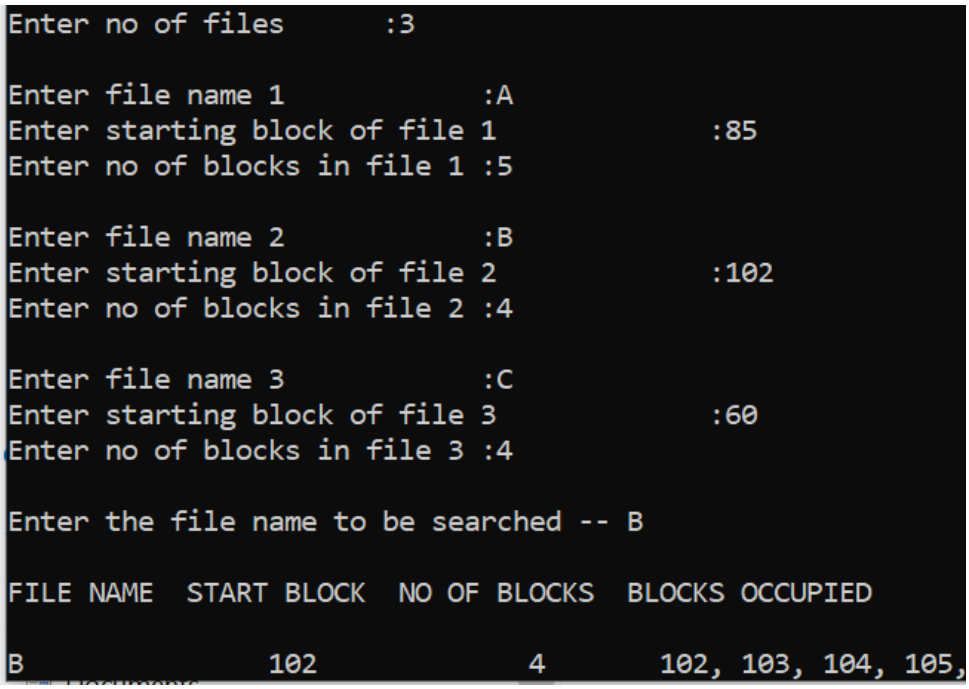
PROGRAM :

(a) SEQUENTIAL FILE ALLOCATION

```
#include
<stdio.h>
#include
<string.h>   struct
fileTable
{
char
name[20]; int
sb, nob;
} ft[30];
void main()
{
int i, j, n;
char
```

```
s[20];
printf("Enter no of files :");
scanf("%d", &n);
for (i = 0; i < n; i++)
{
printf("\nEnter file name %d :", i + 1);
scanf("%s", ft[i].name);
printf("Enter starting block of file %d :", i +
1); scanf("%d", &ft[i].sb);
printf("Enter no of blocks in file %d :", i + 1);
scanf("%d", &ft[i].nob);
}
printf("\nEnter the file name to be searched -- ");
scanf("%s", s);
for (i = 0; i < n; i++)
if (strcmp(s, ft[i].name) ==
0) if (i == n)
printf("\nFile Not Found");
else
{
printf("\nFILE NAME START BLOCK NO OF BLOCKS BLOCKS OCCUPIED\n");
printf("\n%s\t\t%d\t\t%d\t", ft[i].name, ft[i].sb, ft[i].nob);
for (j = 0; j < ft[i].nob; j++)
printf("%d, ", ft[i].sb + j);
}
}
```

OUTPUT:

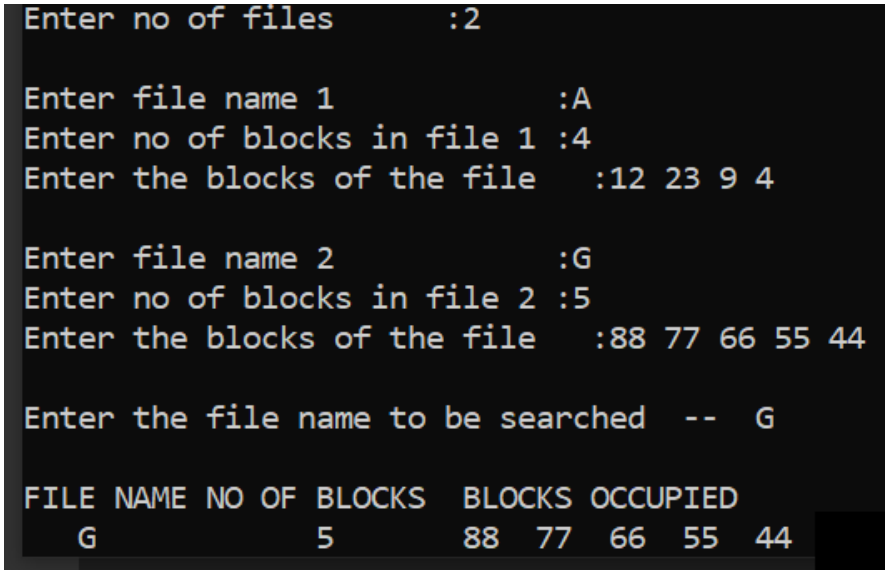


(b) LINKED FILE ALLOCATION

```
#include
<stdio.h>
#include
<string.h>
#include
<stdlib.h>    struct
fileTable
{
char
name[20]; int
nob;
struct block *sb;
} ft[30];
struct block
{
int bno;
struct block *next;
};
void main()
{
int i, j, n;
char
s[20];
struct block *temp;
printf("Enter no of files :");
scanf("%d", &n);
for (i = 0; i < n; i++)
{
printf("\nEnter file name %d :", i + 1);
scanf("%s", ft[i].name);
printf("Enter no of blocks in file %d :", i + 1);
scanf("%d", &ft[i].nob);
ft[i].sb = (struct block *)malloc(sizeof(struct
block)); temp = ft[i].sb;
printf("Enter the blocks of the file :");
scanf("%d", &temp->bno);
temp->next = NULL;
for (j = 1; j < ft[i].nob; j++)
{
temp->next = (struct block *)malloc(sizeof(struct
block)); temp = temp->next;
scanf("%d", &temp->bno);
}
temp->next = NULL;
}
printf("\nEnter the file name to be searched -- ");
```

```
scanf("%s", s);
for (i = 0; i < n; i++)
if (strcmp(s, ft[i].name) == 0)
break;
if (i == n)
printf("\nFile Not Found");
else
{
printf("\nFILE NAME NO OF BLOCKS BLOCKS OCCUPIED");
printf("\n %s\t\t%d\t", ft[i].name, ft[i].nob);
temp = ft[i].sb;
for (j = 0; j < ft[i].nob; j++)
{
printf("%d ", temp->bno);
temp = temp->next;
}
}
}
```

OUTPUT:



(c) INDEXED FILE ALLOCATION

```
#include
<stdio.h>
#include
<string.h>   struct
fileTable
{  char  name[20];
int              nob,
blocks[30];
} ft[30];
void main()
{ int i, j, n;
```

```
char s[20];
printf("Enter no of files :");
scanf("%d", &n);
for (i = 0; i < n;
i++)
{ printf("\nEnter file name %d :", i + 1);
scanf("%s", ft[i].name);
printf("Enter no of blocks in file %d :", i + 1);
scanf("%d", &ft[i].nob);
printf("Enter the blocks of the file
:"); for (j = 0; j < ft[i].nob; j++)
scanf("%d", &ft[i].blocks[j]);
}
printf("\nEnter the file name to be searched -- ");
scanf("%s", s);
for (i = 0; i < n; i++)
if (strcmp(s, ft[i].name) == 0)
break;
if (i == n)
printf("\nFile Not Found");
else
{ printf("\nFILE NAME NO OF BLOCKS BLOCKS OCCUPIED");
printf("\n %s\t\t%d\t", ft[i].name, ft[i].nob);
for (j = 0; j < ft[i].nob; j++)
printf("%d, ", ft[i].blocks[j]);
}
}
```

OUTPUT :

```
Enter no of files      :2

Enter file name 1            :A
Enter no of blocks in file 1 :4
Enter the blocks of the file   :12 23
9 4

Enter file name 2            :H
Enter no of blocks in file 2 :5
Enter the blocks of the file   :88 77 88 66 55

Enter the file name to be searched -- H

FILE NAME NO OF BLOCKS  BLOCKS OCCUPIED
   H            5       88, 77, 88, 66, 55,
```

RESULT :

Exp-11 Date - 8/7/22	DISK SCHEDULING ALGORITHM

OBJECTIVE : Write a C program to simulate disk scheduling algorithms

a) FCFS b) SCAN

ENVIRONMENT :

LINUX **LANGUAGE :**

C **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.

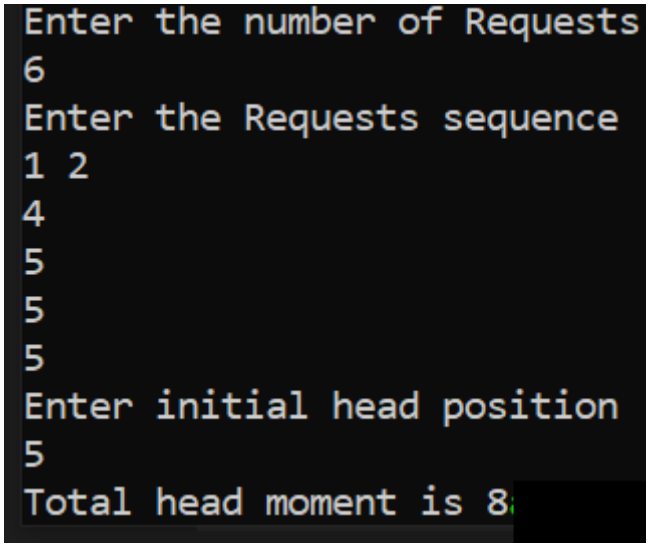
PROGRAM :

a) FCFS

```
#include
<stdio.h>
#include
<stdlib.h>        int
main()
{
int RQ[100], i, n, TotalHeadMoment = 0, initial;
printf("Enter the number of Requests\n");
scanf("%d", &n);
printf("Enter the Requests
sequence\n"); for (i = 0; i < n; i++)
scanf("%d", &RQ[i]);
printf("Enter initial head position\n");
scanf("%d", &initial);
// logic for FCFS disk
scheduling for (i = 0; i < n; i++)
{
TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
initial = RQ[i];
}
printf("Total head moment is %d",
TotalHeadMoment); return 0;
```

}

OUTPUT :



b) SCAN

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

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

```
1-0);
initial =size-1;
for(i=n-1;i>=index;i--)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial); initial=RQ[i];
}
}
printf("Total head movement is
%d",TotalHeadMoment); return 0;
}
```

OUTPUT :

```
Enter the number of Requests
8
Enter the Requests sequence
22
33 44 55 66 77 88 99
Enter initial head position
50
Enter total disk size
200
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 392
```

RESULT :