SRI MANAKULA VINAYAGAR ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi & Affiliated to Pondicherry University)
(Accredited by NBA-AICTE, New Delhi, ISO 9001:2000 Certified Institution &
Accredited by NAAC with "A" Grade)
Madagadipet, Puducherry - 605 107

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(JAN 2022-MAY 2022)

U20CSP303/LINUX INTERNALS LABORATORY

III SEMESTER

SRI MANAKULA VINAYAGAR ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi & Affiliated to Pondicherry University)
(Accredited by NBA-AICTE, New Delhi, ISO 9001:2000 Certified Institution &
Accredited by NAAC with "A" Grade)
Madagadipet, Puducherry - 605 107

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SUBJECT: LINUX INTERNALS LABORATORY **SUBJECT CODE: U20CSP303**

LIST OF EXPERIMENTS

1. Study of basic UNIX/Linux commands
2. Shell Programming - I
   (a) To Write a Shell program to count the number of words in a file.
   (b) To Write a Shell program to calculate the factorial of a given number.
   (c) To write a Shell program to generate Fibonacci series.
   (d) Write a Shell Program to wish the user based on the login time.
3. Shell Programming - II
   (a) Loops
   (b) Patterns
   (c) Expansions
   (d) Substitutions
4. Programs using the following system calls of UNIX/Linux operating system: fork, exec, getpid, exit, wait,
   close, stat, opendir, readdir.
5. To write a program to simulate cat commands.
6. To write a program to simulate head and tail commands.
7. Simulate UNIX commands like ls, grep.
8. Process Scheduling- FCFS, SJF, Priority and Round robin.
9. Implementation of Banker's algorithm.
10. Write a C program to simulate producer and consumer problem using semaphores

LIST OF EXPERIMENTS

S.NO	TITLE OF EXPERIMENTS	Page No	Marks	Staff Signature
1	Study of basic UNIX/Linux commands			
	SHELL PROGRAMMING - I			
2a	Counting of a Words			
2b	Count occurrence of a words in a file			
2c	Factorial of a Number			
2d	Fibonacci Series			
2e	Greeting Message based on time			
2f	String Comparison			
	SHELL PROGRAMMING - II			
3a	Loops			
3b	Patterns			
3c	Expansions			
3d	Substitutions			
	SYSTEM CALLS			
4a	Create a Child Process using Fork			
4b	Exec ( ) System Call			
4c	Getpid ( ) System Call			
4d	read( ) and write( ) system call			
4e	Wait( ) And Exit( ) System Call			
4f	Stat( ) System Call			
4g	Directory System Call			
4h	File System Call Using Open And Close			
4i	system call to find mode of the file			

S.NO	TITLE OF EXPERIMENTS	Page No	Marks	Staff Signature
	SIMULATION OF LINUX COMMANDS			
5a	Simulation of CAT command			
5b	Simulation of Head and Tail Command			
5c	Simulation of GREP Command			
5d	Simulation of FGREP Command			
5e	Simulation of ls Command			
	PROCESS SCHEDULING			
6a	Implementation of FCFS Scheduling			
6b	Implementation of SJF Scheduling			
6c	Implementation of Priority Scheduling			
6d	Implementation of Round Robin Scheduling			
6e	Implementation of Shortest Remaining Time first scheduling algorithm			
	DEADLOCK HANDLING			
7a	Banker's algorithm for Deadlock Avoidance			
	SEMAPHORE			
8a	Producer and Consumer Problem			
8b	Dining Philosopher Problem			

EX.NO: 01	STUDY OF BASIC LINUX COMMANDS
DATE:	

AIM:

To study the basic commands in Linux.

COMMANDS:

1. Calendar

NAME : calendar
(i) SYNTAX : cal
DESCRIPTION : Displays a simple calendar. If arguments are not Specified, the current month is displayed.
EXAMPLE : cal
OUTPUT :

June 2014

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					

(ii) SYNTAX : cal year
DESCRIPTION : Displays calendar of that year
EXAMPLE : cal 2012
OUTPUT :

January

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				

February

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			

March

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

April

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

May

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

June

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

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

July

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				

August

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	

September

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						

October

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			

November

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	

December

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					

(iii) SYNTAX : cal -3
DESCRIPTION : Displays calendar of previous, current, next months of current year
OUTPUT :

July

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				

August

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	

September

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						

(iv)SYNTAX : cal month year
DESCRIPTION : Displays the calendar for corresponding month and year .
EXAMPLE :
OUTPUT :

2. Date

NAME : DATE- print or set the system date and time
(i) SYNTAX : date
DESCRIPTION : Display the current time in the given format or set the system date.
OUTPUT :

(ii) SYNTAX : date +% H
DESCRIPTION : Display the current hour.
OUTPUT :

(iii) SYNTAX : date +% h
DESCRIPTION : Display the current month name.
OUTPUT :

(iv) SYNTAX : date +% m
DESCRIPTION : Display the current month number.
OUTPUT :

(v) SYNTAX : date +% a
DESCRIPTION : Display the abbreviated weekday name.
OUTPUT :

(vi) SYNTAX : date +% y
DESCRIPTION : Display the current year.
OUTPUT :

(vii) SYNTAX : date +% S
DESCRIPTION : Display the current second.
OUTPUT :

3. Script

NAME : SCRIPT – makes typescript of terminal session
DESCRIPTION : Makes a typescript of everything printed on your terminal. It is useful for students who need a hardcopy record of an interactive session as proof of an assignment, as the typescript file can be printed out later with lpr(1).
SYNTAX : script scriptname
……..
……..
……..
Exit

OPENING A SCRIPT:

SYNTAX : vi scriptname
EXAMPLE : vi date.txt
OUTPUT : ~date
~ Mon Jul 23 12:17:50 IST 2012
~
~
INSERT

4. ls

NAME : LIST – list directory contents
(i) SYNTAX : ls
DESCRIPTION : List information about the Files (the current directory by default).
OUTPUT :

greatest.sh	cse.txt	mouse.txt	digit.sh
emp.sh	num.sh	case.sh	

(ii) SYNTAX : ls –l
DESCRIPTION : Displays files in long listing format.
OUTPUT :

(iii)SYNTAX : ls -r
DESCRIPTION : Displays the files in reverse sorted order.
OUTPUT :

(iv) SYNTAX : ls -s
DESCRIPTION : Displays the size of each files.
OUTPUT :

(v) SYNTAX : ls -S
DESCRIPTION : Displays the files in sorted order.
OUTPUT :

5. cp

NAME : cp – copy files and directories
SYNTAX : cp fi f2
DESCRIPTION :

8. mkdir

NAME : mkdir – makes directory
SYNTAX : mkdir DirectoryName
DESCRIPTION : Creates the directory, if they do not exist already
EXAMPLE :

9. rmdir

NAME : rmdir – removes directory
SYNTAX : rmdir DirectoryName
DESCRIPTION : Removes the directory, only it is empty.
EXAMPLE :

10. Pwd

NAME : pwd – Present Working Directory displays the name of the current/working directory
SYNTAX : pwd
DESCRIPTION : Displays the name of the current/working directory
OUTPUT :

11. Cd

NAME : cd – Change Directory
(i) SYNTAX : cd dirname
DESCRIPTION : Change the directory which we use to work with.

EXAMPLE :

(ii) SYNTAX : cd ..
DESCRIPTION : Quits from the current directory.

(iii)SYNTAX : cd\
DESCRIPTION : Returns to the home directory.

12. Cat

NAME : cat- concatenate & open files and print on the standard output
(i) SYNTAX : cat > filename
DESCRIPTION : This command is used to open a new file.
EXAMPLE : cat > a.txt

OUTPUT :

NAME : ZZZZ
ROLL NO: XX

(ii) SYNTAX : cat filename
DESCRIPTION : To view the contents of the file.
EXAMPLE : cat a.txt
OUTPUT :

NAME : ZZZZ
ROLL NO: XX

(iii) SYNTAX : cat f1 f2 > f3
DESCRIPTION : To concatenate f1 and f2 save in f3
EXAMPLE : cat a.txt b.txt > c.txt
OUTPUT :

a.txt=>
NAME : ZZZZ
ROLL NO: XX

b.txt=>
COLLEGE.SMVEC

c.txt=>
NAME : ZZZZ
ROLL NO: XX
COLLEGE.SMVEC

(iv) SYNTAX : cat –n filename
DESCRIPTION : To display the contents of the file along with the line numbers.
EXAMPLE : cat -n sample.txt
OUTPUT :

(v) SYNTAX : cat f1 >> f2
DESCRIPTION : To redirect the data from one file to another.
EXAMPLE : cat sample.txt new.txt

OUTPUT : cat new.txt

13. Whoami

NAME : Displays the current user login and identity.
SYNTAX : whoami
OUTPUT :

14. Man

NAME : man – help command
SYNTAX : man command
DESCRIPTION : Displays the description of a command
EXAMPLE : man rm

15. Head

NAME : head
SYNTAX : head filename
DESCRIPTION : Displays the first ten lines in the file.
EXAMPLE : head fruits.txt
OUTPUT : apple
banana
cherry
jack fruit
strawberry
orange
pineapple
mango
grape
papaya

16. Tail

NAME : tail
SYNTAX : tail filename
DESCRIPTION : Displays the last ten lines in the file.
EXAMPLE : tail fruits.txt
OUTPUT : apple
banana
cherry
jack fruit
strawberry
orange
pineapple
mango
grape
papaya

17. Clear

NAME : clear
SYNTAX : clear
DESCRIPTION : Clears the content of the command prompt.

18. Sort

NAME : sort
(i) SYNTAX : sort filename
DESCRIPTION : Sorts the content of the file in ascending order.
EXAMPLE : sort names.txt
OUTPUT : Arun
Balu
Chandra
David
Thinesh

(ii) SYNTAX : sort –r filename
DESCRIPTION : Sorts the content of the file in descending order.
EXAMPLE : sort –r names.txt
OUTPUT : Thinesh
David
Chandra
Balu
Arun

19. Who

NAME : who
SYNTAX : who
DESCRIPTION : Displays all the users currently logged it.
OUTPUT :

```
csea13   pts/1       Jul 23 10:08 (172.17.22.38)
csea06   pts/10      Jul 23 10:13 (172.17.21.11)
csea12   pts/12      Jul 23 10:13 (172.17.21.35)
csea18   pts/11      Jul 23 10:13 (172.17.20.9)
csea24   pts/8       Jul 23 10:14 (172.17.22.33)
csea05   pts/13      Jul 23 10:15 (172.17.21.36)
root     :0          Jul 23 10:20
csea20   pts/16      Jul 23 10:22 (172.17.21.43)
csea11   pts/20      Jul 23 10:28 (172.17.21.27)
csea23   pts/5       Jul 23 10:37 (172.17.22.14)
csea03   pts/0       Jul 23 10:37 (172.17.222.37)
csea01   pts/4       Jul 23 11:00 (172.17.22.13)
csea21   pts/15      Jul 23 11:00 (172.17.21.34)
csea07   pts/18      Jul 23 11:00 (172.17.20.28)
csea08   pts/14      Jul 23 11:01 (172.17.22.16)
csea16   pts/7       Jul 23 11:01 (172.17.21.24)
csea02   pts/19      Jul 23 11:01 (172.17.21.45)
```

staff pts/26 Jul 23 12:20 (172.17.21.21)

20. Finger

NAME : finger
SYNTAX : finger
DESCRIPTION : Displays the detailed information about the system users.
OUTPUT :

```
Login    Name      Tty     Idle  Login Time   Office     Office Phone
csea01           pts/4        2   Jul 23 11:00 (172.17.22.13)
csea02   csea02   pts/19       2  Jul 23 11:01 (172.17.21.45)
csea03           pts/0            Jul 23 10:37 (172.17.222.37)
csea05           pts/13           Jul 23 10:15 (172.17.21.36)
csea06           pts/10           Jul 23 10:13 (172.17.21.11)
csea07           pts/18           Jul 23 11:00 (172.17.20.28)
csea08           pts/14           Jul 23 11:01 (172.17.22.16)
csea09           pts/3            Jul 23 12:13 (172.17.22.115)
csea10           pts/2        1   Jul 23 12:08 (172.17.22.26)
csea11           pts/20       1   Jul 23 10:28 (172.17.21.27)
csea12           pts/12       1   Jul 23 10:13 (172.17.21.35)
csea13           pts/1        1   Jul 23 10:08 (172.17.22.38)
csea14           pts/21           Jul 23 11:01 (172.17.22.29)
csea15           pts/9            Jul 23 11:25 (172.17.22.130)
root     root    *:0              Jul 23 10:20
staff            pts/26       1   Jul 23 12:20 (172.17.21.21)
```

21. Last

NAME : last
SYNTAX : last
DESCRIPTION : Displays the list of last logged-in users for a month.
OUTPUT :

```
staff    pts/26       172.17.21.21     Mon Jul 23 12:20   still logged in
csea09   pts/3        172.17.22.115    Mon Jul 23 12:13   still logged in
csea04   pts/23       172.17.22.60     Mon Jul 23 12:12 - 12:18  (00:05)
csea23   pts/25       172.17.22.60     Mon Jul 23 12:11 - 12:11  (00:00)
csea10   pts/2        172.17.22.26     Mon Jul 23 12:08   still logged in
csea04   pts/23       172.17.22.60     Mon Jul 23 11:51 - 11:52  (00:00)
csea08   pts/23       172.17.22.60     Mon Jul 23 11:50 - 11:51  (00:01)
csea04   pts/23       172.17.22.60     Mon Jul 23 11:48 - 11:50  (00:01)
csea06   pts/23       172.17.22.60     Mon Jul 23 11:46 - 11:47  (00:01)
csea23   pts/6        172.17.22.14     Mon Jul 23 11:45   still logged in
csea09   pts/3        172.17.22.14     Mon Jul 23 11:44 - 11:45  (00:01)
csea09   pts/6        172.17.22.115    Mon Jul 23 11:38 - 11:42  (00:04)
reboot   system boot  2.4.20-8smp      Mon Jul 23 08:57          (03:24)
wtmp begins Mon Jul  2 10:08:30 2012
```

22. And

NAME : and - &&
SYNTAX : cmd1 && cmd2
DESCRIPTION : Used to combine more than one commands.
EXAMPLE : whoami && date
OUTPUT :

23. Or

NAME : or - ||
SYNTAX : cmd1 || cmd2
DESCRIPTION : Displays the output for one command which is true.
EXAMPLE : whoami || date
OUTPUT :

24. Alias

NAME : alias
SYNTAX : alias name="value"
DESCRIPTION : To create simple names or abbreviations for commands
EXAMPLE : alias p="pwd"

OUTPUT :

25. Edit

NAME : edit
SYNTAX : vi filename
DESCRIPTION : Edits the content of the file. To edit press I and to save press esc:wq
EXAMPLE : vi names.txt

26. cut

NAME : cut

(i) SYNTAX : cut *OPTION -m* [*FILE*]...
DESCRIPTION : extracts m characters from the beginning of each line from the specified file.
EXAMPLE : cut –c -3 test.txt
OUTPUT :
BEFORE EXECUTION : cat > test.txt
Smvec
manakula
vinayagar

AFTER EXECUTION : smv
man
vin

27. Touch

NAME : touch
SYNTAX : touch filename
DESCRIPTION : Creates an empty file.
EXAMPLE : touch hello.txt
OUTPUT :

BEFORE EXECUTION : ls

Sample.txt welcome.txt

AFTER EXECUTION : ls

Sample.txt welcome.txt hello.txt

28. Uniq

NAME : uniq
SYNTAX : uniq option filename1 filename2
DESCRIPTION : Discard all but one of successive identical lines from filename1 to filename2
EXAMPLE : uniq –d a.txt b.txt
OUTPUT :
BEFORE EXECUTION : cat > a.txt
smvec
smvec
manakula
vinayagar
AFTER EXECUTION : cat b.txt
smvec

29. Semicolon

NAME : Semicolon (;)

SYNTAX : cmd1 ; cmd2; cmd3
DESCRIPTION : Similar to 'and' command which combines more than one command.
EXAMPLE : whoami && date
OUTPUT :

30. Echo

NAME : echo – displays a line of text.
SYNTAX : echo "……."
DESCRIPTION : Displays the statement within double quotes.
EXAMPLE : echo "hai"
OUTPUT :

31. Word Count

NAME : wc – word count
(i) SYNTAX : wc filename
DESCRIPTION : Displays the number of lines, words and characters in files.
EXAMPLE : wc a.txt
OUTPUT : a.txt=>

(ii) SYNTAX : wc -l filename
DESCRIPTION : Displays the number of lines in files.
EXAMPLE : wc –l a.txt
OUTPUT : a.txt=>
hai
how are u

2 lines

(iii)SYNTAX : wc -m filename
DESCRIPTION : Displays the number of characters in files.
EXAMPLE : wc –m a.txt
OUTPUT :

(iv)SYNTAX : wc -w filename
DESCRIPTION : Displays the number of words in files.
EXAMPLE : wc –w a.txt
OUTPUT :

32. Grep

NAME : grep
(i) SYNTAX : grep pattern filename
DESCRIPTION : To search for a regular expression or a pattern in a file
EXAMPLE : grep apple b.txt
OUTPUT :

(ii) SYNTAX : grep –c pattern filename
DESCRIPTION : To search for a regular expression or a pattern in a file and displays how many times that pattern is repeated in the file.
EXAMPLE : grep –c apple b.txt
OUTPUT :

(iii) SYNTAX : grep –n pattern filename
DESCRIPTION : To search for a regular expression or a pattern in a file and displays the searched content along with the line and line number, if found.
EXAMPLE : grep –n apple b.txt
OUTPUT :

(iv) SYNTAX : grep –i pattern filename
DESCRIPTION : To search for a regular expression or a pattern in a file irrespective of the case.
EXAMPLE : grep –n APPLE a.txt
OUTPUT :

33. Read

NAME : read – reads a value(s)
SYNTAX : read identifier
DESCRIPTION : Reads a value(s)
EXAMPLE : read a
OUTPUT :

34. Fgrep

NAME : fgrep
(i)SYNTAX : fgrep pattern f1 f2
DESCRIPTION : To search for a regular expression or a pattern in two files
EXAMPLE : fgrep hai a.txt d.txt
OUTPUT :

(ii) SYNTAX : fgrep –c pattern f1 f2
DESCRIPTION : To search for a regular expression or a pattern in two files and displays how many times that pattern is repeated in the files.
EXAMPLE : fgrep –c file a.txt d.txt
OUTPUT :

(iii) SYNTAX : fgrep –n pattern f1 f2
DESCRIPTION : To search for a regular expression or a pattern in two files and displays the searched content along with the line and line number, if found.

EXAMPLE : fgrep –c file a.txt d.txt
OUTPUT :

(iv)SYNTAX : fgrep –i pattern f1 f2
DESCRIPTION : To search for a regular expression or a pattern in two files irrespective of the case.
EXAMPLE : fgrep –c HaI a.txt d.txt
OUTPUT :

35. Pipe

NAME : Pipe - |
SYNTAX : cmd1 | cmd2 | cmd3
DESCRIPTION : Makes the output of one command as input for another command.
EXAMPLE : date | wc -w
OUTPUT :

36. Tee

NAME : Tee
SYNTAX : cmd1 | tee filename
DESCRIPTION : Used to read the standard input and then write to standard output or file.
EXAMPLE : date | tee f.txt | wc -w
OUTPUT :

37. Write

NAME : write
SYNTAX : write login_name
DESCRIPTION : Used to communicate with other logged in users.
EXAMPLE :

38. Mail

NAME : Mail

(i) SYNTAX : mail login_name

DESCRIPTION : Used to send mail to a user.
EXAMPLE :

(ii) SYNTAX : mail
DESCRIPTION : Used to view the mails in the mailbox.

39. Terminal Name

NAME : tty
SYNTAX : tty
DESCRIPTION : Used to display the terminal path name.
OUTPUT :

40. Expression

NAME : expr
SYNTAX : `expr expression`
DESCRIPTION : Used to evaluate an expression
EXAMPLE : echo `expr 10 + 10`
OUTPUT :

CRITERIA	MAX.MARKS	MARKS OBTAINED
AIM & ALGORITHM	5	
EXECUTION & OUTPUT	10	
VIVA	10	
TOTAL	25	

RESULT:

Thus all the Linux commands are executed.

SHELL PROGRAMMING – I

EX.NO: 2a	COUNTING WORDS IN A FILE
DATE:	

AIM:

To write shell program to count the words in file

ALGORITHM:

Step 1: read a file name.
Step 2: using cat command count the word -w.
Step 3 using cat command count the word -c
Step 4: using grep command get no of line in file
Step 5: Stop the program.

PROGRAM:

```
Echo “enter the filename”
Read file
L=`wc –l $file`
W=`wc –w $file`
C=`wc –m $file`
Echo “no of line in $file is $L”
Echo “no of word in $file is $W”
Echo “ no of character in $file is $C”
```

OUTPUT:

```
Enter the file name:
abc.txt
no of line in abc.txt  is 2
no of word in abc.txt is 15
no of character in  abc.txt  is 43
```

RESULT:

Thus the program has been executed successfully.

EX.NO:2b	COUNT OCCURANCE OF A WORDS IN A FILE
DATE:	

EX.NO:2c	FACTORIAL OF NUMBER
DATE:	

AIM:
To find a factorial of a number using shell script.

ALGORITHM:
Step 1: read a number.
Step 2: Initialize fact as 1.
Step 3: Initialize I as 1.
Step 4: While I is lesser than or equal to no.
Step 5: Multiply the value of I and fact and assign to fact increment the value of I by 1.
Step 6: print the RESULT.
Step 7: Stop the program.

PROGRAM:

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

OUTPUT:
Enter the number:
4
The factorial of 4 is 24.

RESULT:
Thus the program has been executed successfully.

EX.NO:2d	FIBONACCI SERIES
DATE:	

AIM:

To write a program to generate a fibonnacci series.

ALGORITHM:

1. Start the program.
2. Get the value of num to generate the fibonnacci series.
3. Initialize a=-1,b=1, and c=0.
4. Check if the num is greater than 0, Add a and b value to store value into c variable goto step 5 else goto step 8.
5. Store the value of b to a, and c to b.
6. Decrement the value n by 1.
7. Print the value of c goto step 4.
8. Stop the program.

PROGRAM:

```
clear
echo "Enter the number"
read num
a=-1
b=1
c=0
echo "Fibnoci series"
while [ $num -gt 0 ]
do
c=` expr $a + $b`
a=$b
b=$c
num=` expr $num - 1`
echo $c
done
```

OUTPUT:

```
Enter the number
4
Fibonacci series
0
1
1
2
```

RESULT:

Thus the various shell programs has been entered and verified .

EX.NO:2e	GREETING MESSAGE BASED ON TIME
DATE:	

AIM:

To write a program to display greeting message.

ALGORITHM:

1. Start the program.
2. Get the value of num to generate the fibonnacci series.
3. Initialize a=-1,b=1, and c=0.
4. Check if the num is greater than 0, Add a and b value to store value into c variable goto step 5 else goto step 8.
5. Store the value of b to a, and c to b.
6. Decrement the value n by 1.
7. Print the value of c goto step 4.
8. Stop the program.

PROGRAM:

```
hour=$(date +"%H")
if [ $hour -ge 0 -a $hour -lt 12 ]
then
    echo "Good Morning"
elif [ $hour -ge 12 -a $hour -lt 18 ]
then
    echo "Good Afternoon"
else
    echo "Good Evening"
fi
```

OUTPUT:

Good morning

RESULT:

Thus the shell programs has been entered and output is verified

EX.NO:2f	STRING COMPARISON
DATE:	

CRITERIA	MAX.MARKS	MARKS OBTAINED
AIM & ALGORITHM	**5**	
EXECUTION & OUTPUT	**10**	
VIVA	**10**	
TOTAL	**25**	

RESULT:

SHELL PROGRAMMING – II

EX.NO:3a	USING LOOPS FINDING EVEN OR ODD
DATE:	

AIM:
To find whether the given number is odd or even .

ALGORITHM:
1. Start the program.
2. Get the value for variable n.
3. Initialize the variable i=0.
4. Check whether i is less than n ,if so divide the value of i by 2.If the remainder is equal to zero then goto step 5 else goto step 6.
5. Print the given number is even number goto step 7.
6. Print the given number is odd number goto step 7.
7. Increment the value of i by 1, goto step 4.
8. Stop the program execution.

PROGRAM:
```
clear
echo "enter the number"
read n
i=0
while [ $i -lt $n ]
do
if [ ` expr $i % 2` -eq 0 ]
then
echo " $i is a even number"
else
echo " $i is a odd number"
fi
i=` expr $i + 1`
done
```

OUTPUT:
```
Enter the number
5
0 is a even number
1 is a odd number
2 is a even number
3 is a odd number
4 is a even number
```

RESULT:

Thus the shell programs has been entered and output is verified

EX.NO:3b	PATTERN PRINTING
DATE:	

AIM:
To print the pattern using a shell program.

ALGORITHM:
1. Start the program.
2. initialize the variable n.
3. Initialize the variable i=0 and j=0.
4. Check whether i is less than n-1 ,if so check the condition j is less than n-1, if so print the pattern #
5. Increment the value of i by 1, goto step 4.
6. Stop the program execution.

PROGRAM:

```
#Bash Shell Script to print half pyramid using *
echo "Enter the number of rows"
read rows
for((i=1; i<=rows; i++))
do
  for((j=1; j<=i; j++))
  do
    echo -n "* "
  done
  echo
done
```

Output:

```
*

* *

* * *

* * * *

* * * * *
```

RESULT:
Thus the shell programs has been entered and output is verified

EX.NO: 3c	EXPANSIONS
DATE:	

AIM:

To read shell expansion

.

Descriptions

Expansion is performed on the command line after it has been split into tokens. There are seven kinds of expansion performed:

- brace expansion
- tilde expansion
- parameter and variable expansion
- command substitution
- arithmetic expansion
- word splitting
- filename expansion

Command Syntax

1.Tilde Expansion

As we recall from our introduction to the **cd** command, the tilde character ("~") has a special meaning. When used at the beginning of a word, it expands into the name of the home directory of the named user, or if no user is named, the home directory of the current user:

Syntax
[me@linuxbox me]$ echo ~
/home/me

2.Arithmetic Expansion

The shell allows arithmetic to be performed by expansion. This allow us to use the shell prompt as a calculator:

[me@linuxbox me]$ **echo $((2 + 2))**

4

Arithmetic expansion uses the form:

$((expression))
[me@linuxbox me]$ echo $(($((5**2)) * 3)) echo $(($((5**2)) * 3))

75

3.Brace Expansion

Perhaps the strangest expansion is called *brace expansion*. With it, we can create multiple text strings from a pattern containing braces. Here's an example:

[me@linuxbox me]$ **echo Front-{A,B,C}-Back**

Front-A-Back Front-B-Back Front-C-Back

4. Parameter Expansion

we're only going to touch briefly on *parameter expansion* in this lesson, but we'll be covering it more later. It's a feature that is more useful in shell scripts than directly on the command line. Many of its capabilities have to do with the system's ability to store small chunks of data and to give each chunk a name. Many such chunks, more properly called *variables*, are available for our examination. For example, the variable named “USER” contains our user name. To invoke parameter expansion and reveal the contents of USER we would do this:

[me@linuxbox me]$ **echo $USER**

me

RESULT:

Thus all the shell expansions are studied.

EX.NO:3d	SUBSTITUTIONS
DATE:	

AIM:
To study the shell substitutions
.
Descriptions

The shell performs substitution when it encounters an expression that contains one or more special characters.

Here, the printing value of the variable is substituted by its value. Same time, **"\n"** is substituted by a new line

```
#!/bin/sh

a=10
echo -e "Value of a is $a \n"
```

You will receive the following RESULT. Here the **-e** option enables the interpretation of backslash escapes.

```
Value of a is 10
```

Following is the RESULT without **-e** option −

```
Value of a is 10\n
```

The following escape sequences which can be used in echo command −

Sr.No.	Escape & Description
1	\\ Backslash
2	**\a** alert (BEL)
3	**\b** Backspace
4	**\c** suppress trailing newline
5	**\f** form feed

6	**\n** new line
7	**\r** carriage return
8	**\t** horizontal tab
9	**\v** vertical tab

You can use the **-E** option to disable the interpretation of the backslash escapes (default).

You can use the **-n** option to disable the insertion of a new line.

Command Substitution

Command substitution is the mechanism by which the shell performs a given set of commands and then substitutes their output in the place of the commands.

Syntax

The command substitution is performed when a command is given as −

`command`

When performing the command substitution make sure that you use the backquote, not the single quote character.

Example

Command substitution is generally used to assign the output of a command to a variable. Each of the following examples demonstrates the command substitution –

```
#!/bin/sh

DATE=`date`
echo "Date is $DATE"

USERS=`who | wc -l`
echo "Logged in user are $USERS"

UP=`date ; uptime`
echo "Uptime is $UP"
```

Upon execution, you will receive the following RESULT −

```
Date is Thu Jul  2 03:59:57 MST 2009
```

Logged in user are 1
Uptime is Thu Jul 2 03:59:57 MST 2009
03:59:57 up 20 days, 14:03, 1 user, load avg: 0.13, 0.07, 0.15

Variable Substitution

Variable substitution enables the shell programmer to manipulate the value of a variable based on its state.

Here is the following table for all the possible substitutions −

Sr.No.	Form & Description
1	**${var}** Substitute the value of *var*.
2	**${var:-word}** If *var* is null or unset, *word* is substituted for **var**. The value of *var* does not change.
3	**${var:=word}** If *var* is null or unset, *var* is set to the value of **word**.
4	**${var:?message}** If *var* is null or unset, *message* is printed to standard error. This checks that variables are set correctly.
5	**${var:+word}** If *var* is set, *word* is substituted for var. The value of *var* does not change.

Example

Following is the example to show various states of the above substitution −

```
#!/bin/sh

echo ${var:-"Variable is not set"}
echo "1 - Value of var is ${var}"

echo ${var:="Variable is not set"}
echo "2 - Value of var is ${var}"

unset var
echo ${var:+"This is default value"}
echo "3 - Value of var is $var"

var="Prefix"
echo ${var:+"This is default value"}
```

```
echo "4 - Value of var is $var"

echo ${var:?"Print this message"}
echo "5 - Value of var is ${var}"
```

Upon execution, you will receive the following RESULT −

```
Variable is not set
1 - Value of var is
Variable is not set
2 - Value of var is Variable is not set

3 - Value of var is
This is default value
4 - Value of var is Prefix
Prefix
5 - Value of var is Prefix
```

CRITERIA	MAX.MARKS	MARKS OBTAINED
AIM & ALGORITHM	**5**	
EXECUTION & OUTPUT	**10**	
VIVA	**10**	
TOTAL	**25**	

RESULT:

Thus all the shell substitutions are studied.

SYSTEM CALLS OF UNIX/LINUX OPERATING SYSTEM

EX.NO: 4a	CREATE A CHILD PROCESS USING FORK
DATE:	

AIM:

To write various system calls.

ALGORITHM:

1. Start the program.
2. Declare the necessary variables.
3. Parent process is the process of the program which is running.
4. Create the child1 process using fork() When parent is active.
5. Create the child2 process using fork() when child1 is active.
6. Create the child3 process using fork() when child2 is active.
7. Stop the process.

PROGRAM:

```
#include<stdio.h>
int main(void)
{
int fork(void),value;
value=fork( );
printf("main:value =%d\n",value);
return 0;
}
```

OUTPUT:

```
[iiiit01@localhost cpro]$ cc pro2.c
[iiiit01@localhost cpro]$ ./a.out

main:value =0
main:value =2860
```

RESULT:

Thus the program for fork system call has been executed successfully.

EX.NO:4b	EXEC ( ) SYSTEM CALL
DATE:	

ALGORITHM:

1. Start the program
2. Declare the necessary variables
3. Use the prototype execv (filename,argv) to transform an executable binary fileinto process
4. Repeat this until all executed files are displayed
5. Stop the program.

PROGRAM:

```
#include<stdio.h>
main()
{
   int pid;
   char *args[]={"/bin/ls","-l",0};
   printf("\nParent Process");
   pid=fork();
   if(pid==0)
   {
            execv("/bin/ls",args);
            printf("\nChild process");
   }
else {
   wait();
   printf("\nParent process");
   printf("\n THE ID NUMBER OF THE CHILD PROCESS IS %d \n",getpid());

   exit(0);
   } }
```

OUTPUT:

```
[iiiit01@localhost cpro]$ cc pro3.c
[iiiit01@localhost cpro]$ ./a.out

 total 440
-rwxrwxr-x 1 skec25 skec25 5210 Apr 16 06:25 a.out
-rw-rw-r-- 1 skec25 skec25 775 Apr 9 08:36 bestfit.c
-rw-rw-r-- 1 skec25 skec25 1669 Apr 10 09:19 correctpipe.c
-rw-rw-r-- 1 skec25 skec25 977 Apr 16 06:15 correctprio.c
-rw------- 1 skec25 skec25 13 Apr 10 08:14 datafile.dat
-rw------- 1 skec25 skec25 13 Apr 10 08:15 example.dat
-rw-rw-r-- 1 skec25 skec25 166 Apr 16 06:25 exec.c
-rw-rw-r-- 1 skec25 skec25 490 Apr 10 09:43 exit.c
Parent Process
```

RESULT:

Thus the program for exec system call has been executed successfully

EX.NO:4c	GETPID ( ) SYSTEM CALL
DATE:	

ALGORITHM:

1. Start the program
2. Declare the necessary variables
3. The getpid() system call returns the process ID of the parent of the Calling process
5. Stop the program.

PROGRAM:

```
#include<stdio.h>
int main()
{
int pid;
pid=getpid();
printf("process ID is %d\n",pid);
pid=getppid();
printf("parent process ID is %d\n",pid);
}
```

OUTPUT:

```
[iiiit01@localhost cpro]$ cc pro4.c
[iiiit01@localhost cpro]$ ./a.out

Process ID is 2848
parent process ID is 2770
```

RESULT:

Thus the program for getpid system call has been executed successfully.

EX.NO:4d	READ( ) AND WRITE( ) SYSTEM CALL
DATE:	

RESULT:

EX.NO:4e	WAIT( ) AND EXIT( ) SYSTEM CALL
DATE:	

AlLGORITHM :

1. Start the program
2. Initialize the necessary variables
3. Use wait() to return the parent id of the child else return -1 for an error
4. Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
int main(void)
{
int pid,status,exitch;
if((pid=fork())==-1)
{
perror("error");
exit (0);
}
if(pid==0)
{
sleep(1);
printf("child process");
exit (0);
}
else
{
printf("parent process\n");
if((exitch=wait(&status))==-1) {
perror("during wait()");
exit (0); }
printf("parent existing\n");
exit (0); }}
```

OUTPUT :

parent process
child process
parent existing

RESULT:

Thus the program for WAIT( ) system call has been executed successfully.

EX.NO:4f	STAT( ) SYSTEM CALL
DATE:	

ALGORITHM:

1. Start the program
2. Declare the variables for the structure stat
3. Allocate the size for the file by using malloc function
4. Get the input of the file whose statistics want to be founded
5. Repeat the above step until statistics of the files are listed
6. Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<stdlib.h>
int main(void)  {
char *path,path1[10];
struct stat *nfile;
nfile=(struct stat *) malloc (sizeof(struct stat));
printf("enter name of file whose statistics has to");
scanf("%s",path1);
stat(path1,nfile);
printf("user id %d\n",nfile->st_uid);
printf("block size :%d\n",nfile->st_blksize);
printf("last access time %d\n",nfile->st_atime);
printf("time of last modification %d\n",nfile->st_atime);
printf("production mode %d \n",nfile->st_mode);
printf("size of file %d\n",nfile->st_size);
printf("number of links:%d\n",nfile->st_nlink);   }
```

OUTPUT:

```
enter name of file whose statistics has to stat.c
user id 621
block size :4096
last access time 1145148485
time of last modification 1145148485
production mode 33204
size of file 654
number of links:1
```

RESULT:

Thus the program for stat system call has been executed successfully.

EX.NO:4g	DIRECTORY SYSTEM CALL
DATE:	

.

ALGORITHM:

1. Start the program.
2. Get the name of the directory through the command line argument.
3. Open directory by executing dirname=opendir(argv[1]).
4. Read the content of the directoryand assigned to the variable preaddr.
5. Check preaddr=null,if so close the directory goto step 6 else print the content of the directory goto step 4.
6. Stop the execution.

PROGRAM:

```
#include<stdio.h>
#include<dirent.h>
main(int argc,char *argv[])
{
DIR *dirname;
struct dirent *preaddr;
dirname=opendir(argv[1]);
while(1)
{
preaddr=readdir(dirname);
if(preaddr==NULL)
{
closedir(dirname);
exit(0);
}
printf("\n\nFOUND ENTRY %s:%s:",argv[1],preaddr->d_name);
}
}
```

OUTPUT:

```
[iiiit01@localhost cpro]$ cc pro1.c
[iiiit01@localhost cpro]$ mkdir hiram
[iiiit01@localhost cpro]$ ./a.out hiram


FOUND ENTRY hiram:.:

FOUND ENTRY hiram:..:
```

RESULT:

Thus the program for stat system call has been executed successfully.

EX.NO:4h	FILE SYSTEM CALL USING OPEN AND CLOSE
DATE:	

ALGORITHM:

1. Start the program.
2. Get the name of the file through the command line argument.
3. Open the file by executing fd=open(argr[1],0),if fd is equal to -1 then print error occur else create a new file by using the statement cr=creat(argr[2],9999).
4. If cr=-1 then print file is not created else a file is successfully created .
5. Read the content of the file by using the statement rd=read(fd,s,size) and write the read content to another file using wd=write(cr,s,size).
6. Stop the execution.

PROGRAM:

```
#include<stdio.h>
#define size 10
main(int argc,char *argr[])
{
int i,n,rd,wd,cr,fd;
char s[size];
if(argc<3)
{
printf("illegal input");
exit(1);
}
fd=open(argr[1],0);
if(fd==-1)
{
printf("error occured");
exit(1);
}
cr=creat(argr[2],9999);
if(cr==-1)
{
printf("file not created");
exit(1);
}
rd=read(fd,s,size);
while(rd>0)
{
wd=write(cr,s,size);
rd=read(fd,s,size);
}
close(fd);
close(cr);
printf("file completed");
}
```

OUTPUT:

```
[iiiit01@localhost cpro]$ cc pro5.c
[iiiit01@localhost cpro]$ vi output
Hi how are u
[iiiit01@localhost cpro]$. /a.out output peruout
File completed
[iiiit01@localhost cpro]$ Cat peruout
Hi how are u
```

RESULT:

Thus the various system calls has been entered and verified

EX.NO:4i	SYSTEM CALL TO FIND MODE OF THE FILE
DATE:	

CRITERIA	MAX.MARKS	MARKS OBTAINED
AIM & ALGORITHM	**5**	
EXECUTION & OUTPUT	**10**	
VIVA	**10**	
TOTAL	**25**	

RESULT:

Thus the various system calls has been entered and verified

SIMULATION OF LINUX COMMANDS

EX.NO:5a	SIMULATION OF CAT COMMAND
DATE:	

AIM:

To write a c program to simulate the Linux command CAT.

ALGORITHM:

1. Start the program.
2. Get the name of the file.
3. Open the file read mode and read the data.
4. Read the content of the file word by word and display it on the screen
5. Stop the execution.

PROGRAM :

```
#include<stdio.h>
#include<string.h>
#define MAX_FILE_NAME_CHARS 255
int main(int argc, char *argv[])
{
 FILE *fp;
 char file_name[MAX_FILE_NAME_CHARS], ch;
 int i;
  if(argc<1){
    printf("Usage mycat <filename> \n");
    return 0;
 }

  for(i=1; i<=argc;i++){
   strncpy(file_name, argv[i], MAX_FILE_NAME_CHARS);
    fp=fopen(file_name, "r");
    if(fp == NULL) {
       printf("%s: No such file or directory\n", file_name);
       return 0;
    }
       while((ch=fgetc(fp)) != EOF){
         putchar(ch);
    }
    fclose(fp);
 }
return 0;
}
```

OUTPUT:

ENTER THE FILE NAME: PCET

THE PATTERN IS FOUND

RESULT:

Thus a C program to simulate CAT command of Unix is written and executed successfully.

EX.NO:5b	SIMULATION OF HEAD AND TAIL COMMAND
DATE:	

AIM:

To write a c program to simulate the Linux command head and tail.

ALGORITHM:

1. Start the program.
2. Get the name of the file.
3. Open the file read mode and read the data.
4. Read the content of the file word by word and display it on the screen
5. Stop the execution.

PROGRAM :

```
#include<stdio.h>
void main(int argc , char *argv[])
{
            FILE  *file;
          char *line[100];
          int count = 0;
                  // initialise file pointer to read
          file  =  fopen(argv[1],"r");
        // read line by line
         while(file , "%[^\n]\n" , line)!=EOF)
         {
                 // break after 10 lines
                 if(count  == 10)
                 {
                       break;
                 }
                 else
                 {
                       printf("%s\n" ,  line);
                  }
                  count++;  }

       fclose(file);      }
```

OUTPUT:

ENTER THE FILE NAME: PCET

THE PATTERN IS FOUND

RESULT:

Thus a C program to simulate HEAD command of Unix is written and executed successfully.

EX.NO:5c	SIMULATION OF GREP
DATE:	

AIM:

To write a c program to simulate the Linux command grep.

ALGORITHM:

1. Start the program.
2. Initialize the flag variable to zero
3. Get the name of the file and pattern to be searched.
4. Open the file in which the searching is going to be performed in read mode
5. Read the contednt of the file word by word and compare it with the string to be searched . If a match occurs go to step 6 else goto step 7.
6. Print the pattern is found .go to step 8
7. Print the pattern is not found .go to step 8
8. Stop the execution.

PROGRAM :

```
#include<stdio.h>
main()
{
FILE *f;
char str[10],strf[10],c[10];
int flag=0;
printf("\nENTER THE PATTERN:");
scanf("%s",str);
f=fopen("cse.txt","r");
while(!feof(f))
{
fscanf(f,"%s",strf);
if(strcmp(str,strf)==0)
{
flag=1;
break;
}
}
if(flag==1)
printf("\nTHE PATTERN IS FOUND\n");
else
printf("\nTHE PATTERN IS NOT FOUND\n");
return 0;
}
```

OUTPUT:

ENTER THE PATTERN: PCET

THE PATTERN IS FOUND

ENTER THE PATTERN: PARK

THE PATTERN IS NOT FOUND

RESULT:

Thus a C program to simulate grep command of Unix is written and executed successfully.

EX.NO:5d	SIMULATION OF FGREP
DATE:	

RESULT:

EX.NO:5e	SIMULATION OF ls COMMAND
DATE:	

AIM:

To write a C program to simulate ls command used in Unix.

Algorithm:

1. Start the program.
2. Get the name of the directory.
3. Open the directory
4. Read the content of the directory
5. Display the content of the directory.
6. Stop the execution.

PROGRAM:

```
#include<dirent.h>
#include<sys/stat.h>
 main()
{
        DIR *dp;
        struct dirent *dir;
        char d[10];
        printf("enter the directory name");
        scanf("%s" ,d);
        dp=opendir(d);
        if(dp!=NULL)
       {
           while((dir=readdir(dp))!=NULL)
                 printf(" %s \n ",dir->d_name);
       }
       else
            printf("\n no such directory found");
       closedir(dp);
       exit(0);
}
```

OUTPUT:

```
Enter the directory name:s05cse15
F1
F2
Vc++
C
```

CRITERIA	MAX.MARKS	MARKS OBTAINED
AIM & ALGORITHM	**5**	
EXECUTION & OUTPUT	**10**	
VIVA	**10**	
TOTAL	**25**	

RESULT:

Thus a C program to simulate ls command of Unix is written and executed successfully.

PROCESS SCHEDULING

EX.NO:6a	IMPLEMENTATION OF FCFS SCHEDULING
DATE:	

AIM:

To write a c program to implement FCFS scheduling.

ALGORITHM:

1: start the program.
2: with burst time, execution time ,arrival Time, waiting time and turnaround time and create object for the structure
3: In main function get the number of processes and burst time arrival time for each processes.
4: calculate the execution time by using the following loop.

- Initially first process execution is zero.
- Next processes execution =previous processes execution time + previous process burst time.

5: waiting time and turn around time can be calculated by

- WAITING TIME=EXECUTIONTIME-ARRIVAL TIME.
- TURNARROUND TIME=WAITING TIME+BURST TIME.

6: Average waiting time and average turn around can be Calculated by

- Average waiting time=sum of waiting time /No of process.
- Average Turnaround time= sum of Turnaround time/ No of processes.

7: Print processes Burst time Arrival time Waiting time Turnaround time for each processes and Average waiting and Turnaround time.
8: Terminate the program.

PROGRAM:

```
#include<stdio.h>
struct fc
{
 int bst,wt,tat,exu;
}p[10];
main()
{
 int pro,i;
 float awt=0,atat=0;
printf("\FCFS PROCESS SCHEDULING ALGORITHM \n");
 printf("\nENTER NUMBER OF PROCESS\n");
 scanf("%d",&pro);
 printf("\nENTER BURST TIME FOR EACH PROCESS\n");
 for(i=0;i<pro;i++)
 {
  scanf("%d",&p[i].bst);
 }
p[i].exu=0;
 for(i=0;i<pro;i++)
```

```
 {
  p[i+1].exu=(p[i].exu+p[i].bst);
 }
for(i=0;i<pro;i++)
 {
  p[i].wt=p[i].exu;
  p[i].tat=p[i].wt+p[i].bst;
 }
 for(i=0;i<pro;i++)
 {
 awt=awt+p[i].wt;
 atat=atat+p[i].tat;
 }
 printf("\nPROCESS   BRUSTTIME    WAITINGTIME  TURNARROUNDTIME");
 for(i=0;i<pro;i++)
 {
 printf("\np %d       %d          %d          %d       %d ",i+1,p[i].bst,p[i].wt,p[i].tat);
 }
printf("\nAVERAGE WAITING TIME= %f",awt/pro);
printf("\nAVERAGE TURNARROUND TIME=%f",atat/pro);
return 0;
}
```

OUTPUT:

```
ENTER NUMBER OF PROCESS
5

ENTER BURST TIME FOR EACH PROCESS
24
3
3

PROCESS     BRUSTTIME       WAITINGTIME    TURNARROUNDTIME
    p 1           24              0              24
    p 2           3               24             27
    p 3           3               27             30


AVERAGE WAIRING TIME= 17.00000
AVERAGE TURNARROUND TIME=27.00000
```

RESULT:

Thus the C program to implement FCFS scheduling has been executed successfully and the output has been verified.

EX.NO:6b	IMPLEMENTATION OF SJF SCHEDULING
DATE:	

AIM:

To write c programs to implement non preemptive SJF scheduling with zero arrival time

ALGORITHM:

1: start the program.

2: Declare a structure with burst time ,execution time,waiting time, and turnaround time ,processes no,variables and create objects for the structure

3: In main function get the number of processes and burst time, Arrival time for each process.

4: Sort the burst time and processes number according to burst time .

5: calculate the execution time by using the following loop.

- Initially first process execution is zero.
- Next processes execution =previous processes execution time + previous process burst time.

6 : waiting time and turn around time can be calculated by

- WAITING TIME=EXECUTIONTIME.
- TURNARROUND TIME=WAITING TIME+BURST TIME.

7: Average waiting time and average turn around can be Calculated by

- Average waiting time=sum of waiting time /No of process.
- Average Turnaround time= sum of Turnaround time/ No of processes.

8: Print processes Burst time Waiting time Turnaround time for each processes and Average waiting and Turnaround time.

9 : Terminate the program.

PROGRAM:

```
#include<stdio.h>
struct fc
{
 int bst,wt,tat,exu,pro;
}p[10],a[10],b[10];
main()
{
 int n,i,j,t,o,k;
 float awt=0,atat=0;
 printf(" **********ARRIVAL TIME IS ZERO**********");
 printf("\nENTER NUMBER OF PROCESS\n");
 scanf("%d",&n);
 printf("\nENTER BURST TIME FOR EACH PROCESS\n");
 for(i=0;i<n;i++)
  scanf("%d",&p[i].bst);
```

```
 for(i=0;i<=n;i++)
  { a[i].pro=i+1;
  a[i].bst=p[i].bst;
  }
 for(i=0;i<n;i++)
 {
  for(j=i+1;j<n;j++)
  {
   if(a[i].bst>a[j].bst)
     {t=a[i].bst;
         a[i].bst=a[j].bst;
         a[j].bst=t;
         k=a[i].pro;
         a[i].pro=a[j].pro;
         a[j].pro=k;
 }}}
a[0].exu=0;
for(i=0;i<n;i++)
  a[i+1].exu=a[i].exu+a[i].bst;
for(i=0;i<n;i++)
{
 a[i].wt=a[i].exu;
 a[i].tat=a[i].wt+a[i].bst;
}
 for(i=0;i<n;i++)
 {
 awt=awt+a[i].wt;
 atat=atat+a[i].tat;
 }
 printf("\nPROCESS BRUSTTIME  WAITINGTIME
        TURNARROUNDTIME");
 for(i=0;i<n;i++)
 {
 printf("\np %d       %d          %d          %d ",a[i].pro,a[i].bst,a[i].wt,a[i].tat);
 }
printf("\nAVERAGE WAITING TIME= %f",awt/n);
printf("\nAVERAGE TURNARROUND TIME=%f",atat/n);
return 0;
}
```

OUTPUT:

```
**********ARRIVAL TIME IS ZERO**********
ENTER NUMBER OF PROCESS
4

ENTER BURST TIME FOR EACH PROCESS
6
8
7
3

PROCESS        BRUSTTIME        WAITINGTIME        TURNARROUNDTIME
   p 4             3                0                  3
   p 1             6                3                  9
   p 3             7                9                  16
   p 2             8                16                 24

AVERAGE WAITING TIME= 7.000000
AVERAGE TURNARROUND TIME=13.000000
```

RESULT:

The C program to implement non preemptive SJF scheduling with zero arrival time has been executed successfully and the output has been verified.

EX.NO:6c	IMPLEMENTATION OF PRIORITY SCHEDULING
DATE:	

AIM:

To write C programs to implement non preemptive priority scheduling with zero arrival time

ALGORITHM:

1: start the program.

2: Declare a structure with burst time, execution time, waiting time, and turnaround time, processes no, variables and create objects for the structure

3: In main function get the number of processes and burst time Arrival time for each process.

4: Sort the burst time and process numbers according to priority.

5: calculate the execution time by using the following loop.

- Initially first process execution is zero.
- Next processes execution =previous processes execution time + previous process burst time.

6 : waiting time and turn around time can be calculated by

- WAITING TIME=EXECUTIONTIME.
- TURNARROUND TIME=WAITING TIME+BURST TIME.

7: Average waiting time and average turn around can be Calculated by

- Average waiting time=sum of waiting time /No of process.
- Average Turnaround time= sum of Turnaround time/ No of processes.

8: Print processes Burst time Waiting time Turnaround time for each processes and Average waiting and Turnaround time.

9: Terminate the program.

PROGRAM:

```
#include<stdio.h>
struct fc
{
 int bst,wt,tat,exu,pro,pr;
}p[10],a[10],b[10];
main()
{
 int n,i,j,t,k,z;
 float awt=0,atat=0;
 printf(" ************ PRIORITY SCHEDULING**************");
 printf("\n********LOW NUMBERS HAVE HIGH PRIORITY*********");
 printf("\nENTER NUMBER OF PROCESS\n");
 scanf("%d",&n);
 printf("\nENTER BURST TIME FOR EACH PROCESS\n");
 for(i=0;i<n;i++)
  scanf("%d",&p[i].bst);
printf("\nENTER  PRIORITY FOR EACH PROCESS\n");
```

```
 for(i=0;i<n;i++)
  scanf("%d",&p[i].pr);

  for(i=0;i<n;i++)
   {
   a[i].pro=i+1;
   a[i].bst=p[i].bst;
   a[i].pr=p[i].pr;
  }

  for(i=0;i<n;i++)
  {
   for(j=i+1;j<n;j++)
   {
    if(a[i].pr>a[j].pr)
      {
        z=a[i].pr;
        a[i].pr=a[j].pr;
         a[j].pr=z;
        t=a[i].bst;
    a[i].bst=a[j].bst;
    a[j].bst=t;
    k=a[i].pro;
    a[i].pro=a[j].pro;
    a[j].pro=k;
      }    }   }
a[0].exu=0;
for(i=0;i<n;i++)
  a[i+1].exu=a[i].exu+a[i].bst;
for(i=0;i<n;i++)
{
 a[i].wt=a[i].exu;
 a[i].tat=a[i].wt+a[i].bst;
}
 for(i=0;i<n;i++)
 {
 awt=awt+a[i].wt;
 atat=atat+a[i].tat;
 }
 printf("\n PROCESSES  BRUSTTIME  PRIORITY   WAITINGTIME TURNARROUNDTIME");
 for(i=0;i<n;i++)
 {
 printf("\np %d       %d          %d               %d         %d ",a[i].pro,a[i].bst,a[i].pr,a[i].wt,a[i].tat);
 }
printf("\nAVERAGE WAITING TIME= %f",awt/n);
printf("\nAVERAGE TURNARROUND TIME=%f",atat/n);
return 0;
}
```

OUTPUT:

********** PRIORITY SCHEDULING**********
*******LOW NUMBERS HAVE HIGH PRIORITY********

```
ENTER NUMBER OF PROCESS
5

ENTER BURST TIME FOR EACH PROCESS
10
1
2
1
5

ENTER  PRIORITY FOR EACH PROCESS
3
1
4
5
2

PROCESS      BRUSTTIME     PRIORITY      WAITINGTIME   TURNARROUNDTIME
   p 2           1            1             0              1
   p 5           5            2             1              6
   p 1           10           3             6              16
   p 3           2            4             16             18
   p 4           1            5             18             19

AVERAGE WAITING TIME= 8.200000
AVERAGE TURNARROUND TIME=12.000000
```

RESULT:

The c programs to implement non preemptive priority scheduling with zero arrival time and non zero arrival time has been executed successfully and the output has been verified.

EX.NO:6d	IMPLEMENTATION OF ROUND ROBIN SCHEDULING
DATE:	

AIM:

To write a c program to implement Round Robin scheduling.

ALGORITHM:

1: start the program.

2: Declare a structure with burst time, execution time, arrival time ,waiting time, and turnaround time , variables and create Objects for that structure.

3: In main function get the number of processes , burst time,Arrival time for each process and time slice.

4:store the burst time in temporary arrays.

5: Sort the burst time arrival time according to burst time and calculate maximum no of execution .

6: Execution time can be calculated by.

- In two loops first loop for max no of execution second loop for noof process.
- If burst time for each process greater than time slice do:
  - Temp var=Temp var+ time slice.
  - Execution time=Temp var.
  - Burst time=burst time –time slice.
- Else burst time less than time slice and greater than zero
  - Temp var =Temp var + burst time.
  - Execution time=Temp var.
  - Burst time=0.

7:Waiting time and turnaround time can be calculated by

- WATING TIME=EXECUTION TIME-ARRIVAL TIME –BRUST TIME
- TURNARROUND TIME=WAITING TIME+BRUST TIME.

8: Average waiting time and average turn around can be Calculated by

- Average waiting time=sum of waiting time /No of process.
- Average Turnaround time= sum of Turnaround time/No of processes.

9: Print processes Burst time Arrival time waiting time Turnaround time for each processes and Average waiting and Turnaround time.

10: Terminate the program.

PROGRAM:

```
#include<stdio.h>
struct fc
{
 int bst,wt,tat,exu,n,ar;
}p[10],a[10],b[10];
main()
```

```
{
 int pn,i,j,ts,t,x,y=0;
 float awt=0,atat=0;
 printf("\nENTER NUMBER OF PROCESS\n");
 scanf("%d",&pn);
 printf("\nENTER BURST TIME FOR EACH PROCESS\n");
 for(i=0;i<pn;i++)
 {
  scanf("%d",&p[i].bst);
 }

printf("\nENTER THE TIME SLICE :");
  scanf("%d",&ts);

for(i=0;i<pn;i++)
  {
  a[i].bst=p[i].bst;
  b[i].bst=p[i].bst;
  }
for(i=0;i<pn;i++)
  {
   for(j=i+1;j<pn;j++)
   {
    if(a[i].bst>a[j].bst)
      {
        t=a[i].bst;
     a[i].bst=a[j].bst;
     a[j].bst=t;
          }    }   }
  x=a[pn-1].bst/ts;
 for(j=0;j<x+5;j++)
 {
  for(i=0;i<pn;i++)
  {
   if(p[i].bst>ts)
     {
      y=y+ts;
      p[i].exu=y;
      p[i].bst=p[i].bst-ts;
      }
   else if((p[i].bst>0) && (p[i].bst<ts))
    {
    y=y+p[i].bst;
    p[i].exu=y;
    p[i].bst=0;

    }   }  }
for(i=0;i<pn;i++)
{
 p[i].wt=p[i].exu-b[i].bst;
```

```
 p[i].tat=p[i].wt+b[i].bst;

}
 for(i=0;i<pn;i++)
 {
 awt=awt+p[i].wt;
 atat=atat+p[i].tat;
 }
 printf("\nPROCESS BRUSTTIME   WAITINGTIME
       TURNARROUNDTIME");
 for(i=0;i<pn;i++)
 {
 printf("\np %d       %d          %d          %d   ",i,b[i].bst,p[i].wt,p[i].tat);
 }
printf("\nAVERAGE WAIRING TIME= %f",awt/pn);
printf("\nAVERAGE TURNARROUND TIME=%f",atat/pn);
return 0;
}
```

OUTPUT:

```
ENTER NUMBER OF PROCESS
3

ENTER BURST TIME FOR EACH PROCESS
24
3
3

ENTER THE TIME SLICE: 4

PROCESS   BRUSTTIME      WAITINGTIME     TURNARROUNDTIME
     p 0              24                  6                   30
     p 1               3                  4                    7
     p 2               3                  7                   10

AVERAGE WAITING TIME= 5.666667

AVERAGE TURNARROUND TIME=15.666667[s06cse34@fileserver today]$ ./a.out
```

RESULT:

The C program to implement RR scheduling has been executed successfully and the output has been verified.

EX.NO:6d	IMPLEMENTATION OF SHORTEST REMAINING TIME FIRST SCHEDULING
DATE:	

CRITERIA	MAX.MARKS	MARKS OBTAINED
AIM & ALGORITHM	5	
EXECUTION & OUTPUT	10	
VIVA	10	
TOTAL	25	

RESULT:

DEADLOCK HANDLING

EX.NO: 7a DATE:	BANKER'S ALGORITHM FOR DEADLOCK AVOIDANCE

AIM:

To write a c program to implement the Banker's Algorithm for Deadlock Avoidance.

DESCRIPTION

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

PROGRAM

```
#include<stdio.h>
struct file
{
int all[10];
int max[10];
int need[10];
int flag;
};
void main()
{
struct file f[10];
int fl;
int i, j, k, p, b, n, r, g, cnt=0, id, newr;
int avail[10],seq[10];
clrscr();
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++)
{
f[i].need[j]=f[i].max[j]-f[i].all[j];
if(f[i].need[j]<0)
f[i].need[j]=0;
}
}
cnt=0;
fl=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");
}
getch();
}
```

OUTPUT:

```
Enter number of processes     –      5
Enter number of resources     --     3
Enter details for P0
Enter allocation        --    0      1      0
Enter Max               --          7      5      3
Enter details for P1

Enter                   --    2      0      0
```

allocation
Enter Max -- 3 2 2
Enter details for P2

Enter -- 3 0 2
allocation
Enter Max -- 9 0 2
Enter details for P3

Enter -- 2 1 1
allocation
Enter Max -- 2 2 2
Enter details for P4

Enter -- 0 0 2
allocation
Enter Max -- 4 3 3
Enter Available Resources 3 3 2
--
Enter New Request Details
--
Enter pid -- 1
Enter Request for -- 1 0 2
Resources

OUTPUT

P1 is 5 3 2)
visited(
P3 is 7 4 3)
visited(
P4 is 7 4 5)
visited(
P0 is 7 5 5)
visited(
P2 is visited(5 7)
10
SYSTEM IS IN SAFE
STATE
The Safe Sequence is -- (P1 P3 P4 P0
P2)

Process	Allocation	Max	Need

P0	0 1 0	7 5 3	7 4 3
P1	3 0 2	3 2 2	0 20
P2	3 0 2	9 0 2	6 00
P3	2 1 1	2 2 2	0 11
P4	0 0 2	4 3 3	4 3 1

CRITERIA	MAX.MARKS	MARKS OBTAINED
AIM & ALGORITHM	**5**	
EXECUTION & OUTPUT	**10**	
VIVA	**10**	
TOTAL	**25**	

RESULT:

Thus the **C** program to implement Bankers algorithm for deadlock avoidance been entered and verified

SEMAPHORE

EX.NO: 8a	PRODUCER AND CONSUMER PROBLEM
DATE:	

AIM:

To write a c program to implement the Producer and consumer problem using semaphore

ALGORITHM:

1.Start the program.
2. Initialize the following variables
   mutex=1;
   empty=5;
   full=0;
3. Display the menu and get the choice.
4. If choice is 1,the producer process is executed,goto step 8.
5. If choice is 2 ,the consumer process is executed goto step 9.
6. If choice is 3, then both producer and consumer processes are executed goto step 10.
7. If choice is 4,go to step 11.
8. Check whether empty equal to zero, if so producer has to wait else allow the producer to produce the item. Go to step 3
9. Check whether full equal to zero, if so consumer has to wait else allow the consumer to consume the item. Go to step 3
10. Check whether empty buffer is available, if available invoke producer else invoke consumer. Go to step 3.
11. Stop the program.

PROGRAM

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
static int full,empty,mutex;
int buffer[5],in=0,out=0;
void wait(int *a);
void signal(int *b);

void producer()
{
int nextp;
printf("producer\n");
wait(&empty);
wait(&mutex);
nextp=rand()%10+1;
buffer[in]=nextp;
printf("produced item is %d\n",nextp);
in=(in+1)%5;
signal(&mutex);
signal(&full);
printf("full=%d\t empty=%d\n",full,empty);
```

```
}

void consumer()
{
int nextc;
printf("consumer\n");
wait(&full);
wait(&mutex);
nextc=buffer[out];
printf("consumerd item is %d\n",nextc);
out=(out+1)%5;
signal(&mutex);
signal(&empty);
printf("full=%d\t empty=%d\n",full,empty);
}

void wait(int *a)
{
while(*a<=0);
*a=*a-1;
}

void signal(int *b)
{
*b=*b+1;
}

main()
{
int c;
mutex=1;
empty=5;
full=0;
clrscr();
while(1)
{
printf("1.producer\t 2.consumer\t 3.both\t 4.Exit\n");
printf("choice\n");
scanf("%d",&c);
switch(c)
{
case 1:
if(empty==0)
 printf("producer has to wait\n");
else
{
producer();
}
break;
case 2:
```

```
if(full==0)
 printf("consumer has to wait");
else
{
consumer();
} break;
case 3:
if(!empty)
{
 printf("producer has to wait\n");
consumer();
}
else if(!full)
{
 printf("consumer has to wait\n");
 producer();
}
else
{
consumer();
producer();
}
break;
case 4:
exit(0);
break;
}
}
getch();
return 0;
}
```

OUTPUT:

```
1.producer      2.consumer      3.both  4.Exit
choice
1
producer
produced item is 7
full=1   empty=4

1.producer      2.consumer      3.both  4.Exit
choice
1
producer
produced item is 1
full=2   empty=3
```

```
1.producer      2.consumer      3.both  4.Exit
choice
2
consumer
consumerd item is 7
full=1   empty=4

1.producer      2.consumer      3.both  4.Exit
choice
2
consumer
consumerd item is 1
full=0   empty=5

1.producer      2.consumer      3.both  4.Exit
choice
2
consumer has to wait1.producer   2.consumer      3.both  4.Exit
choice
1
producer
produced item is 3
full=1   empty=4

1.producer      2.consumer      3.both  4.Exit
choice
3
consumer
consumerd item is 3
full=0   empty=5
producer
produced item is 1
full=1   empty=4

1.producer      2.consumer      3.both  4.Exit
Choice          4
```

RESULT:

Thus the **C** program to implement semaphore using Producer Consumer problem has been entered and verified

EX.NO: 8b	DINING PHILOSOPHER PROGRAM
DATE:	

CRITERIA	MAX.MARKS	MARKS OBTAINED
AIM & ALGORITHM	5	
EXECUTION & OUTPUT	10	
VIVA	10	
TOTAL	25	

RESULT: