SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY

COIMBATORE – 641008

(AN AUTONOMOUS INSTITUTION, AFFILIATED TO ANNA UNIVERSITY, CHENNAI)

DEPARTMENT OF COMPUTER SCIENCE AND DESIGN

22CY202 – OPERATING SYSTEMS LABORATORY

RECORD WORK

Submitted by

Name ` :

Register No. :

Degree & Branch : B.E Computer Science and Design

Class : II B.E CSD

SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY

COIMBATORE - 641008

(An AUTONOMOUS INSTITUTION, AFFILIATED TO ANNA UNIVERSITY, CHENNAI)

DEPARTMENT OF COMPUTER SCIENCE AND DESIGN

22CY202 – OPERATING SYSTEMS LABORATORY

Continuous Assessment Record

Submitted by

Name : **Register No. :**

Degree & Branch: B.E – CSD **Year & Semester : II / III**

BONAFIDE CERTIFICATE

This is to certify that this record is the bonafide record of work done by Mr./Ms. during the academic year 2024 – 2025.

Staff In-charge **HOD / CSD**

Submitted for the practical examination held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

EXP. NO	DATE	EXPERIMENT NAME	MARKS	SIGNATURE
1.		Study of Basic Linux Commands, proc file system of linux, disk I/O, buffer caches, disk monitoring tool.		
2.		Implementation of Shell Programming.		
3.		Implementation of Unix System Calls.		
4.		Implementation of Non Preemptive and Preemptive CPU Scheduling Algorithms.		
5.		Implementation of Dining Philosophers Problem to Demonstrate Process Synchronization.		
6.		Implementation of Banker's Algorithm for deadlock avoidance.		
7.		Implementation of Memory Allocation and Memory Management Techniques.		
8.		Implementation of Page Replacement Techniques.		
9.		Implementation of File organization Techniques.		
10.		Implementation of Disk Scheduling Algorithms.		
		AVERAGE		

Faculty In-Charge

SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY
An Autonomous Institution | Approved by AICTE | Affiliated to Anna University | Accredited by NAAC with A++ Grade
Kuniamuthur, Coimbatore – 641008
Phone : (0422)-2628001 (7 Lines) | Email : info@skcet.ac.in | Website : www.skcet.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND DESIGN

22CY202 – OPERATING SYSTEMS LABORATORY

Reg.No:
Name:

Components	Experiment Nos.									
	1	2	3	4	5	6	7	8	9	10
Aim & Algorithm (20)										
Coding (30)										
Compilation & Debugging (30)										
Execution & Result (20)										
TOTAL										
AVERAGE										

SIGNATURE OF THE FACULTY

SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY
An Autonomous Institution | Approved by AICTE | Affiliated to Anna University | Accredited by NAAC with A++ Grade
Kuniamuthur, Coimbatore – 641008
Phone : (0422)-2628001 (7 Lines) | Email : info@skcet.ac.in | Website : www.skcet.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND DESIGN

22CY202 – OPERATING SYSTEMS LABORATORY

Reg.No: **Name:**

Components	Experiment Nos.									
	1	2	3	4	5	6	7	8	9	10
Observation (80)										
Record (10)										
Viva (10)										
TOTAL										
AVERAGE										

SIGNATURE OF THE FACULTY

SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY
An Autonomous Institution | Approved by AICTE | Affiliated to Anna University | Accredited by NAAC with A++ Grade
Kuniamuthur, Coimbatore – 641008
Phone : (0422)-2628001 (7 Lines) | Email : info@skcet.ac.in | Website : www.skcet.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND DESIGN

22CY202–OPERATING SYSTEMS LABORATORY

ODD SEMESTER 2024-2025

Name of Faculty Members:
Ms. M.Sanmuga Priya

METHOD OF CONTINUOUS EVALUATION

EVALUATION RUBRICS - OBSERVATION

Criteria	Range of Marks			
	Excellent	**Good**	**Average**	**Below Average**
Aim & Algorithm (20)	18-20	14-17	10-13	0-9
Coding (30)	27-30	21-26	15-20	0-14
Compilation and Debugging (30)	27-30	21-26	15-20	0-14
Execution and Result (20)	18-20	14-17	10-13	0-9

RUBRICS - RECORD

Criteria	Range of Marks			
	Excellent	**Good**	**Average**	**Below Average**
Observation (80)	71-80	61-70	41-60	0-40
Record (10)	9-10	7-8	5-6	0-4
Viva (10)	9-10	7-8	5-6	0-4

Ex.1 Study of Basic Linux Commands, proc file system of linux, disk I/O, buffer caches, disk monitoring tool.

Date:

AIM: To study the basic commands in Linux.

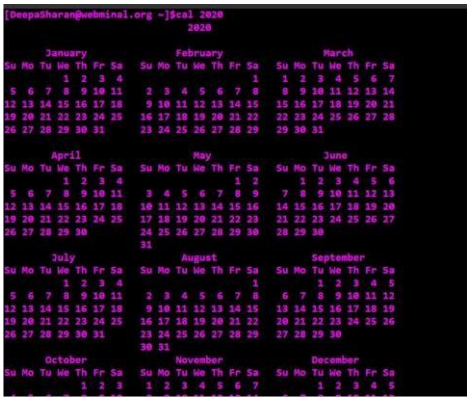
COMMANDS:

A. Linux Basic Commands

1. TASK : To display the current working directory.
   COMMAND : pwd
   SYNTAX : pwd
   EXPLANATION : This command displays the current working directory showing the path.
   OUTPUT :

```
[DeepaSharan@webminal.org ~]$pwd
/home/DeepaSharan
[DeepaSharan@webminal.org ~]$
```

2. TASK : To display the calendar of the current month.
   COMMAND : cal
   SYNTAX : cal 2020
   cal 10 2020
   EXPLANATION : This displays calendar of the current month on screen.
   OUTPUT :



3. TASK : To display user-defined message.
   COMMAND : echo.
   SYNTAX : echo “message”.
   EXPLANATION : This command displays the message after echo command on the screen
   OUTPUT :

```
[DeepaSharan@webminal.org ~]$echo "basics of linux"
basics of linux
[DeepaSharan@webminal.org ~]$
```

4. TASK : To display the current date, time, month and year.

COMMAND : date

SYNTAX : $date +%d /display date
$date +%m /display month
$date +%h /display month in words
$date +%y /display year
$date +%R /display the time with hour and mins
$date +%T /display time with hour,mins and sec

EXPLANATION : It is used to display the current date, time, month and year.

OUTPUT :

5. TASK : To display the user detail.

COMMAND : whoami

SYNTAX : whoami.

EXPLANATION : This command displays current user of the system on the

OUTPUT :

```
[DeepaSharan@webminal.org ~]$whoami DeepaSharan
```

6. TASK : To prints user and groups (UID and GID) of the current user.

COMMAND : id

SYNTAX : id

EXPLANATION : This command prints user and groups (UID and GID) of the current user.

OUTPUT :

```
[DeepaSharan@webminal.org ~]$id
uid=186146(DeepaSharan) gid=186205(DeepaSharan)
groups=186205(DeepaSharan) c
ontext=guest_u:guest_r:guest_t:s0
```

7. TASK : To clear the screen.

COMMAND : clear
SYNTAX : clear
EXPLANATION : This command clears the screen.
OUTPUT : After typing clear in the command line.

```
[DeepaSharan@webminal.org ~]$clear
```

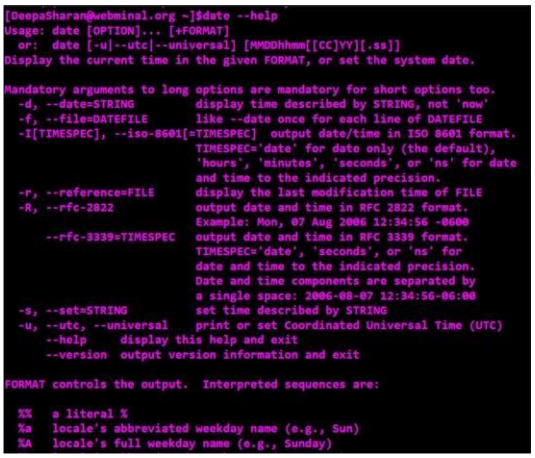
8. TASK : To gives a one line description about the command.

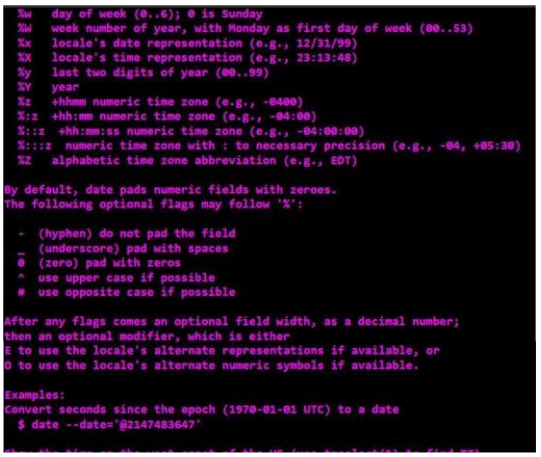
COMMAND : whatis
SYNTAX : whatis date
EXPLANATION : This command gives a one line description about the command. It can be used as a quick reference for any command.
OUTPUT :

```
[DeepaSharan@webminal.org ~]$whatis date
date (1)              - print or set the system date and time
```

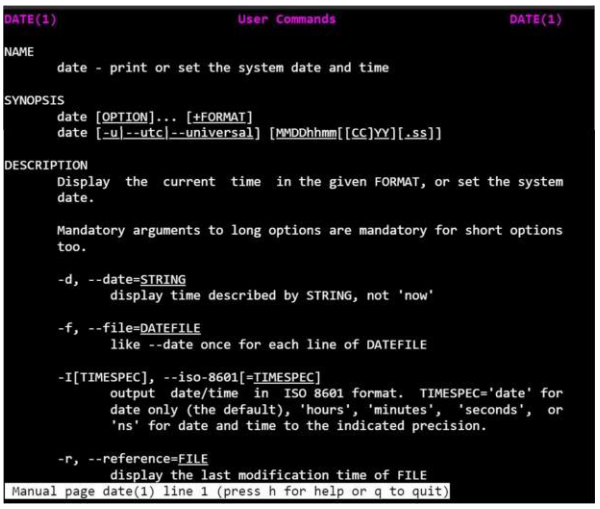
9. TASK : To display the summary of any command.

COMMAND : help
SYNTAX : date --help
EXPLANATION : With almost every command, „--help" option shows usage summary for that command.
OUTPUT :

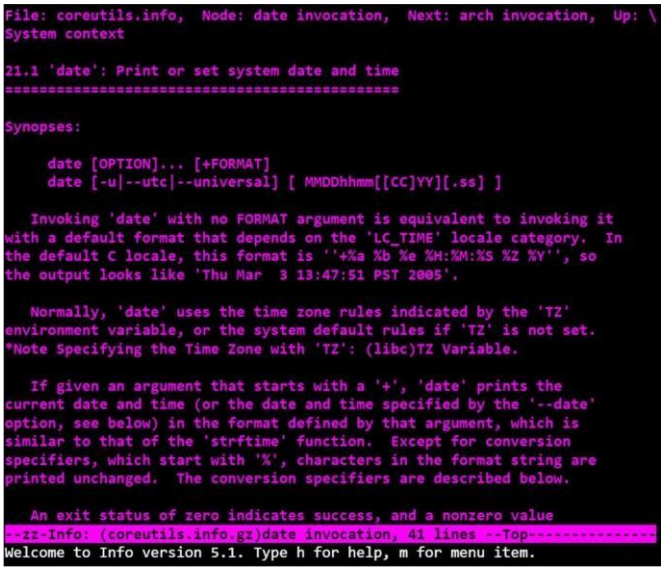




10. TASK : To display an interface to the on-line reference manuals
    COMMAND : man
    SYNTAX : man date
    EXPLANATION : This command displays the information about date.
    OUTPUT :



11. TASK : To display readable online documentation
    COMMAND : info
    SYNTAX : info date
    EXPLANATION : This command is to display readable online documentation
    OUTPUT :



B. Linux File system commands

12. TASK : To create a directory

COMMAND : mkdir
SYNTAX : mkdir dir
EXPLANATION : Creates a directory dir
OUTPUT :

```
[DeepaSharan@webminal.org ~]$mkdir dir
[DeepaSharan@webminal.org ~]$
```

13. TASK : To create an empty file.

COMMAND : touch
SYNTAX : touch file1
EXPLANATION : Creates an empty file file1
OUTPUT :

```
[DeepaSharan@webminal.org ~]$touch file1
[DeepaSharan@webminal.org ~]$
```

14. TASK : To create a file, display the contents of a file and concatenating the files.

COMMAND : cat
SYNTAX : cat > filename /for create
cat filename /for display
cat file1 file2>file3 /for concatenate
EXPLANATION : This command used to create a file, display the contents of a file and concatenating the files.
OUTPUT :

15. TASK : To create a file.
COMMAND vi
SYNTAX : vi filename
EXPLANATION : This command creates a file and the contents can be typed.
OUTPUT :

```
[DeepaSharan@webminal.org ~]$vi samplefile
[DeepaSharan@webminal.org ~]$
[DeepaSharan@webminal.org ~]$vi samplefile
```

```
"samplefile" [New File]                                    0,0-1         All
```

```
-- INSERT --                                               0,1           All
```

```
Hello world!
```

16. TASK : To display the files and folders present in the login.
COMMAND : ls
SYNTAX : ls
EXPLANATION : This command displays the files and folders present in the login.
OUTPUT

```
[DeepaSharan@webminal.org ~]$ls
deepaFile  deepaSharan.txt  deepa.txt  dir  file1  sharanya.txt
```

17. TASK : To delete a directory.
COMMAND : rmdir
SYNTAX : rmdir directory name
EXPLANATION : This command is used to delete the specified directory.
OUTPUT :

```
[DeepaSharan@webminal.org ~]$rmdir dir
[DeepaSharan@webminal.org ~]$
```

18. TASK : To copy a file.
COMMAND : copy
SYNTAX : cp sourcefile destinationfile.
EXPLANATION : This command produces a copy of the source file and is stored in the specified destination file by overwriting its previous contents.
OUTPUT :

```
[DeepaSharan@webminal.org ~]$cp deepa.txt sharanya.txt
[DeepaSharan@webminal.org ~]$cat deepa.txt
Hello i am deepa
I am new to linux
[DeepaSharan@webminal.org ~]cat sharanya.txt
Hello i am deepa
I am new to linux
[DeepaSharan@webminal.org ~]$
```

19. TASK : To rename or move a file.
COMMAND : mv
SYNTAX : mv sourcefile destinationfile.
EXPLANATION : After moving the contents of the source file into destination file, the source file is deleted.
OUTPUT :

```
[DeepaSharan@webminal.org ~]$mv deepa.txt deepaSharan.txt
[DeepaSharan@webminal.org ~]$cat deepa.txt
cat: deepa.txt: No such file or directory
```

20. TASK : To delete a file.

COMMAND : rm
SYNTAX : rm file name.
EXPLANATION : This command deletes the specified file from the directory.
OUTPUT :

```
[DeepaSharan@webminal.org ~]$rm sharanya.txt
[DeepaSharan@webminal.org ~]$ls
deepaFile  deepaSharan.txt  file1
```

21. TASK : To retrieve a part of a file.
COMMAND : head.
SYNTAX : head -no. of rows file name.
EXPLANATION : This command displays no. of rows from the top of the specified file.
OUTPUT :

```
[DeepaSharan@webminal.org ~]$head -2 deepaSharan.txt
Hello i am deepa
I am new to linux
```

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

COMMAND : tail.
SYNTAX : tail -no. of rows file name.
EXPLANATION : This command displays no. of rows from the bottom of the specified file.
OUTPUT :

```
[DeepaSharan@webminal.org ~]$tail -2 deepaSharan.txt
Hello i am deepa
I am new to linux
```

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

COMMAND : sort.
SYNTAX : sort file name.
EXPLANATION : This command sorts the contents of a file in ascending order.
OUTPUT :

```
[DeepaSharan@webminal.org ~]$sort deepaSharan.txt
Hello i am deepa
I am new to linux
```

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

COMMAND : wc
SYNTAX : wc file name.
EXPLANATION : This command displays the no. of lines, words, and characters of the file.
OUTPUT :

```
[DeepaSharan@webminal.org ~]$wc deepaSharan.txt
 11 53 deepaSharan.txt
```

25. TASK : To add line number to file content.
COMMAND : nl
SYNTAX : nl filename /add number to the file content
EXPLANATION : This command used to add line number to file content.
OUTPUT :

```
[DeepaSharan@webminal.org ~]$cp deepa.txt sharanya.txt
[DeepaSharan@webminal.org ~]$cat deepa.txt
Hello i am deepa
I am new to linux
[DeepaSharan@webminal.org ~]cat sharanya.txt
Hello i am deepa
I am new to linux
[DeepaSharan@webminal.org ~]$
```

26. TASK : To compress a given file or a directory.
COMMAND : gzip
SYNTAX : gzip file name.
EXPLANATION : This command compresses a given file or directory.
OUTPUT :

```
[DeepaSharan@webminal.org ~]$gzip deepaSharan.txt
[DeepaSharan@webminal.org ~]$cat deepaSharan.txt
cat: deepaSharan.txt: No such file or directory
[DeepaSharan@webminal.org ~]$gunzip deepaSharan.txt
```

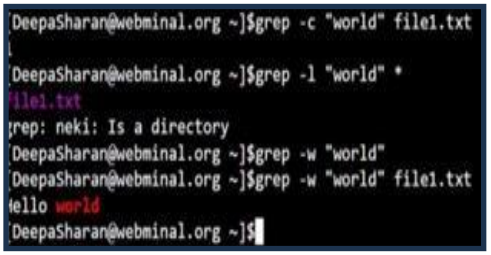
27. TASK : To uncompress a given file or a directory.
COMMAND : gunzip
SYNTAX : gunzip file name.
EXPLANATION : This command uncompress a given file or directory.
OUTPUT :

```
[DeepaSharan@webminal.org ~]$gunzip deepaSharan.txt
[DeepaSharan@webminal.org ~]$cat deepaSharan.txt
Hello i am deepa
I am new to linux
Almighty
Equation
[DeepaSharan@webminal.org ~]$
```

28. TASK : To compare the contents of two files.
COMMAND : cmp
SYNTAX : cmp file1 file2
EXPLANATION : This command compares a given file and displays the area at which it differs
OUTPUT :

29. TASK : To find the difference between two files.
COMMAND : diff
SYNTAX : diff file1 file2
EXPLANATION : This command compares a given file and displays the area at which it differs i.e line no and characters etc.
OUTPUT :

30. TASK : To match the given pattern.
COMMAND : grep
SYNTAX : $grep (option) text filename /search and print given text from file
Options
-c : This prints only a count of the lines that match a pattern.
-h : Display the matched lines, but do not display the filenames.
-i : Ignores, case for matching.
-l: Displays list of a filenames only.
-n : Display the matched lines and their line numbers
-o: print only the matched parts of a matching line
EXPLANATION : This command verifies the file name and checks whither the pattern is present in the file or not.
OUTPUT :

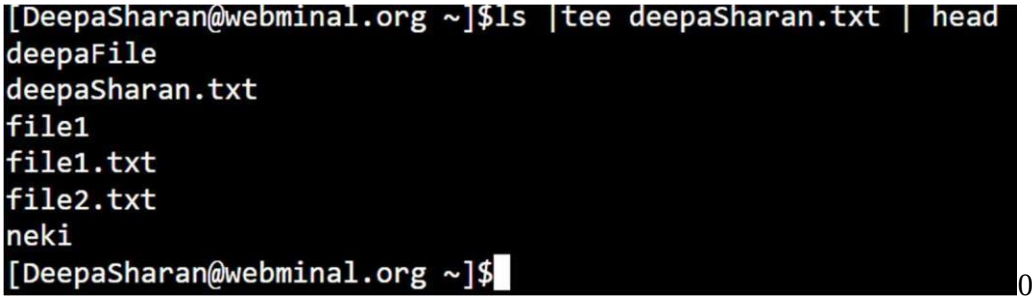


31. TASK : To mechanism by which the output of one command can be channeled into the input of another command.

COMMAND : pipe
SYNTAX : cat filename | sort | nl
EXPLANATION : This command selects a given file, sorts it and displays the content with line number.
OUTPUT :

32. TASK : To save intermediate results in a file

COMMAND : tee
SYNTAX : ls | tee filename | head
EXPLANATION : This command saves the intermediate results in a file.
OUTPUT



0

RESULT: Thus the analysis and the synthesis of the basic linux commands is successfully executed**.**

Ex.No : 2 **IMPLEMENTATION OF SHELL PROGRAMMING**

Date :

AIM:

To write a programs using shell programming.

DESCRIPTION:

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

What is a shell script?

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

SYNTAX

1. EXPRESSION Command:

To perform all arithmetic operations.

Syntax:

var = „expr $value1 + $value2"

or

var = $(($value1 + $value2))

Example:

```
a=10
b=20
sum=$(( $a + $b ))
echo $sum
```

2. OPERATORS:

Shell uses the built-in test command operators to test numbers and strings.

Equality:

=	string
!=	string
-eq	number
-ne	number

ARITHMETIC OPERATORS OPERATOR	DESCRIPTION	EXAMPLE
+ Addition	Adds value on either side of the operator	`expr $a + $b` will give 30
-Subtraction	Subtracts right hand operand from left hand operand	`expr $a - $b` will give -10
* (Multiplication)	Multiplies values on either side of the operator	`expr $a \* $b` will give 200
/ (Division)	Divides left hand operand by right hand operand	`expr $b / $a` will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder	`expr $b % $a` will give 0
= (Assignment)	Assigns right operand in left operand	a = $b would assign value of b into a

RELATIONAL OPERATORS OPERATOR	DESCRIPTION	EXAMPLE
-eq	Checks if the value of two operands are equal or not; ifyes, then the condition becomes true.	[$a -eq $b] is not true.
-ne	Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true.	[$a -ne $b] is true.
-gt	Checks if the value of left operand is greater than the value of right operand; if yes,then the condition becomes true.	[$a -gt $b] is not true.
-lt	Checks if the value of left operand is less than the valueof right operand; if yes, then the condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is greater than orequal to the value of right	[$a -ge $b] is not true

BOOLEAN OPERATORS OPERATOR	DESCRIPTION	EXAMPLE
!	This is logical negation. This inverts a true condition intofalse and vice versa.	[! false] is true.
-o	This is logical **OR**. If one of the operands is true, then the condition becomes true.	[$a -lt 20 -o $b -gt 100] is true
-a	This is logical **AND**. If both the operands are true, then thecondition becomes true otherwise false.	[$a -lt 20 -a $b -gt 100] is false.

STRING OPERATORS	DESCRIPTION	EXAMPLE
=	Checks if the value of two operands are equal or not; ifyes, then the condition becomes true.	[$a = $b] is not true
!=	Checks if the value of two operands are equal or not; ifvalues are not equal then thecondition becomes true.	[$a != $b] is true.
-z	Checks if the given string operand size is zero; if it is zerolength, then it returns true.	[-z $a] is not true.
-n	Checks if the given string operand size is non-zero; if it isnonzero length, then it returns true.	[-n $a] is not false.
str	Checks if str is not the emptystring; if it is empty, then it returns false.	[$a] is not false.

3. DESCISION MAKING STATEMENTS

The **if...else** statements
Example

```
if [ $a -gt $b ]
then
echo “$a is Big”
else
echo “$b is Big”
fi
```

The **case...esac** statement
Example

```
echo "1.print 2.Exit"
read op
case $op in
1)echo “Hello”;;
2)exit
esac
```

4. LOOPING STATEMENTS

for loop statements

Example 1

```
for i in 1
2do
echo
“welcome”
done
```

Example 2

```
n=10
for (( i=0; i<n; i+))
do
echo
$idone
```

while loop statements

Example

```
a=0
while [ $a -lt 10]
do
echo  $a
a=`expr $a +
1`done
```

Note:

i) echo –n

-n option lets echo avoid printing a new line character.

ii) To calculate more than two numbers use

echoExample:

```
x=
3
y=
6
z=9
echo "$x+$y+$z" | bc
```

PROGRAMS:

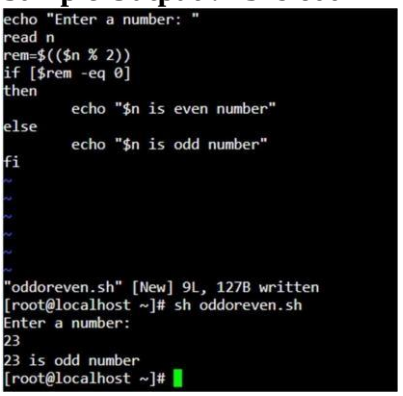
1. **Write a Shell program to check the given number is even or odd.**

   Sample Input :

   Enter any number

   23

   Sample Output : 23 is odd



Result: Thus program using shell programming successfully executed and verified

2. **Write a Shell program to check the given number and its reverse are same.**

Sample Input :
Enter a
number252
Sample Output : The given number and its reverse are same

```
echo "Enter the number: "
read n
number=$n
reverse=0
while [$n -gt 0]
do
a = `expr $n %10`
n = `expr $n / 10`
reverse = `expr $reverse \*10 + $a `
done
echo $reverse
if [$number -eq $reverse]
then
echo "The Number is same as the Reverse"
else
echo "The Number is NOT same as the Reverse"
fi
~
~
~
~
~
[root@localhost ~]# sh reverse.sh
Enter the number:
12345
The Number is NOT same as the Reverse
[root@localhost ~]#
```

Result: Thus program using shell programming successfully executed and verified

3. **Write a Shell Program to find a factorial of a number.**
   Sample Input :
   Enter number : 5
   Sample Output : 120

```
read -p "Enter a number: "
fact=1
while [$num -gt 1]
do
fact=$((fact*num))
num=$((num-1))
done
echo $fact

~
~
~
~
~
~
~
~
"fact.sh" [New] 9L, 107B written
[root@localhost ~]# sh fact.sh
Enter a number: 4
24
```

Result: Thus program using shell programming successfully executed and verified

4. **Write a Shell Program for finding sum of Odd and Even numbers up to 'N'.**

Sample Input :

Enter the number of elements

5

Enter the number

23

34

45

56

67

Sample Output :

The sum of odd numbers is : 135

The sum of even numbers is : 90

```
read -p "Enter a number: "
fact=1
while [$num -gt 1]
do
fact=$((fact*num))
num=$((num-1))
echo "Enter n value: "
read n
sumodd=0
sumeven=0
i=0
while [&i -ne $n]
do
echo "Enter Number: "
read num
if [`expr $num % 2` -ne 0]
then
sumodd = `expr $sumodd + $num`
sumeven = `expr $sumeven + $num`
fi
i = `expr $i +1`
done
echo "Sum of odd numbers = $sumodd"
echo "Sum of even numbers = $sumeven"
~
~
~
~
~
~
~
~
~
~
"fifty.sh" [New] 18L, 300B written
[root@localhost ~]# sh fifty.sh
Enter n value:
5
Enter Number:
23
34
45
56
67
Sum of odd numbers = 135
Sum of even numbers = 90
```

Result: Thus program using shell programming successfully executed and verified

5. **Write a Shell program to display student grades.**

Sample Output :

Roll no.	Name	Total	Average	Grade
19skcet001	Anil	201	67	First class

```
echo "Enter Student name :"
read name
echo "Enter the Reg.no :"
read rno
echo "Enter Mark1 :"
read m1
echo "Enter Mark2 :"
read m2
echo "Enter Mark3 :"
read m3
tot = $(expr $m1 + $m2 + $m3)
avg = $(expr $tot/3)
echo "Student Name : $name"
echo "Reg.no : $rno"
echo "Total : $tot"
echo "Average : $avg"
if [$m1 -ge 65] && [$m2 -ge 65] && [m3 -ge 65]
then
echo "Grade : First Class"
else
echo "Grade : NOT First Class"
~
~
~
~
~
~
"student.sh" [New] 21L, 418B written
[root@localhost ~]# sh student.sh
Enter Student name :
Arthika
Enter the Reg.no :
20EUCS015
Enter Mark1 :
75
Enter Mark2 :
85
Enter Mark3 :
95
Student Name : Arthika
Reg.no : 20EUCS015
Total : 255
Average : 85
Grade : First Class
```

Result: Thus program using shell programming successfully executed and verified

6. **Write a Shell Program to have to print half pyramid using for loop.**

Sample Input : n=10

Sample Output :

```
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
```

```
echo "Enter a number :"
read rows
for ((i=1 ; i<=rows ; i++))
do
 for ((j=1 ; j<=i ; j++))
  do
   echo -n "* "
  done
 echo
done
~
~
~
~
~
~
~
~
"pattern.sh" [New] 10L, 130B written
[root@localhost ~]# sh pattern.sh
Enter a number :
10
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
[root@localhost ~]#
```

Result: Thus program using shell programming successfully executed and verified

7. Write a Shell Program to perform Arithmetic Operation using Case statement.

Sample Input :

Enter two number
2 3
1. Add 2.Sub 3.Mul 4.Div 5.Exit
Enter the option: 1

Sample Output :

Addition is 5

```
echo "Enter 2  numbers :"
read a b
echo "Enter Your Choice :"
echo "1)Add 2)Sub 3)Mul 4)Div 5)Exit"
read n
case "$n" in
        1) echo "Sum : `expr $a + $b`";;
        2) echo "Difference : `expr $a - $b`";;
        3) echo "Product : `expr $a \* $b`";;
        4) echo "Quotient : `expr $a / $b`";;
esac
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"menu.sh" 11L, 278B written
[root@localhost ~]# sh menu.sh
Enter 2  numbers :
2 3
Enter Your Choice :
1)Add 2)Sub 3)Mul 4)Div 5)Exit
3
Product : 6
[root@localhost ~]#
```

Result: Thus the programs using Shell programming is successfully executed.

8. **Write a Shell Program for comparison of strings.**

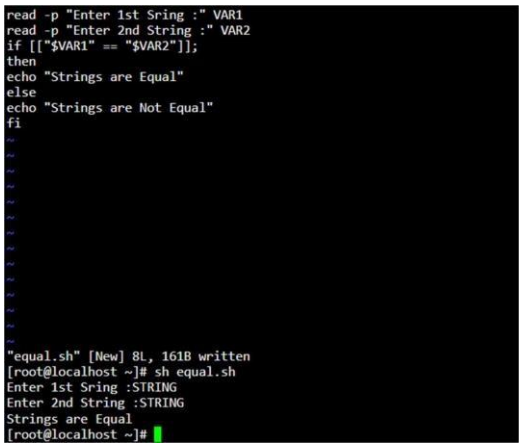
   Sample Input :

   Enter string 1: OS

   Enter string 2: OS

   Sample Output :

   String 1 and String 2 are identical



Result: Thus the programs using Shell programming is successfully executed**.**

9. **Write a Shell program to find the smallest number from a set of numbers.**

Sample Input :

Enter the number of elements: 5

Enter the numbers:

34

23

45

37

56

Sample Output :

The smallest number is : 23

```
echo "Enter the No.of Elements :"
read n
for ((i=0 ; i<n ; i++))
        do
                read nos[$i]
        done
if [${nos[$i]} -lt $small ]; then
        small = ${nos[$i]}
fi

echo "Smallest Number : $small"
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"small.sh" 11L, 180B written
[root@localhost ~]# sh small.sh
Enter the No.of Elements :
3
21
34
5
Smallest Number : 5
[root@localhost ~]#
```

RESULT: Thus the programs using Shell programming is successfully executed.

10. **Write a Shell program to find the sum of all numbers between 50 and 100, which are divisible by 3 and not divisible by 5.**

Sample Output :

The sum of all numbers between 50 and 100, which are divisible by 3 and not divisible by 5 are:
51
54
57
63
66
69
72
78
81
84
87
93
96
99

```
for ((i=50 ; i<=100 ; i++)) do
        if [`expr $i %3` = 0 -a `expr $i % 5` !=0 ] then
                echo $i
        fi
done
~
~
~
~
~
~
~
~
~
~
~
"sum1.sh" 14L, 112B written
[root@localhost ~]# sh sum1.sh
51
54
57
63
66
69
72
78
81
84
87
93
96
99
[root@localhost ~]#
```

Result:

Thus the programs using Shell programming is successfully executed.

EX NO: 3 **IMPLEMENTATION OF UNIX SYSTEM CALLS**
DATE:

DESCRIPTION:

When a computer is turned on, the program that gets executed first is called the "operating system". It controls pretty much all activity in the computer. This includes who logsin, how disks are used, how memory is used, how the CPU is used, and how you talk with othercomputers. The operating system we use is called "Unix".

The way that programs talk to the operating system is via ``system calls." A system call looks like a procedure call but it's different -- it is a request to the operating system to perform some activity.

1. fork( )

Fork system call used for creating a new process, which is called ***child process***, which runs concurrently with a process (which process called system call fork) and this process is called ***parent process***. It takes no parameters and returns an integer value.
Negative Value: creation of a child process was unsuccessful.
Zero: Returned to the newly created child process.
Positive value: Returned to parent or caller. The value contains process ID of newly created child process
Syntax: fork( )

2. wait( )

A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After the child process terminates, the parent ***continues*** its execution after the wait system call instruction.
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)

4. getpid( )

It returns the process ID of the current process.

Syntax : getpid()

5. getppid( )

It returns the process ID of the parent of the current process.

Syntax : getppid()

6. perror( )

Indicate the process error.

Syntax : perror()

7. opendir( )

Open a directory.

Syntax : opendir()

8. readdir( )

Read a directory.

Syntax : readdir()

9. closedir()

Close a directory.

Syntax : closedir()

10. open()

It is used to open a file for reading, writing or both

Syntax : open(const char* path, int flags [, int mode])

Modes

O_RDONLY: read only, **O_WRONLY**: write only, **O_RDWR**: read and write, **O_CREAT**: create file if it doesn‟t exist, **O_EXCL**: prevent creation if it already exists.

Example:

```
int fd = open("foo.txt", O_RDONLY | O_CREAT);
printf("fd = %d/n", fd);
```

11. close ()

Tell the operating system you are done with a file descriptor and Close the file which is pointed by fd.

Syntax: close(fd);

12. read ()

The file indicated by the file descriptor fd, the read() function reads cnt bytes of input into the memory area indicated by buf. To read data from is said to be **buf.** A successful read() updates the access time for the file.

fd: file descriptor

buf: buffer to read data from

cnt: length of buffer

Syntax: read(int fd, void* buf,size_t cnt);

Example:

```
int fd = open("foo.txt", O_RDONLY);
read(fd, &c, 1);
```

13. Write()

Writes cnt bytes from buf to the file or socket associated with fd. cnt should not be greater than INT_MAX (defined in the limits.h header file). If cnt is zero, write() simply returns 0 without attempting any other action.

Syntax : write(int fd, void* buf, size_t cnt);

Example:

```
int fd = open("foo.txt", O_WRONLY | O_CREAT );
write(fd, "hello world", strlen("hello world"));
```

PROGRAMS

1) Implementation of process management using the following system calls of the UNIX operating system: **fork, wait, exec, getpid, getppid and exit.**

AIM: To implement the process management system call commands of fork, wait, exec, getpid, getppid and exit.

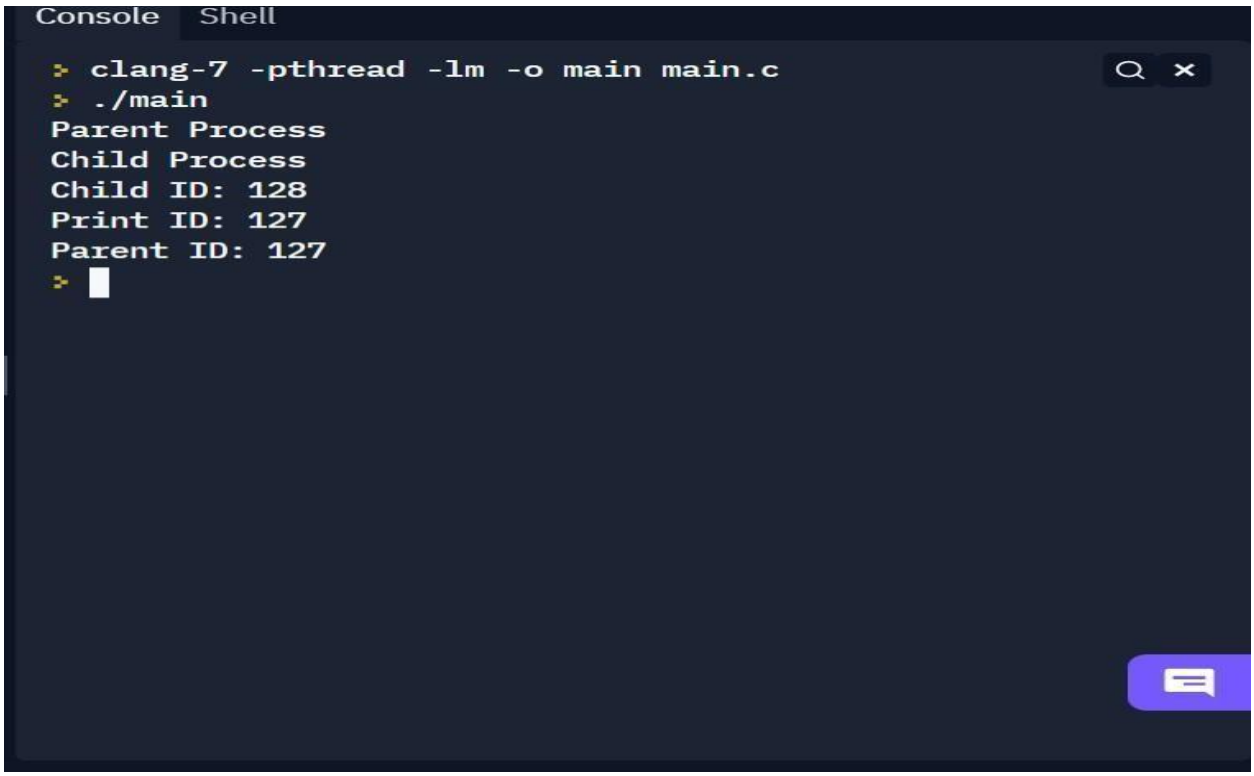
ALGORITHM:

STEP1: Declare the variable.
STEP2: Assign value to the variable by calling the function fork().
STEP3: Check if the variable is lesser than 0 and print Error.
STEP4: Else check if the variable is equal to 0 and print getpid() and getppid() and exit.
STEP5: Else call wait() function with NULL parameter, then print getpid().

PROGRAM:

```c
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
int main()
{
  int pid;
  pid=fork();
  if(pid<0)
  printf("Error");
  else if(pid==0)
  {
    printf("Child Process\n");
    printf("Child ID: %d\n",getpid());
    printf("Print ID: %d\n",getppid());
    exit(0);
  }
  else
  {
    printf("Parent Process\n");
    wait(NULL);
    printf("Parent ID: %d\n",getpid());
  }
  return 0;
}
```

OUTPUT:



RESULT:

The above program has been executed and output has been verified.

2) Implementation of directory management using the following system calls of the UNIX operating system: **opendir, readdir, closedir.**

AIM:

To implement the directory management using unix system calls of open,close and read directories.

ALGORITHM:

STEP1: Declare pointers dirp and dp.

STEP2: Check if drip is NULL and print can‟t open the file.

STEP3: Set a for loop until dp is not NULL.

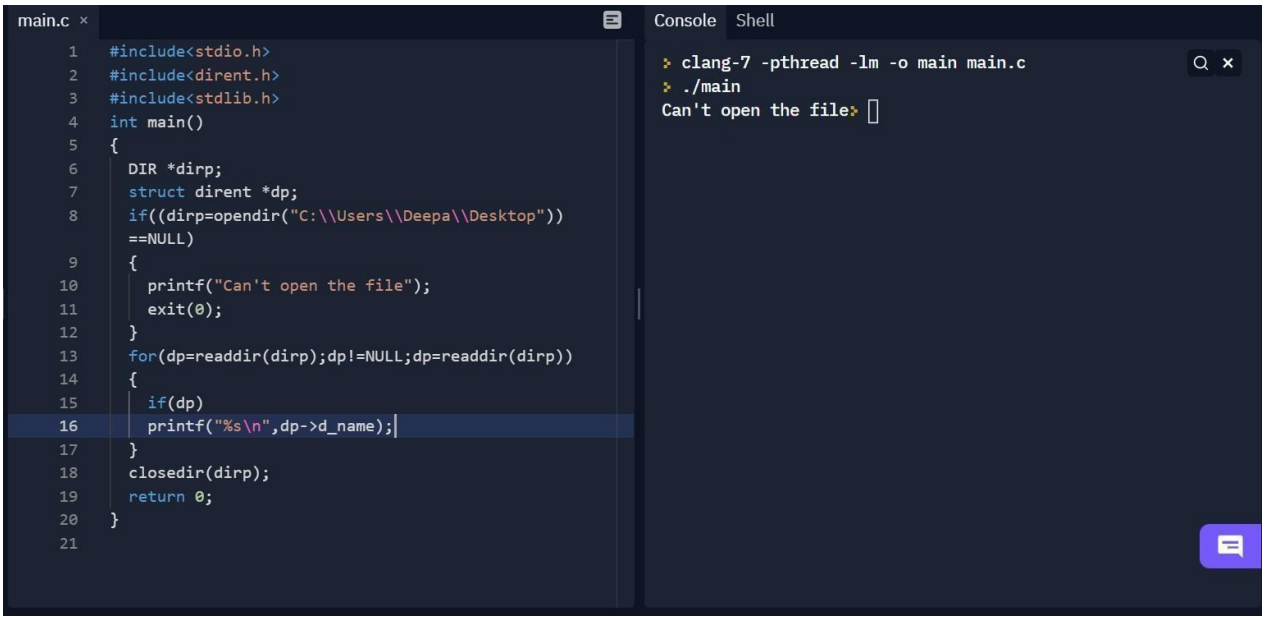
STEP4: Check dp and print the name pointed by the pointer dp.

STEP5: Close directory dirp.

PROGRAM:

```
#include<stdio.h>
#include<dirent.h>
#include<stdlib.h>
int main()
{
  DIR *dirp;
  struct dirent *dp;
  if((dirp=opendir("C:\\Users\\Deepa\\Desktop"))==NULL)
  {
    printf("Can't open the file");
    exit(0);
  }
  for(dp=readdir(dirp);dp!=NULL;dp=readdir(dirp))
  {
    if(dp)
    printf("%s\n",dp->d_name);
  }
  closedir(dirp);
  return 0;
}
```

OUTPUT:



RESULT:

The above program has been executed and output has been verified.

3) Write a program using the I/O system calls of UNIX operating system: **open, read, write,close.**

AIM:

To implement the program using input and output system calls of unix operating system.

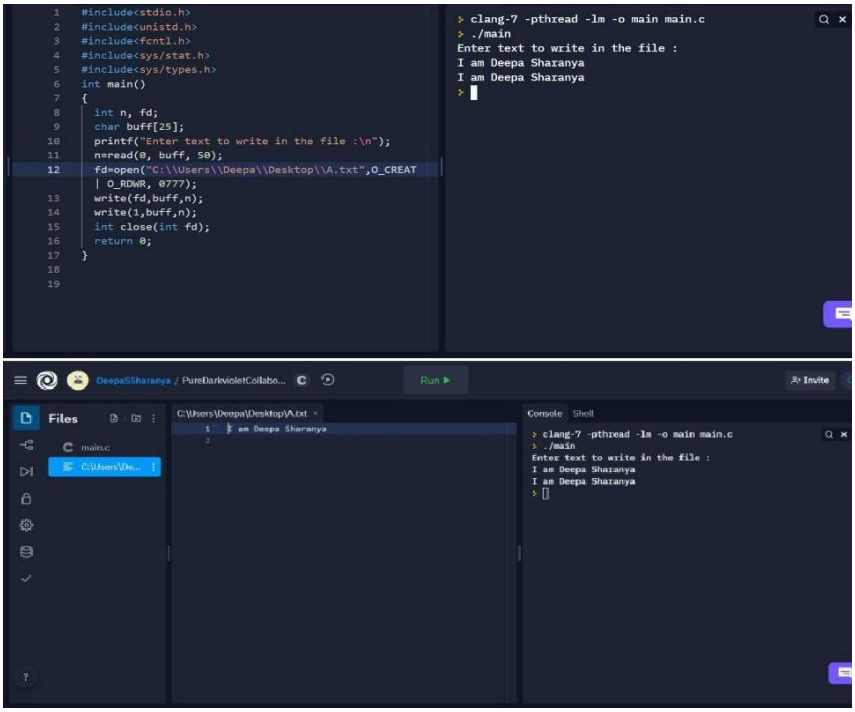
ALGORITHM:

STEP1: Declare variables n, fd and an array buff with size 25.
STEP2: Get text from the user and assign it to variable n.
STEP3: Open the text file on the variable fd.
STEP4: Write the text got from the user at the file opened at variable fd.
STEP5: Close the file opened at fd.

PROGRAM:

```c
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<sys/types.h>
int main()
{
  int n, fd;
  char buff[25];
  printf("Enter text to write in the file :\n");
  n=read(0, buff, 50);
  fd=open("C:\\Users\\Deepa\\Desktop\\A.txt",O_CREAT | O_RDWR, 0777);
  write(fd,buff,n);
  write(1,buff,n);
  int close(int fd);
  return 0;
}
```

OUTPUT:



RESULT:

The above program has been executed and output has been verified.

EX NO : 4 IMPLEMENTATION OF NON PRE-EMPTIVE AND PRE-EMPTIVE SCHELDULING ALGORITHM

DATE :

AIM :

To simulate and analyze the operation of CPU scheduling using FCFS,SJF, Priority and Round Robin algorithms.

DESCRIPTION :

When a computer is multi-programmed, it frequently has multiple processes competing forthe CPU at the same time frequently. This situation occurs whenever two or more processes are simultaneously in the ready state. If only one CPU is available, a choice has to be made which process has to be in the CPU. The part of the operating system that makes the choice is called thescheduler and the algorithm is called scheduling algorithm.

In the FCFS algorithm the process which arrives first is given the c CPU after finishing itsrequest;only it will allow the CPU to execute another process. In the SJF algorithm the process which has less service time given the CPU after finishing its request only will allow the CPU to execute the next other process. In priority scheduling, processes are allocated to the CPU on the basis of an externally assigned priority. The key to the performance of priority scheduling is in choosing priorities for the processes. In the Round Robin algorithm we are assigning some time slice .The process is allocated according to the time slice, if the process service time is less than the time slice then the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue .If the CPU burst of the currently running process is longer than time quantum; the timer will go off and will cause an interrupt to the operating system. A context switch will be executed and the process will be put at the tail of the ready queue.

PROGRAM :

a) NON PRE-EMPTIVE FIRST COME FIRST SERVE

```
#include<stdio.h>
void findWaitingTime(int processes[], int n, int bt[], int wt[])
{
wt[0] = 0;

for (int i = 1; i < n ; i++ ) wt[i] = bt[i-1] + wt[i-1];
}

void findTurnAroundTime( int processes[], int n, int bt[], int wt[], int tat[])
{
for (int i = 0; i < n ; i++) tat[i] = bt[i] + wt[i];
}
void findavgTime( int processes[], int n, int bt[])
{
int wt[n], tat[n], total_wt = 0, total_tat = 0; findWaitingTime(processes, n, bt, wt);
findTurnAroundTime(processes, n, bt, wt, tat);
printf("Processes Burst time Waiting time Turn around time\n"); for (int i=0; i<n; i++)
total_wt = total_wt + wt[i]; total_tat = total_tat + tat[i]; printf(" %d
",(i+1));printf(" %d ", bt[i] );
printf(" %d",wt[i] );
printf(" %d\n",tat[i] );
}
float s=(float)total_wt / (float)n;
float t=(float)total_tat / (float)n;
      printf("Average waiting time = %f",s);
      printf("\n");
      printf("Average turn around time = %f ",t);
  }
  int main()
  {
      int processes[] = { 1, 2, 3, 4};
      int n = sizeof processes / sizeof processes[0];int

      burst_time[] = {21, 3, 6, 2};

      findavgTime(processes, n, burst_time); return 0;

  }
```

b) NON PRE-EMPTIVE SHORTEST JOB FIRST

```
#include<stdio.h>
 int main()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;float
    avg_wt,avg_tat;
    printf("Enter number of process:");
    scanf("%d",&n);
    printf("\nEnter Burst Time:\n");
    for(i=0;i<n;i++)
    {
        printf("P%d:",i+1);
        scanf("%d",&bt[i]);
        p[i]=i+1;
    }

    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
                pos=j;
        }
        temp=bt[i];
```

```
    bt[i]=bt[pos];
    bt[pos]=temp;
    temp=p[i];
    p[i]=p[pos];
    p[pos]=temp;
}
wt[0]=0;
for(i=1;i<n;i++)
{
    wt[i]=0;
    for(j=0;j<i;j++)
        wt[i]+=bt[j];
    total+=wt[i];
}
avg_wt=(float)total/n;total=0;
printf("\nProcess\t       Burst Time      \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];
    total+=tat[i];
    printf("\np%d\t\t  %d\t\t        %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
}
avg_tat=(float)total/n;
printf("\n\nAverage Waiting Time=%f",avg_wt);
```

```
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

c) PRE-EMPTIVE SHORTEST JOB FIRST (SHORTEST REMAINING TIME FIRST)

```
#include <stdio.h>
int main()
{
 int a[10],b[10],x[10],i,j,smallest,count=0,time,n;
 double avg=0,tt=0,end;
  printf("Enter the number of Processes : ");
  scanf("%d",&n);
 printf("Enter the Arrival Time\n");
 for(i=0;i<n;i++) scanf("%d",&a[i]);
 printf("Enter the Burst Time\n");
 for(i=0;i<n;i++)
 scanf("%d",&b[i]);
 for(i=0;i<n;i++)
 x[i]=b[i];

  b[9]=9999;

 for(time=0;count!=n;time++)
 {
   smallest=9;
```

```
 for(i=0;i<n;i++)
 {
  if(a[i]<=time && b[i]<b[smallest] && b[i]>0 )
  smallest=i;
 }
 b[smallest]--;
 if(b[smallest]==0)
 {
  count++;
  end=time+1;
  avg=avg+end-a[smallest]-x[smallest];tt=
  tt+end-a[smallest];
 }
}
 printf("\n\nAverage waiting time = %lf\n",avg/n);
    printf("Average Turnaround time = %lf",tt/n);
    return 0;
}
```

d) NON PRE-EMPTIVE PRIORITY SCHEDULING

```
#include<stdio.h>
int main()
{
    int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;printf("Enter
    Total Number of Process:");
```

```
scanf("%d",&n);

printf("\nEnter Burst Time and Priority\n");
for(i=0;i<n;i++)
{
    printf("\nP[%d]\n",i+1);
    printf("Burst Time:");
    scanf("%d",&bt[i]);
    printf("Priority:");
    scanf("%d",&pr[i]);
    p[i]=i+1;
}

for(i=0;i<n;i++)
{
    pos=i;
    for(j=i+1;j<n;j++)
    {
        if(pr[j]<pr[pos])
        pos=j;
    }
    temp=pr[i];
    pr[i]=pr[pos];
    pr[pos]=temp;
```

```
    temp=bt[i];
    bt[i]=bt[pos];
    bt[pos]=temp;

    temp=p[i];
    p[i]=p[pos];
    p[pos]=temp;
}

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

    total+=wt[i];
}

avg_wt=total/n;
total=0;
printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];
```

```
        total+=tat[i]; printf("\nP[%d]\t\t%d\t\t%d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }

    avg_tat=total/n;

    printf("\n\nAverage Waiting Time=%d",avg_wt);

    printf("\nAverage Turnaround Time=%d\n",avg_tat)

            return 0;

}
```

e) PRE-EMPTIVE PRIORITY SCHEDULING

```
#include<iostream>

using namespace std;

int main()

{

    int a[10],b[10],x[10];

    int waiting[10],turnaround[10],completion[10],p[10];int

    i,j,smallest,count=0,time,n;

    double avg=0,tt=0,end;

   cout<<"\nEnter the number of Processes: ";

    cin>>n;

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

    {

      cout<<"\nEnter arrival time of process: ";

      cin>>a[i];

    }

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

    {

      cout<<"\nEnter burst time of process: ";

      cin>>b[i];
```

```
}
for(i=0;i<n;i++)
{
  cout<<"\nEnter priority of process: ";
  cin>>p[i];
}
for(i=0; i<n; i++)
    x[i]=b[i];

p[9]=-1;
for(time=0; count!=n; time++)
{
    smallest=9; for(i=0;
    i<n; i++)
    {
        if(a[i]<=time && p[i]>p[smallest] && b[i]>0 )
            smallest=i;
    }
    b[smallest]--;
    if(b[smallest]==0)
    {
        count++;
        end=time+1;
        completion[smallest] = end;
        waiting[smallest] = end - a[smallest] - x[smallest];
        turnaround[smallest] = end - a[smallest];
    }
}
```

```
     cout<<"Process"<<"\t"<< "burst-time"<<"\t"<<"arrival-time" <<"\t"<<"waiting-time"
<<"\t"<<"turnaround-time"<< "\t"<<"completion-time"<<"\t"<<"Priority"<<endl;for(i=0;

    i<n; i++)

    {


cout<<"p"<<i+1<<"\t\t"<<x[i]<<"\t\t"<<a[i]<<"\t\t"<<waiting[i]<<"\t\t"<<turnaround[i]<<"\t\t"
<<completion[i]<<"\t\t"<<p[i]<<endl;avg

        = avg + waiting[i];

        tt = tt + turnaround[i];


    }

   cout<<"\n\nAverage waiting time ="<<avg/n; cout<<"

    Average Turnaround time ="<<tt/n<<endl;

}
```

f) PRE-EMPTIVE ROUND ROBIN SCHEDULING

```
#include<stdio.h>

int main()
{
    int i, limit, total = 0, x, counter = 0, time_quantum;
    int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10], temp[10];float
    average_wait_time, average_turnaround_time;
    printf("\nEnter Total Number of Processes:\t");
    scanf("%d", &limit);
    x = limit;
    for(i = 0; i < limit; i++)
    {
        printf("\nEnter Details of Process[%d]\n", i + 1);
        printf("Arrival Time:\t");
        scanf("%d", &arrival_time[i]);
        printf("Burst Time:\t");
        scanf("%d", &burst_time[i]);
        temp[i] = burst_time[i];
    }
        if(i == limit - 1)
        {
            i = 0;
        }
        else if(arrival_time[i + 1] <= total)
        {
            i++;
        }
        else
```

```
        {
            i = 0;

        }

    }
    average_wait_time = wait_time * 1.0 / limit; average_turnaround_time =

    turnaround_time * 1.0 / limit; printf("\n\nAverage Waiting Time:\t%f",

    average_wait_time); printf("\nAvg Turnaround Time:\t%f\n",

    average_turnaround_time);return 0;

}
```

SAMPLE INPUT:

PROCESS	PRIORITY	BURST TIME
P1	**2**	**21**
P2	**1**	**3**
P3	**4**	**6**
P4	**3**	**2**

OUTPUTS :

a) NON PRE-EMPTIVE FIRST COME FIRST SERVE

```
Processes   Burst time   Waiting time   Turn around time
   1            21            0              21
   2            3             21             24
   3            6             24             30
   4            2             30             32
Average waiting time = 18.750000
Average turn around time = 26.750000
```

b) NON PRE-EMPTIVE SHORTEST JOB FIRST

```
Enter number of process:4

Enter Burst Time:
P1:21
P2:3
P3:6
P4:2

Process     Burst Time          Waiting Time    Turnaround Time
p4               2                   0               2
p2               3                   2               5
p3               6                   5               11
p1               21                  11              32

Average Waiting Time=4.500000
Average Turnaround Time=12.500000
```

c) PRE-EMPTIVE SHORT JOB FIRST (SHORTEST REMAINING TIME FIRST

```
Enter the number of Processes : 4
Enter the Arrival Time
0 0 0 0
Enter the Burst Time
21 3 6 2


Average waiting time = 4.500000
Average Turnaround time = 12.500000
```

d) NON PRE-EMPTIVE PRIORITY SCHEDULING ALGORITHM

```
Enter Total Number of Process:4

Enter Burst Time and Priority

P[1]
Burst Time:21
Priority:2

P[2]
Burst Time:3
Priority:1

P[3]
Burst Time:6
Priority:4

P[4]
Burst Time:2
Priority:3

Process         Burst Time      Waiting Time    Turnaround Time
P[2]            3               0                       3
P[1]            21              3                       24
P[4]            2               24                      26
P[3]            6               26                      32

Average Waiting Time=13
Average Turnaround Time=21
```

e) PRE-EMPTIVE ROUND ROBIN SCHEDULING ALGORITHM

```
Enter Total Number of Processes:        4

Enter Details of Process[1]
Arrival Time:   0
Burst Time:     21

Enter Details of Process[2]
Arrival Time:   0
Burst Time:     3

Enter Details of Process[3]
Arrival Time:   0
Burst Time:     6

Enter Details of Process[4]
Arrival Time:   0
Burst Time:     2

Enter Time Quantum:     2

Process ID              Burst Time      Turnaround Time         Waiting Time

Process[4]              2               8                       6
Process[2]              3               11                      8
Process[3]              6               17                      11
Process[1]              21              32                      11

Average Waiting Time:   9.000000
Avg Turnaround Time:    17.000000
```

RESULT :

Thus, the operations of CPU scheduling using FCFS, SJF, Priority and Round Robin algorithms are simulatedand analysed successfully.

EX NO: 5 **IMPLEMENTATION OF DINING PHILOSOPHERS' PROBLEM TO DEMONSTRATE PROCESS SYNCHRONIZATION**

DATE :

AIM:

To implement the dining philosopher's problem to demonstrate process synchronization.

ALGORITHM:

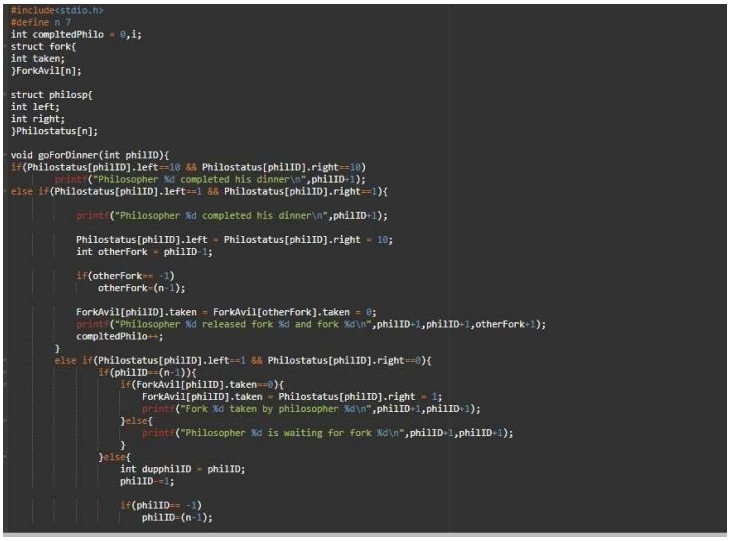
1. Start
2. Get the total number of philosophers and number of hungry philosophers from the user.
3. Get the positions of each philosopher.
4. In switch cases, one philosopher will eat at a time or two.
5. Based on the number of philosophers, a process will be granted to eat, if any other process is already eating, then it should wait.
6. Stop

PROGRAM:

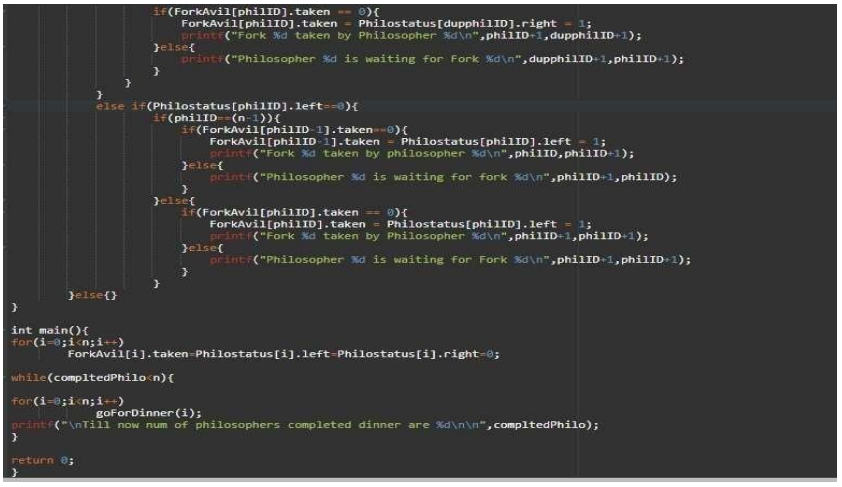
```
#include<stdio.h>

#define n 7

int compltedPhilo = 0,i;

struct fork{
int taken;
}ForkAvil[n];

struct philosp{
int left;
int right;
}Philostatus[n];

void goForDinner(int philID){
if(Philostatus[philID].left==10 && Philostatus[philID].right==10)
      printf("Philosopher %d completed his dinner\n",philID+1);
//if already completed dinner
else if(Philostatus[philID].left==1 && Philostatus[philID].right==1){
         //if just taken two forks
         printf("Philosopher %d completed his dinner\n",philID+1);
```

```
        Philostatus[philID].left = Philostatus[philID].right = 10;
        int otherFork = philID-1;

        if(otherFork== -1)
           otherFork=(n-1);
        ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0;
                                    printf("Philosopher %d released fork %d and fork
%d\n",philID+1,philID+1,otherFork+1);
        compltedPhilo++;
      }
       else if(Philostatus[philID].left==1 && Philostatus[philID].right==0){ //left already taken,
trying for right fork
          if(philID==(n-1)){
             if(ForkAvil[philID].taken==0){
                ForkAvil[philID].taken = Philostatus[philID].right = 1;
                printf("Fork %d taken by philosopher %d\n",philID+1,philID+1);
             }else{
                printf("Philosopher %d is waiting for fork %d\n",philID+1,philID+1);
             }
          }else{ //except last philosopher case
             int dupphilID = philID;
             philID-=1;

             if(philID== -1)
                philID=(n-1);

             if(ForkAvil[philID].taken == 0){
                ForkAvil[philID].taken = Philostatus[dupphilID].right = 1;
                printf("Fork %d taken by Philosopher %d\n",philID+1,dupphilID+1);
             }else{
                printf("Philosopher %d is waiting for Fork %d\n",dupphilID+1,philID+1);
             }
          }
        }
        else if(Philostatus[philID].left==0){ //nothing taken yet
             if(philID==(n-1)){
                if(ForkAvil[philID-1].taken==0){
                   ForkAvil[philID-1].taken = Philostatus[philID].left = 1;
                   printf("Fork %d taken by philosopher %d\n",philID,philID+1);
                }else{
                   printf("Philosopher %d is waiting for fork %d\n",philID+1,philID);
                }
             }else{ //except last philosopher case
                if(ForkAvil[philID].taken == 0){
                   ForkAvil[philID].taken = Philostatus[philID].left  = 1;
                   printf("Fork %d taken by Philosopher %d\n",philID+1,philID+1);
                }else{
```

```
                    printf("Philosopher %d is waiting for Fork %d\n",philID+1,philID+1);
                }
            }
        }else{}
}

int main(){
for(i=0;i<n;i++)
    ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;

while(compltedPhilo<n){

for(i=0;i<n;i++)
        goForDinner(i);
printf("\nTill now num of philosophers completed dinner are %d\n\n",compltedPhilo);
}

return 0;
}
```

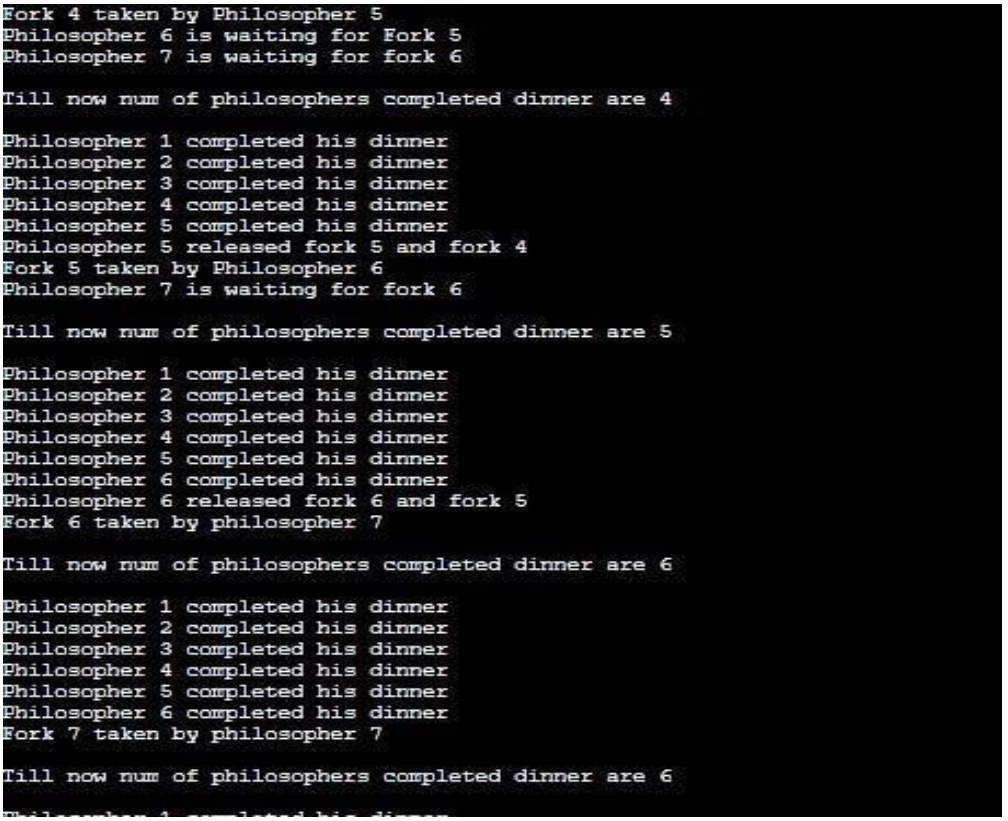


```
#include<stdio.h>
#define n 7
int compltedPhilo = 0,i;
struct fork{
int taken;
}ForkAvil[n];

struct philosp{
int left;
int right;
}Philostatus[n];

void goForDinner(int philID){
if(Philostatus[philID].left==10 && Philostatus[philID].right==10)
    printf("Philosopher %d completed his dinner\n",philID+1);
else if(Philostatus[philID].left==1 && Philostatus[philID].right==1){

        printf("Philosopher %d completed his dinner\n",philID+1);

        Philostatus[philID].left = Philostatus[philID].right = 10;
        int otherFork = philID-1;

        if(otherFork== -1)
            otherFork=(n-1);

        ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0;
        printf("Philosopher %d released fork %d and fork %d\n",philID+1,philID+1,otherFork+1);
        compltedPhilo++;
    }
    else if(Philostatus[philID].left==1 && Philostatus[philID].right==0){
            if(philID==(n-1)){
                if(ForkAvil[philID].taken==0){
                    ForkAvil[philID].taken = Philostatus[philID].right = 1;
                    printf("Fork %d taken by philosopher %d\n",philID+1,philID+1);
                }else{
                    printf("Philosopher %d is waiting for fork %d\n",philID+1,philID+1);
                }
            }else{
                int dupphilID = philID;
                philID-=1;

                if(philID== -1)
                    philID=(n-1);
```



```
                    if(ForkAvil[philID].taken == 0){
                        ForkAvil[philID].taken = Philostatus[dupphilID].right = 1;
                        printf("Fork %d taken by Philosopher %d\n",philID+1,dupphilID+1);
                    }else{
                        printf("Philosopher %d is waiting for Fork %d\n",dupphilID+1,philID+1);
                    }
                }
            }
            else if(Philostatus[philID].left==0){
                if(philID==(n-1)){
                    if(ForkAvil[philID-1].taken==0){
                        ForkAvil[philID-1].taken = Philostatus[philID].left = 1;
                        printf("Fork %d taken by philosopher %d\n",philID,philID+1);
                    }else{
                        printf("Philosopher %d is waiting for fork %d\n",philID+1,philID);
                    }
                }else{
                    if(ForkAvil[philID].taken == 0){
                        ForkAvil[philID].taken = Philostatus[philID].left = 1;
                        printf("Fork %d taken by Philosopher %d\n",philID+1,philID+1);
                    }else{
                        printf("Philosopher %d is waiting for Fork %d\n",philID+1,philID+1);
                    }
                }
        }else{}
}

int main(){
for(i=0;i<n;i++)
        ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;

while(compltedPhilo<n){

for(i=0;i<n;i++)
        goForDinner(i);
printf("\nTill now num of philosophers completed dinner are %d\n\n",compltedPhilo);
}

return 0;
}
```

OUTPUT:



RESULT:

The program has been executed successfully.

Ex No: 6 **Implementation of Banker's Algorithm for Deadlock Avoidance**

Date:

AIM:

To implement of safety algorithm for deadlock avoidance using C.

ALGORITHM:

1. Work and Finish be the vector of length m and n respectively,
   Work=Available and Finish[i] =False.
2. Find an i such that both Finish[i] =False Need<=Work If no such I exists go to step 4.
3. work= work + Allocation, Finish[i] =True;
4. if Finish[1]=True for all I, then the system is in safe state. Resource request algorithm

Let Request i be request vector for the process Pi, If request i=[j]=k, then process Pi wants k instances of resource type Rj.

1. if Request<=Need I go to step 2. Otherwise raise an error condition.
2. if Request<=Available go to step 3. Otherwise Pi must since the resources are available.
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows;

Available=Available-Request I;
Allocation I =Allocation +Request I;
Need i=Need i-Request I;

If the resulting resource allocation state is safe, the transaction is completed and process Pi is allocated its resources. However if the state is unsafe, the Pi must wait for Request i and the old resource-allocation state is restored.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int max[100][100];
```

```
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void input();
void show();
void cal();
int main()
{
int i,j;
printf("********** Banker's 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 resources instances\t");
scanf("%d",&r);
printf("Enter the Max Matrix\n");
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
scanf("%d",&max[i][j]);
}
}
printf("Enter the Allocation Matrix\n");
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
scanf("%d",&alloc[i][j]);
}

}
printf("Enter the available Resources\n");
for(j=0;j<r;j++)
{
```

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

{
need[i][j]=max[i][j]-alloc[i][j];
}
```

```
}
printf("\n");
while(flag)
{
flag=0;
for(i=0;i<n;i++)
{
int c=0;
for(j=0;j<r;j++)
{
if((finish[i]==0)&&(need[i][j]<=avail[j]))
{
c++;
if(c==r)
{
for(k=0;k<r;k++)
{
avail[k]+=alloc[i][j];
finish[i]=1;
flag=1;
}
printf("P%d->",i);
if(finish[i]==1)
{
i=n;
}
}
}
}
}
}
for(i=0;i<n;i++)
{
if(finish[i]==1)
{
c1++;
}
else
{
printf("P%d->",i);
}

}
if(c1==n)
```

```
{
printf("\n The system is in safe state");
}
else
{
printf("\n Process are in dead lock");
printf("\n System is in unsafe state");
}
}
```

SAMPLE INPUT:

System consists of 5 processes: P_0 P_1 P_2 P_3 P_4

resource types:

R1 - 10 instances

R2 - 5 instances

R3 - 7 instances

Process	Maximum			Allocation		
	R1	R2	R3	R1	R2	R3
P0	7	5	3	0	1	0
P1	3	2	2	2	0	0
P2	9	0	2	3	0	2
P3	2	2	2	2	1	1
P4	4	3	3	0	0	2

OUTPUT:

```
********** Banker's Algo ************
Enter the no of Processes       5
Enter the no of resources instances     3
Enter the Max Matrix
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the Allocation Matrix
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the available Resources
3 2 2
Process  Allocation      Max     Available
P1       0 1 0  7 5 3   3 2 2
P2       2 0 0  3 2 2
P3       3 0 2  9 0 2
P4       2 1 1  2 2 2
P5       0 0 2  4 3 3
P1->P3->P4->P2->P0->
 The system is in safe state

...Program finished with exit code 0
Press ENTER to exit console.
```

RESULT:

Thus to implement of safety algorithm for deadlock avoidance using C has been executed successfully

EXP: 7 IMPLEMENTATION OF MEMORY ALLOCATIONAND MANAGEMENT TECHNIQUES

DATE:

AIM:

To implement the memory management concept using C.

ALGORITHM

- Start
- Read the number of processes and number of the block from the user
- Read the size of each block and the size of all the process requests.
- Start allocating the processes
- Display the results
- Stop

PROGRAM

```
#include<stdio.h>void
main()
{
        int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;
        for(i = 0; i < 10; i++)
        {
                flags[i] = 0;
                allocation[i] = –1;
        }
        printf("Enter no. of blocks: ");scanf("%d",
        &bno);
        printf("\nEnter size of each block: ");for(i =
        0; i < bno; i++)
                scanf("%d", &bsize[i]);
        printf("\nEnter no. of processes: ");
        scanf("%d", &pno);
        printf("\nEnter size of each process: ");for(i
        = 0; i < pno; i++)
                scanf("%d", &psize[i]);
        for(i = 0; i < pno; i++)
                for(j = 0; j < bno; j++)
```

```
                if(flags[j] == 0 && bsize[j] >= psize[i])
                {
                        allocation[j] = i;
                        flags[j] = 1;
                        break;
                }

        printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");for(i
        = 0; i < bno; i++)
        {
                printf("\n%d\t\t%d\t\t", i+1, bsize[i]);
                if(flags[i] == 1)
                        printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);
                 else
                        printf("Not allocated");
        }
```

```c
#include<stdio.h>
void main()
{
    int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;
    for(i = 0; i < 10; i++)
    {
        flags[i] = 0;
        allocation[i] = -1;
    }
    printf("Enter no. of blocks: ");
    scanf("%d", &bno);
    printf("\nEnter size of each block: ");
    for(i = 0; i < bno; i++)
        scanf("%d", &bsize[i]);
    printf("\nEnter no. of processes: ");
    scanf("%d", &pno);
    printf("\nEnter size of each process: ");
    for(i = 0; i < pno; i++)
        scanf("%d", &psize[i]);
    for(i = 0; i < pno; i++)
        for(j = 0; j < bno; j++)
            if(flags[j] == 0 && bsize[j] >= psize[i])
            {
                allocation[j] = i;
                flags[j] = 1;
                break;
            }

    printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");
    for(i = 0; i < bno; i++)
    {
        printf("\n%d\t\t%d\t\t", i+1, bsize[i]);
        if(flags[i] == 1)
            printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);
        else
            printf("Not allocated");
    }
}
}
```

SAMPLE INPUT

Number of process: P0 P1 P2 P3

Number of blocks: 5 memory partitions

Processes size: 212 Kb, 417 Kb, 112 Kb, and 426 Kb (in order)Block

size: 100Kb, 500Kb, 200Kb, 300Kb, 600Kb (in order)

OUTPUT

```
Enter no. of blocks: 5

Enter size of each block: 100 500 200 300 600

Enter no. of processes: 4

Enter size of each process: 212 417 112 476

Block no.       size            process no.             size
1               100             Not allocated
2               500             1                       212
3               200             3                       112
4               300             Not allocated
5               600             2                       417

...Program finished with exit code 0
Press ENTER to exit console.
```

RESULT:

Thus to implement the memory management concept using C has been executed successfully.

EX.NO: 8 **IMPLEMENTATION OF PAGE REPLACEMENT TECHNIQUES**

DATE:

AIM

To write a c program to implement page replacement algorithms

ALGORITHM:

STEP-1: Enter 1 for entering data and provide the length of the string, referencestring and frame size as the input.

STEP-2: Enter 2 to perform FIFO page replacement algorithm.FIFO PAGE

REPLACEMENT ALGORITHM:

STEP-2(i): Start traversing the pages.

STEP-2(ii): If set holds less pages than capacity.

a) Insert page into the set one by one until
the size of set reaches capacity or allpage
requests are processed.

b) Simultaneously maintain the pages in thequeue to
perform FIFO.

c) Increment page fault

STEP-2(iii): Else

If current page is present in set, do nothing.

Else

a) Remove the first page from the queueas it was
the first to be entered in
the memory

b) Replace the first page in the queue with the current page in the string.

c) Store current page in the queue.

d) Increment page faults.

STEP-2(iv): Return page faults.

STEP-3: Enter 3 to perform Optimal page replacement algorithm.OPTIMAL PAGE

REPLACEMENT ALGORITHM:

STEP-3(i): Start traversing the pages.

STEP-3(ii): If set holds less pages than capacity.a)Insert page
into the set one by one until
the size of set reaches capacity or all
pagerequests are processed.

b) Simultaneously maintain the pages in the queue to perform Optimal.

c) Increment page fault

STEP-3(iii): Else

If current page is present in set, do nothing.

Else

a) the page that is never referenced in the future , if that page is not present then the page that is referenced farthest in the future is removed from the queue

b) Replace this page in the queue withthe current page in the string.

c) Store current page in the queue.

d) Increment page faults.

STEP-3(iv): Return page faults.

STEP-4: Enter 4 to perform LRU page replacement algorithm.

LRU PAGE REPLACEMENT ALGORITHM:

STEP-4(i): Start traversing the pages.

STEP-4(ii): If set holds less pages than capacity.a)Insert page into the set one by one until the size of set reaches capacity or allpage requests are processed.

b) Simultaneously maintain the pages in thequeue to perform Optimal.

c) Increment page fault

STEP-4(iii): Else

If current page is present in set, do nothing.

Else

a) the page that is Least Recently Used is removed from the queue

b) Replace this page in the queue withthe current page in the string.

c) Store current page in the queue.

d) Increment page faults.

e)

STEP-4(iv): Return page faults. STEP-5: Enter 5 to exit the program.

PROGRAM:

```
#include<stdio.h>i

ntn,nf;

int in[100];

 int p[50];inthit=0; inti,j,k;
int pgfaultcnt=0;

void getData()

{
    printf("\nEnter length of page reference

    sequence:");scanf("%d",&n);

    printf("\nEnter the page reference sequence:");for(i=0;

    i<n;i++)

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

    printf("\nEnterno of

    frames:");scanf("%d",&nf);

}
void initialize()

{
    pgfaultcnt=0;

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

        p[i]=9999;
}
int isHit(int data)

{
    hit=0;

    for(j=0; j<nf; j++)

    {
        if(p[j]==data)

        {
            hit=1;

            break;

        }
```

```
    }
    return hit;
}
int getHitIndex(int data)
{
    int hitind;
    for(k=0; k<nf; k++)
    {
        if(p[k]==data)
        {
            hitind=k;break;
        }
    }
    return hitind;
}
void dispPages()
{
    for (k=0; k<nf; k++)
    {
        if(p[k]!=9999)
            printf(" %d",p[k]);
    }
}
void dispPgFaultCnt()
{
    printf("\nTotal no of page faults:%d",pgfaultcnt);
}
void fifo()
```

```
{
    initialize(); for(i=0;
    i<n; i++)
    {
        printf("\nFor %d :",in[i]);
        if(isHit(in[i])==0)
        {
            for(k=0; k<nf-1; k++)
                p[k]=p[k+1];
            p[k]=in[i];
            pgfaultcnt++;
            dispPages();
        }
        else
            printf("No page fault");
    }
    dispPgFaultCnt();
}
void optimal()
{
    initialize(); int
    near[50];
    for(i=0; i<n; i++)
    {
        printf("\nFor %d :",in[i]);

        if(isHit(in[i])==0)
        {
            for(j=0; j<nf; j++)
            {
```

```
            int pg=p[j]; int
            found=0;
            for(k=i; k<n; k++)
            {
                if(pg==in[k])
                {
                    near[j]=k;
                    found=1;
                    break;
}
                else
                    found=0;
            }
            if(!found)
                near[j]=9999;
        }
        int max=-9999; int
        repindex; for(j=0;
        j<nf; j++)
        {
            if(near[j]>max)
            {
                max=near[j];
                repindex=j;
            }
        }
        p[repindex]=in[i];
        pgfaultcnt++;
```

```
            dispPages();
        }
        else
            printf("No page fault");
    }
    dispPgFaultCnt();
}
void lru()
{
    initialize();
    int least[50]; for(i=0;
    i<n; i++)
    {
        printf("\nFor %d :",in[i]);
        if(isHit(in[i])==0)
        {
            for(j=0; j<nf; j++)
            {
                int pg=p[j]; int
                found=0;
                for(k=i-1; k>=0; k--)
            {
                    if(pg==in[k])
                    {
                        least[j]=k;
                        found=1;
                        break;
                    }
```

```
                else
                    found=0;
                }
                if(!found) least[j]=-
                    9999;
            }
            int min=9999;int
            repindex;
            for(j=0; j<nf; j++)
            {
                if(least[j]<min)
                {
                    min=least[j]; repindex=j;
                }
            }
            p[repindex]=in[i];
            pgfaultcnt++;
            dispPages();
        }
        else
            printf("No page fault!");
    }
    dispPgFaultCnt();
}
int main()
{
    int choice;
    while(1)
```

```
        printf("\nPage Replacement Algorithms\n1.Enter
data\n2.FIFO\n3.Optimal\n4.LRU\n5.Exit\nEnter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
            getData();
            break;
        case 2:
            fifo();
            break;
        case 3:
            optimal();
            break;
        case 4:
            lru();
            break;
        default:
            return 0;
            break;
        }
    }
}
```

SAMPLE INPUT:

Length of page reference string: 12
Reference String: 0 1 2 3 0 1 4 0 1 2 3
4Frame size = 3

FIFO – no of page faults: 9
OPTIMAL –no of page faults: 7LRU
– no of page faults: 10

OUTPUT:

```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:1

Enter length of page reference sequence:12

Enter the page reference sequence:0 1 2 3 0 1 4 0 1 2 3 4

Enter no of frames:3

Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:2

For 0 : 0
For 1 : 0 1
For 2 : 0 1 2
For 3 : 1 2 3
For 0 : 2 3 0
For 1 : 3 0 1
For 4 : 0 1 4
```

```
For 0 :No page fault
For 1 :No page fault
For 2 : 1 4 2
For 3 : 4 2 3
For 4 :No page fault
Total no of page faults:9
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:3

For 0 : 0
For 1 : 0 1
For 2 : 0 1 2
For 3 : 0 1 3
For 0 :No page fault
For 1 :No page fault
For 4 : 0 1 4
For 0 :No page fault
For 1 :No page fault
For 2 : 2 1 4
For 3 : 3 1 4
For 4 :No page fault
Total no of page faults:7
Page Replacement Algorithms
1.Enter data
2.FIFO
```

```
3.Optimal
4.LRU
5.Exit
Enter your choice:4

For 0 : 0
For 1 : 0 1
For 2 : 0 1 2
For 3 : 3 1 2
For 0 : 3 0 2
For 1 : 3 0 1
For 4 : 4 0 1
For 0 :No page fault!
For 1 :No page fault!
For 2 : 2 0 1
For 3 : 2 3 1
For 4 : 2 3 4
Total no of page faults:10
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:5


...Program finished with exit code 0
Press ENTER to exit console.
```

RESULT:

Thus the page replacement algorithms (FIFO,Optimal,LRU) are successfully implemented

EX NO: 9 IMPLEMENTATION OF FILE ORGANIZATION TECHNIQUES

DATE:

a. Single level directory

AIM:

To write a C program to implement File Organization concept using the technique Singleleveldirectory.

ALGORITHM:

Step-1: Start the program.
Step-2: Declare the count, file name, graphical interface.
Step-3: Read the number of files
Step-4: Read the file name
Step-5: Declare the root directory
Step-6: Using the file eclipse function define the files in a single level
Step-7: Display the files
Step-8: Stop the program

PROGRAM:

```
#include<stdlib.
h>
#include<string.
h>
#include<stdio.h
> struct
{

char
dname[10],fname[10][10];int
fcnt;
}dir;

void main()

{
int i,ch;
charf[30];
dir.fcnt = 0;
```

```
printf("\nEnter name of directory :
");scanf("%s", dir.dname);
while(1)

{

printf("\n\n1. Create File\t2. Delete File\t3. Search File \n 4. Display Files\t5. Exit\nEnter
yourchoice :");
scanf("%d",&ch
);switch(ch)
{

case 1: printf("\nEnter the name of the file : ");
scanf("%s",dir.fname[dir.fcnt]);
dir.fcnt+
+;break;
case 2: printf("\nEnter the name of the file : ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)

{

if(strcmp(f, dir.fname[i])==0)

{

printf(" %s is deleted ",f);
strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);
break; } }
if(i==dir.fcnt) printf(" %s not
found",f);else
dir.fcnt--
;break;
case 3:
printf("\nEnter the name of the file to search : ");
scanf("%s",f);
```

```
for(i=0;i<dir.fcnt;i++)
{

if(strcmp(f, dir.fname[i])==0)

{

printf(" %s is found ",
f);break;
}

}

if(i==dir.fcnt)

printf(" %s not
found",f);break;
case 4:
if(dir.fcnt==
0)
printf("\nDirectory
Empty");else
{

printf("\nThe Files are :
");
for(i=0;i<dir.fcnt;i++)
printf("%s\n",dir.fname[i]);
}

break;

default: exit(0);

}

}

}
```

RESULT:

Thus the program has been executed successfully.

B. Implementation Of Two Level Directory In File Organization Techniques

AIM:

To write a C program to implement File Organization concept using the technique TwoLevel Directory.

ALGORITHM:

Step-1: Start the program.

Step-2: Declare the count, file name, graphical

interface.Step-3: Read the number of files

Step-4: Read the file name
Step-5: Declare the root directory

Step-6: Using the file eclipse function define the files in a

single levelStep-7: Display the files

Step-8: Stop the program

PROGRAM:

```
#include<string.
h>
#include<stdlib.
h>
#include<stdio.h
> struct
{
char
dname[10],fname[10][10];int
fcnt;
}dir[10];
void main()
{
int i,ch,dcnt,k;
char f[30],
d[30];dcnt=0;
while(1)
```

```
{
printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
printf("\n4.Search File\t\t5. Display\t6. Exit\nEnter your choice :
");scanf("%d",&ch);
```

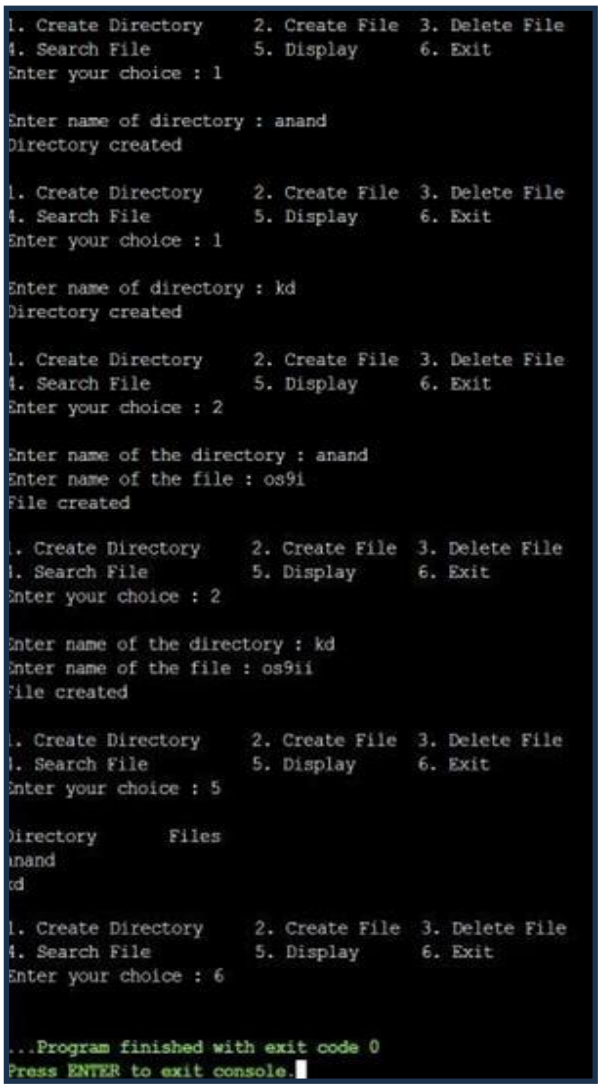
```
switch(ch)
{
case 1: printf("\nEnter name of directory : ");
scanf("%s",
dir[dcnt].dname);
dir[dcnt].fcnt=0;
dcnt++;
printf("Directory
created");break;
case 2: printf("\nEnter name of the directory
: ");scanf("%s",d);
for(i=0;i<dcnt;i++)
if(strcmp(d,dir[i].dname)=
=0)
{
printf("Enter name of the file :
");
scanf("%s",dir[i].fname[dir[i].fcn
t]);printf("File created");
break;
}
if(i==dcnt)
printf("Directory %s not
found",d);break;
case 3: printf("\nEnter name of the directory
: ");scanf("%s",d);
for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file :
```

```
");scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
```

```
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is deleted
",f);dir[i].fcnt--;
strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
gotojmp;
}
}
printf("File %s not
found",f);goto jmp;
}
}
printf("Directory %s not
found",d);jmp : break;
case 4: printf("\nEnter name of the directory
: ");scanf("%s",d);
for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter the name of the file :
");scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is found
",f);goto jmp1;
}
}
```

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

OUTPUT:



RESULT:

Thus the program has been executed successfully.

EX NO: 10 **IMPLEMENTATION OF DISK SCHEDULING ALGORITHM.**

DATE:

AIM:

To write a C program to implement disk scheduling algorithm.

ALGORITHM:

FCFS:

Step 1: Input the processes along with their burst time (bt).
Step 2: Find waiting time (wt) for all processes.
Step 3: As first process that comes need not to wait so waiting time for process 1 will be 0 i.e.wt[0] = 0.
Step 4: Find waiting time for all other processes i.e.
forprocess i -> wt[i] = bt[i-1] + wt[i-1] .
Step 5: Find turnaround time = waiting_time + burst_time for all processes.
Step 6: Find average waiting time = total_waiting_time / no_of_processes.
Step 7: Similarly, find average turnaround time = total_turn_around_time / no_of_processes.

SSTF:

Step 1: Let Request array represents an array storing indexes of tracks that have beenrequested. „head‟ is the position of disk head.
Step 2: Find the positive distance of all tracks in the request array from head.
Step 3: Find a track from requested array which has not been accessed/serviced yet and hasminimumdistance from head.
Step 4: Increment the total seek count with this distance.
Step 5: Currently serviced track position now becomes the new head position.
Step 6: Go to step 2 until all tracks in request array have not been serviced.

SCAN:

Step 1: Let Request array represents an array storing indexes of tracks that have been requested inascending order of their time of arrival. „head‟ is the position of disk head.**Step 2:** Let direction represents whether the head is moving towards left or right.
Step 3: In the direction in which head is moving service all tracks one by one.
Step 4: Calculate the absolute distance of the track from the head.
Step 5: Increment the total seek count with this distance.
Step 6: Currently serviced track position now becomes the new head position.
Step 7: Go to step 3 until we reach at one of the ends of the disk.
Step 8: If we reach at the end of the disk reverse the direction and go to step 2 until all tracksinrequest array have not been serviced.

PROGRAM:

FCFS:

```
#include<stdio.h>
void findWaitingTime(int processes[], int n, int bt[], int wt[])
{
        wt[0] = 0;
        for (int i = 1; i < n ; i++ )
                wt[i] = bt[i-1] + wt[i-1] ;
}

void findTurnAroundTime( int processes[], int n, int bt[], int wt[], int tat[])
{
        for (int i = 0; i < n ; i++)
                tat[i] = bt[i] + wt[i];
}

void findavgTime( int processes[], int n, int bt[])
{
        int wt[n], tat[n], total_wt = 0, total_tat = 0;
        findWaitingTime(processes, n, bt, wt);
        findTurnAroundTime(processes, n, bt, wt, tat);
        printf("ProcessesBurst time Waiting time Turn around
        time\n");

        for (int i=0; i<n; i++)
        {
                total_wt = total_wt + wt[i];
                total_tat = total_tat + tat[i];
                printf(" %d ",(i+1));
                printf(" %d ", bt[i] );
                printf(" %d",wt[i] );
                printf(" %d\n",tat[i] );
        }
        int s=(float)total_wt / (float)n;int
        t=(float)total_tat / (float)n;
        printf("Average waiting time =
        %d",s);printf("\n");
        printf("Average turn around time = %d ",t);
}
```

```
int main()
{
        int processes[] = { 1, 2, 3};
        int n = sizeof processes / sizeof

        processes[0];intburst_time[] = {10, 5, 8};

        findavgTime(processes, n,
        burst_time);return 0;
}
```

SSTF
:

```
#include<stdio.h
>
#include<stdlib.
h>int main()
{
   int
   RQ[100],i,n,TotalHeadMoment=0,initial,count=0;
   printf("Enter the number of Requests : ");
   scanf("%d",&n);
   printf("Enter the Requests
   sequence...\n");for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position :
   ");scanf("%d",&initial);

   while(count!=n)
   {
      int
      min=1000,d,index;
      for(i=0;i<n;i++)
      {
        d=abs(RQ[i]-
        initial);if(min>d)
        {
           min=d;
           index=i
           ;
        }

      }
      TotalHeadMoment=TotalHeadMoment+
```

```
min;initial=RQ[index];
```

```
    RQ[index]=100
    0;count++;
  }

  printf("Total head movement is %d",TotalHeadMoment);
  return 0;
}
```

SCAN:

```
#include<stdio.h>
int
absoluteValue(int);
void main()
{
  int queue[25],n,headposition,i,j,k,seek=0, maxrange,
  difference,temp,queue1[20],queue2[20],temp1=0,temp2=0
  ;floataverageSeekTime;
  printf("Enter the maximum range of
  Disk: ");scanf("%d",&maxrange);

  printf("Enter the number of queue
  requests: ");scanf("%d",&n);

  printf("Enter the initial head position: ");
  scanf("%d",&headposition);
  printf("Enter the disk positions to be read(queue):
  ");for(i=1;i<=n;i++)
  {
    scanf("%d",&temp)
    ;
    if(temp>headpositio
    n)
    {
      queue1[temp1]=temp;
      temp1++;
    }
    else
    {
      queue2[temp2]=temp;
      temp2++;
    }
  }
  for(i=0;i<temp1-1;i++)
```

```
{
   for(j=i+1;j<temp1;j++)
   {
      if(queue1[i]>queue1[j])
      {
         temp=queue1[i];
         queue1[i]=queue1[
         j];queue1[j]=temp;
      }
   }
}
for(i=0;i<temp2-1;i++)
{
   for(j=i+1;j<temp2;j++)
   {
      if(queue2[i]<queue2[j])
      {
         temp=queue2[i];
         queue2[i]=queue2[
         j];queue2[j]=temp;
      }
   }
}
for(i=1,j=0;j<temp1;i++,j++)
{
   queue[i]=queue1[j];
}
queue[i]=maxrange;

for(i=temp1+2,j=0;j<temp2;i++,j++)
{
   queue[i]=queue2[j];
}
queue[i]=0;
queue[0]=headposition;
for(j=0; j<=n; j++)
{
   difference = absoluteValue(queue[j+1]-queue[j]);seek

   = seek + difference;
```

```
        printf("Disk head moves from position %d to %d with Seek %d
        \n",queue[j],queue[j+1], difference);
    }
    averageSeekTime = seek/(float)n;
    printf("Total Seek Time= %d\n",
    seek);
    printf("Average Seek Time= %f\n", averageSeekTime);
}
int absoluteValue(int x)
{
    if(x>0)
        return
    x;else
        return x*-1;
}
```

SAMPLE INTPUT:

Enter the No of Cylinder:

200Enter the No of

Requests: 8

Requests: 98 183 37 122 14 124 65 67

Starting Head position: 53

1. FCFS
2. SSTF
3. SCAN

FCFS:

Total Seek time: 640

SSTF:

Total Seek time: 236

SCAN:

Total Seek time: 236

OUTPUT:

FCFS:

```
Processes Burst time Waiting time Turn around time
 1        10         0            10
 2        5          10           15
 3        8          15           23
Average waiting time = 8
Average turn around time = 16

...Program finished with exit code 0
Press ENTER to exit console.
```

SSTF:

```
Enter the number of Requests : 8
Enter the Requests sequence...
98
183
37
122
14
124

65
67
Enter initial head position : 53
Total head movement is 236

...Program finished with exit code 0
Press ENTER to exit console.
```

SCAN:

```
Enter the maximum range of Disk: 183
Enter the number of queue requests: 8
Enter the initial head position: 53
Enter the disk positions to be read(queue): 98
183
37
122
14
124
65
67
Disk head moves from position 53 to 65 with Seek 12
Disk head moves from position 65 to 67 with Seek 2
Disk head moves from position 67 to 98 with Seek 31
Disk head moves from position 98 to 122 with Seek 24
Disk head moves from position 122 to 124 with Seek 2
Disk head moves from position 124 to 183 with Seek 59
Disk head moves from position 183 to 183 with Seek 0
Disk head moves from position 183 to 37 with Seek 146
Disk head moves from position 37 to 14 with Seek 23
Total Seek Time= 299
Average Seek Time= 37.375000

...Program finished with exit code 0
Press ENTER to exit console.
```

RESULT:

Disk scheduling algorithm is implemented using c program.