INDEX

S.No	Exp No.	Date	Name of the Experiment	Page	Marks obtained	Teacher's Initial
1	1	23/02	BASIC COMMANDS IN LINUX			
2	2	03/03	CREATE A NEW PROCESS USING FORK AND VFORK			
3	3	05/03	IMPLEMENTATION OF GREP SYSTEM CALL			
4	4	12/03	SHELL PROGRAMMING			
5	5A	31/03	IMPLEMENTATION OF FCFS SCHEDULING ALGORITHM			
6	5B	31/03	IMPLEMENTATION OF SJF SCHEDULING ALGORITHM			
7	5C	31/03	IMPLEMENTATION OF ROUND ROBIN ALGORITHM			
8	5D	31/03	PRIORITY SCHEDULING ALGORITHM			
9	6	07/04	IMPLEMENTATION OF SEMAPHORES			
10	7	21/04	INTERPROCESS COMMUNICATION			
11	8	28/04	IMPLEMENTATION OF BANKER'S ALGORITHM			
12	9	05/05	IMPLEMENTATION OF DEADLOCK DETECTION			
13	10	12/05	THREADING AND SYNCHRONIZATION			

INDEX

S.No.	Exp.No.	Date	Name of the Experiment	Page	Marks obtained	Teacher's Initial
14	11	19/05	IMPLEMENTATION OF MEMORY ALLOCATION METHODS			
15	12	26/05	IMPLEMENTATION OF PAGING TECHNIQUES			
16	13A	02/06	IMPLEMENTATION OF FIFO PAGE REPLACEMENT ALGORITHM			
17	13B	02/06	IMPLEMENTATION OF LRU PAGE REPLACEMENT ALGORITHM			
18	14	09/06	IMPLEMENTATION OF VARIOUS FILE ORGANIZATION TECHNIQUES			
19	15A	16/06	IMPLEMENTATION OF CONTIGUOUS FILE ALLOCATION			
20	15B	16/06	IMPLEMENTATION OF LINKED FILE ALLOCATION			
21	15C	16/06	IMPLEMENTATION OF INDEX FILE ALLOCATION			

EX. NO. : 1 **BASIC COMMANDS IN LINUX**
DATE:

AIM: To study the basic commands in Linux.

COMMANDS:

1. TASK : To display the system date and time.
COMMAND : date.
SYNTAX : date.
EXPLANATION: This command displays the current system date and time on the screen.
OUTPUT : Tue Jun 19 11:37:17 GMT 2007.

2. TASK : To display the current month.
COMMAND : date.
SYNTAX : date +%m.
EXPLANATION: This command displays the current month on the screen.
OUTPUT : 06.

3. TASK : To display the name of the current month.
COMMAND : date.
SYNTAX : date +%h.
EXPLANATION: This command displays the name of the current month on the screen.
OUTPUT : Jun.

4. TASK : To display the current system date.
SYNTAX : date +%d.
EXPLANATION: This command displays the current system date on the screen.
OUTPUT : 19.

5. TASK : To display the current system date (year).
SYNTAX : date +%y.
EXPLANATION: This command displays the current year on the screen.
OUTPUT : 09

6. TASK : To display the current system time.
SYNTAX : date +%H.
EXPLANATION: This command displays the current system time (in hours) on the screen.
OUTPUT : 11.

7. TASK : To display the current system time.

SYNTAX : date +%M.
EXPLANATION: This command displays the current system time (in minutes) on thescreen.
OUTPUT : 43.

8. TASK : To display the current system time.
SYNTAX : date +%S.
EXPLANATION: This command displays the current system time (in seconds) on thescreen.
OUTPUT : 15.

9. TASK : To display the calendar of the current month.
COMMAND : calendar.
SYNTAX : cal.
EXPLANATION: This command displays the calendar of the current month on the screen.
OUTPUT : Jun 07

S M T W T F S
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

10. TASK : To display user-defined messages.
COMMAND : echo.
SYNTAX : echo “message”.
EXPLANATION: This command displays on the screen the argument of the echo command.
OUTPUT : echo “OS”.

 OS

11. TASK : To display the details of all users.
COMMAND : who.
SYNTAX : who.
EXPLANATION : This command lists the information about all the users who have logged on to that system.
OUTPUT :2

12. TASK : To display the user detail.
COMMAND : who.
SYNTAX : whoami.
EXPLANATION : This command displays information about the current user of the system on the screen.
OUTPUT : root.

13. TASK : To create a directory.
COMMAND : make directory.
SYNTAX : mkdir.
EXPLANATION : This command is used to create a new directory with the specified name.
EXAMPLE : mkdir student.
OUTPUT : The directory “**student**” is created.

14. TASK : To change directory.
COMMAND : change directory.
SYNTAX : cd directory name.
EXPLANATION : This command is used to switch from one directory to another.
EXAMPLE : cd staff.
OUTPUT : The directory "**staff**" is switched onto.

15. TASK : To delete a directory.
COMMAND : remove directory.
SYNTAX : rmdir directory name
EXPLANATION : This command is used to delete the specified directory.
EXAMPLE : rmdir student.
OUTPUT : The "**student**" directory is deleted.

16. TASK : To come out of a sub-directory.
COMMAND : change directory.
SYNTAX : cd ..
EXPLANATION : This command helps in switching to the main directory.
OUTPUT :

17. TASK : To list all the files and directories.
COMMAND : list.
SYNTAX : ls.
EXPLANATION : This command displays all the files and directories of the system.
OUTPUT :

18. TASK : To create a file.
COMMAND : cat.
SYNTAX : cat> file name.
EXPLANATION : This command leads to the creation of a new file with the specified name and contents.
EXAMPLE : cat> wind.
OUTPUT : A null file called "**wind**" is created.

19. TASK : To view a file.
COMMAND : cat.
SYNTAX : cat file name.
EXPLANATION : This command displays the contents of the specified file.
EXAMPLE : cat wind.
OUTPUT : Contents of the file called "**wind**" will be displayed on the screen.

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

EXAMPLE : cp sun moon.
OUTPUT : The contents of "**sun**" file will be copied to the "**moon**" file.

21. TASK : To move a file.
COMMAND : move.
SYNTAX : mv sourcefile destinationfile.
EXPLANATION : After moving the contents of the source file into the destination file, the source file is deleted.
EXAMPLE : mv sun moon.
OUTPUT : After copying contents from the "**sun**" file to "**moon**" file, the "**sun**" file is deleted.

22. TASK : To display / cut a column from a file.
SYNTAX : cut –c no. file name.
EXPLANATION : This command displays the characters of a particular column in a specified file.
EXAMPLE : cut –c3 moon.
OUTPUT : Those characters occurring in the 3 rd column of the file called "**moon**" are displayed.

23. TASK : To delete a file.
COMMAND : remove.
SYNTAX : rm file name.
EXPLANATION : This command deletes the specified file from the directory.
EXAMPLE : rm sun.
OUTPUT : The file called "**sun**" will be deleted.

24. TASK : To retrieve a part of a file.
COMMAND : head.
SYNTAX~~: head n~~o. of rows file name.
EXPLANATION : This command displays the specified no. of rows for the top of the specified file.
EXAMPLE : head -1 sun.
OUTPUT : The first row of the file called "**sun**" is displayed.

25. TASK : To retrieve a file.
COMMAND : tail.
SYNTAX~~: tail n~~o. of rows file name.
EXPLANATION : This command displays the specified no. of rows from the bottom of the specified file.
EXAMPLE : tail -1 moon.
OUTPUT : The last row of the file called "**moon**" is displayed.

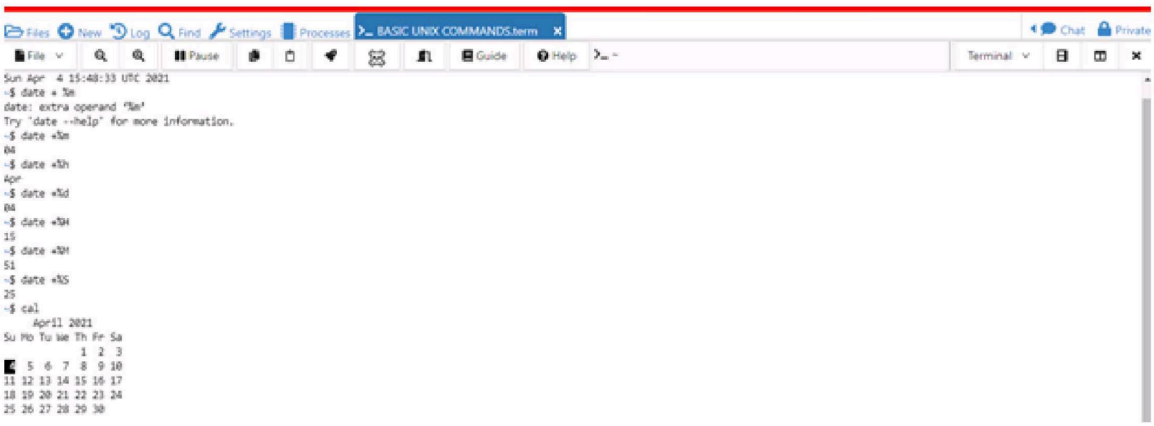
26. TASK : To sort the contents of a file.
COMMAND : sort.
SYNTAX : sort file name.
EXPLANATION : This command helps in sorting the contents of a file in ascending order.
EXAMPLE : sort win.
OUTPUT : The contents of the file "**win**" are displayed on the screen in a sorted order.

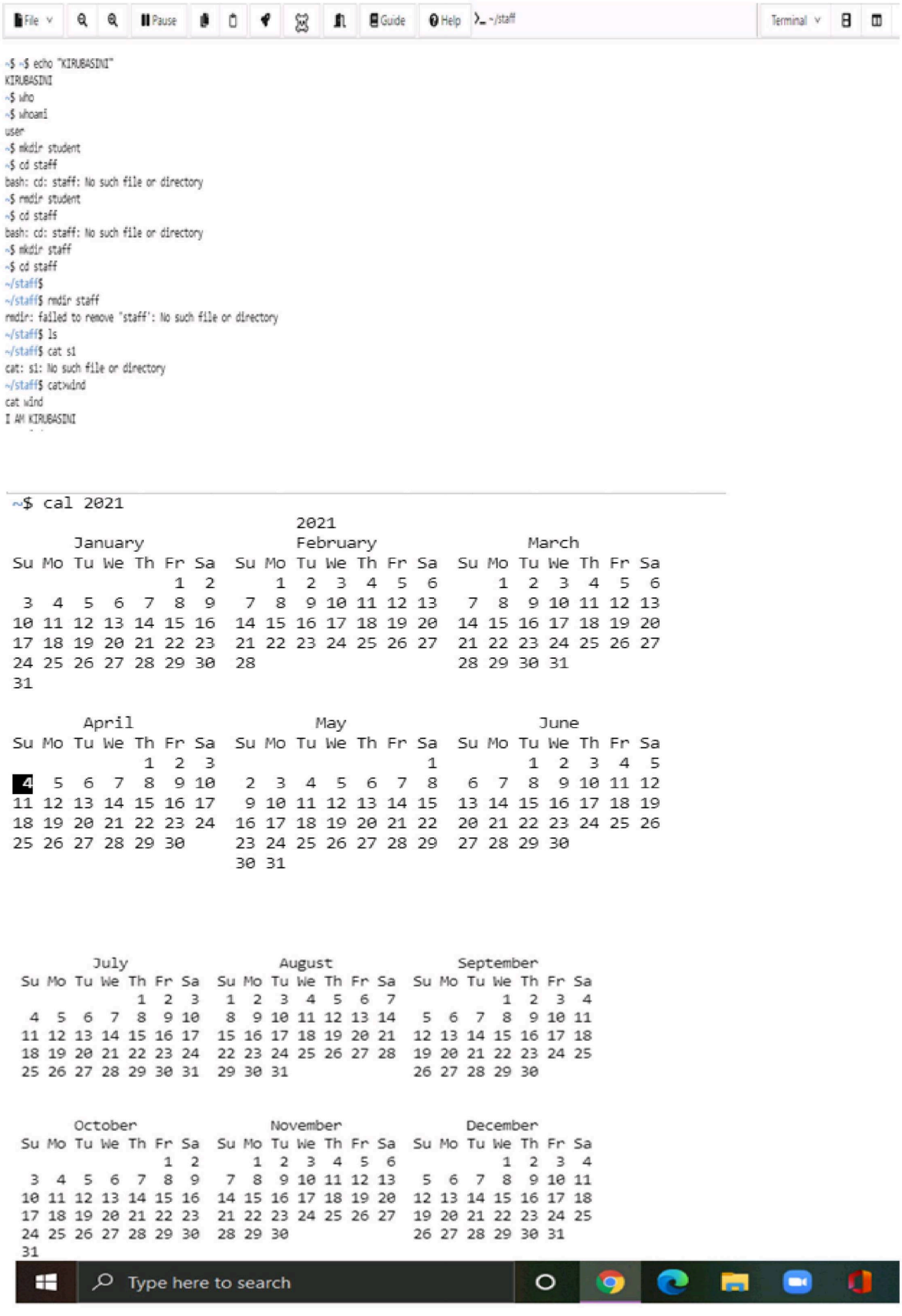
27. TASK : To display the no. of characters in a file..

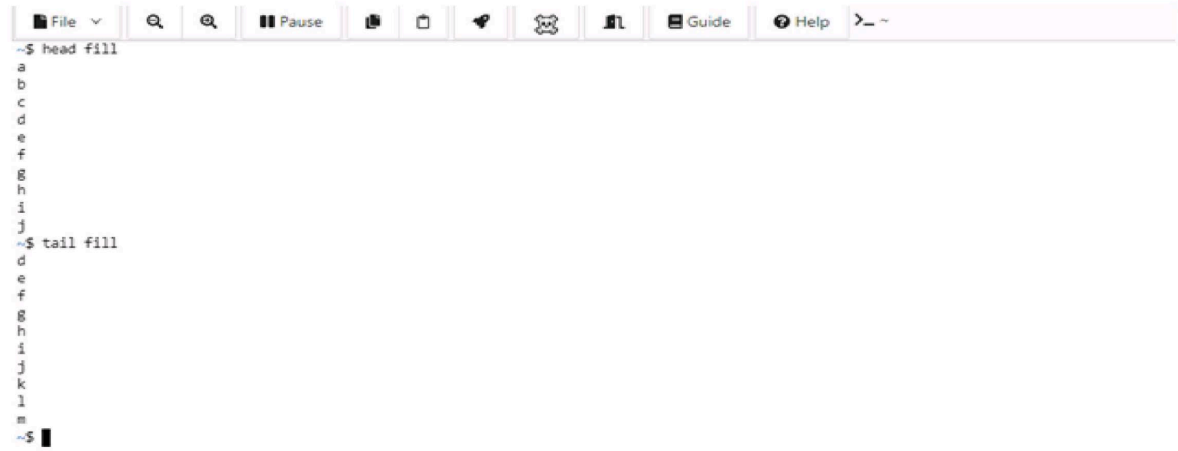
SYNTAX : wc file name.
EXPLANATION : This command displays on the screen the no. of rows, words, and the sum of no**.** of characters and words.
EXAMPLE : wc ball.
OUTPUT : The no. of rows, words, and no. of characters present in the file "**ball**" are displayed.

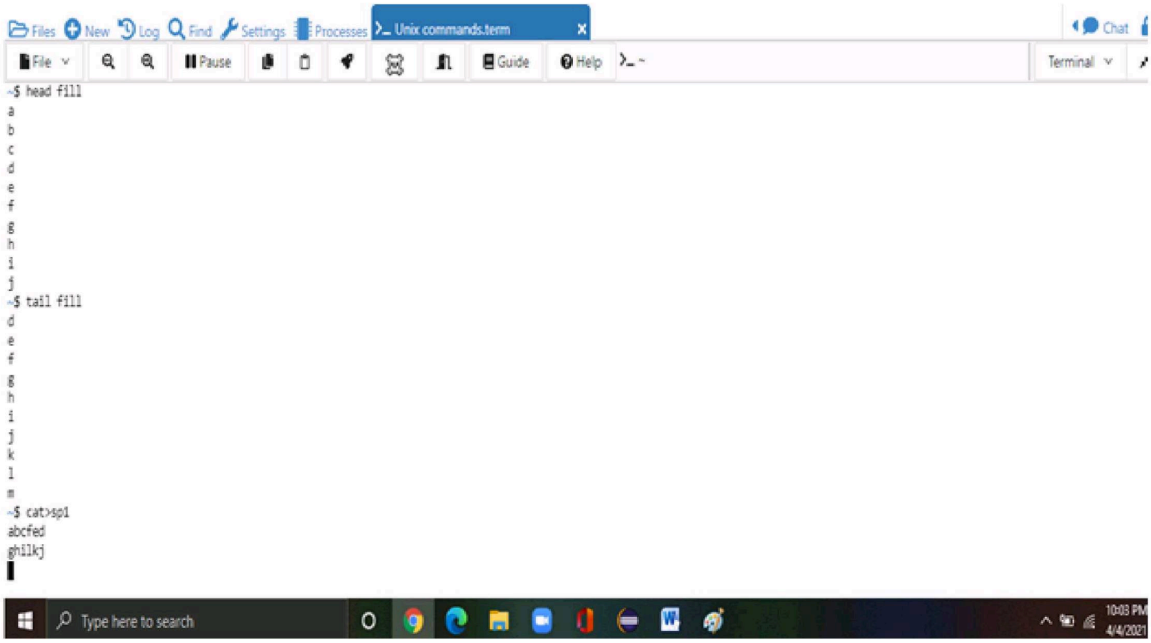
28. TASK : To display the calendar of a year.
COMMAND : cal.
SYNTAX : cal year.
EXPLANATION : This command displays on the screen the calendar of the specified year.
EXAMPLE : cal 2007.
OUTPUT : The calendar of the year **2007** will be displayed.

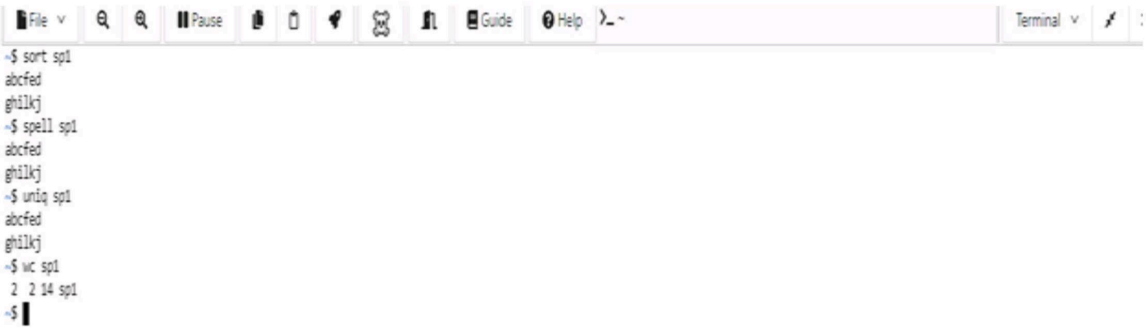
****************OUTPUT FOR BASIC LINUX COMMANDS*******************




File
Pause
Guide
Help
~/staff
Terminal
~$ ~$ echo "KIRUBASINI"
KIRUBASINI
~$ who
~$ whoami
user
~$ mkdir student
~$ cd staff
bash: cd: staff: No such file or directory
~$ rmdir student
~$ cd staff
bash: cd: staff: No such file or directory
~$ mkdir staff
~$ cd staff
~/staff$
~/staff$ rmdir staff
rmdir: failed to remove 'staff': No such file or directory
~/staff$ ls
~/staff$ cat s1
cat: s1: No such file or directory
~/staff$ catwind
cat wind
I AM KIRUBASINI
~$ cal 2021
2021
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
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 29 30
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 27 28 29
30 31
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 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
Type here to search


File
Pause
Guide
Help
~$ head fill
a
b
c
d
e
f
g
h
i
j
~$ tail fill
d
e
f
g
h
i
j
k
l
m
~$


Files
New
Log
Find
Settings
Processes
Unix commands.term
Chat
File
Pause
Guide
Help
Terminal
~$ head fill
a
b
c
d
e
f
g
h
i
j
~$ tail fill
d
e
f
g
h
i
j
k
l
m
~$ cat>sp1
abcfed
ghilkj
Type here to search
10:03 PM
4/4/2021



RESULT:

Thus the above commands were created and executed successfully.

EX.NO :2 WRITE PROGRAMS USING THE FOLLOWING SYSTEM CALLS OF UNIX

DATE : OPERATING SYSTEM FORK AND VFORK.

AIM:

To write a program to create a new process using fork and vfork.

ALGORITHM:

Fork()

1. Create a child process using fork ().
2. Check the process id returned by fork, if the process id is 0 displays the running process is child.
3. If the process id is non-zero, the running process is the parent process.
4. Show that the variables are not shared between parent and child process

Vfork().

1. Create a child process using fork ().
2. Check the process id returned by fork, if the process id is 0 displays the runningprocess is child.
3. Check the process id returned by fork, if the process id is non-zero display the running process is parent.
4. Show how the variables are not shared between parent and child processes.

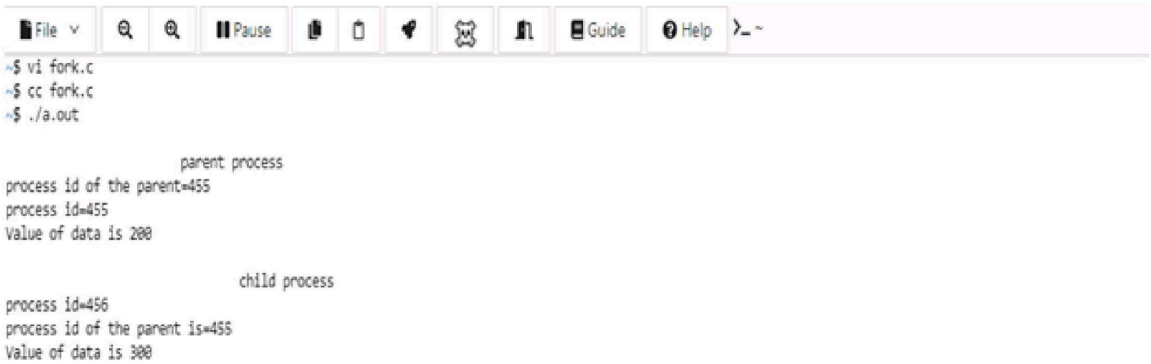
SOURCE CODE:

FORK()

```
#include<sys/types.h>
#include<unistd.h>
#include<stdio.h>
int main()
{
int data=100;
int pid;
pid=vfork();
if(pid==0)
{
printf("child process\n");
data=data+100;
printf("process id is %d\n",getpid());
printf("procee id of parent is %d\n",getppid());
printf("value of data is %d",data);
printf("\n.....\n");
```

```
}
else
{
printf("parent process\n");
printf("process id is %d\n",getpid());
printf("process id of parent is %d\n",getppid());
printf("value of data is %d\n",data);
}
return 0;
}
```

OUTPUT:



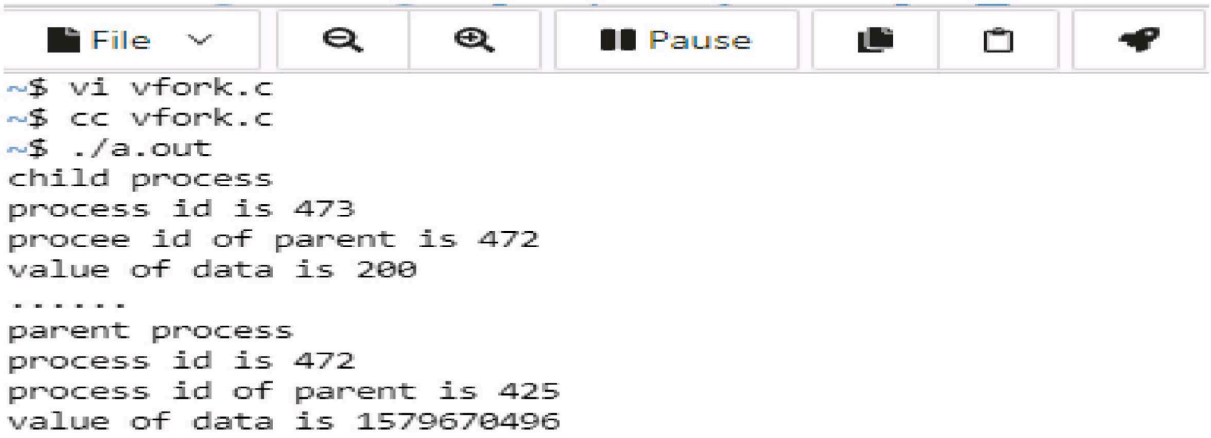
SOURCE CODE:

VFORK():

```
#include<sys/types.h>
#include<unistd.h>
  main()
  {
  int data=100;
  int pid;
  pid=vfork();
  if(pid==0)
  {
  printf("child process\n");
  data=data+100;
  printf("process id is %d\n",getpid());
  printf("procee id of parent is %d\n",getppid()); printf("value
  of data is %d",data);
  printf("\n.....\n");
  }
  else
  {
  printf("parent process\n");
  printf("process id is %d\n",getpid());
```

```
printf("process id of parent is %d\n",getppid()); printf("value
of data is %d\n",data);
}
}
```

OUTPUT:



RESULT:

Thus the above programs were created and executed successfully.

EX.NO : 3 **IMPLEMENTATION OF GREP SYSTEM CALL**
DATE :

AIM:

To implement the unix command ‘grep’ which displays the files in the directory.

ALGORITHM:

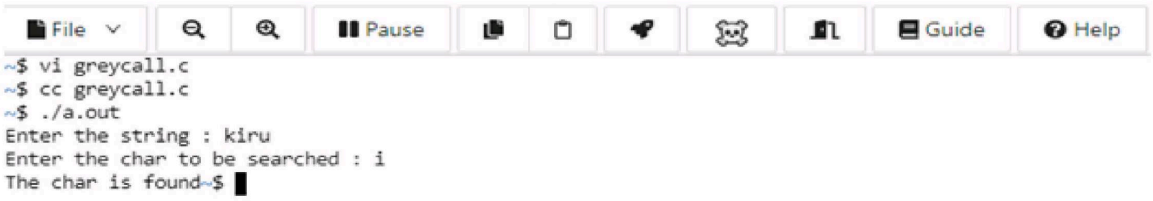
1. Include the header file dirent.h.
2. DIR is the internal structure to maintain information about the directory beingread. 3. DECLARE the structures dirent to get the files in the directory.
4. Open the directory given at the command line.
5. Set a while loop to read the files under the directory.
6. Display the names of the files that reside in the given directory. 7. Lose the directory that was opened.

SOURCE CODE:

```
#include<stdio.h>

void main()

{

int i=0,j=0;

char a[5],b[1];

printf("Enter the string : ");

scanf("%s",a);

printf("Enter the char to be searched : ");

scanf("%s",b);

while(j<5)

{

if(a[j]==b[0])

i++;

j++;

}

if(i!=0)

printf("The char is found");
```

```
else

printf("\n Not found");

}
```

OUTPUT:



RESULT:

Thus the above program was created and executed successfully.

EX. NO. :4 **SHELL PROGRAMMING**
DATE :

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

How to work with shells ?

Step1:
In the dollar prompt type
$ vi < filename>
Where vi is the editor ,it will open a new window in which you can type the program you want

Step2:
After typing the program press ESC and : together then at the bottom of the vi screen can see i.e. prompt .In that type as wq which means write and quit i.e. the content what is typed will be written and saved into that file that has been created

Step3:
Once wq is typed at the : prompt ,the prompt would change to $ symbol in which you have to do the following

$ sh < file name >

Sh – command is used to run the shell program <file name> - is the name of the file for which the output is to be got

Basically to print a text in the your shell programs **echo** command is used **IF –THEN-ELSE**

CONSTRUCT

```
if [ condition]
then <action>
else
statements
fi (end of if)
```

CASE

```
Case $<option> in
    1) <statements>;;
    2) <statements>;;
    3) ..
    4)
     .
     .
     .
     *) <error statement>;;
   esc
```

SYNTAX FOR LOOPING STATEMENTS

WHILE

```
      while <condition>
          do
              <statements>
           Done
```

FOR loop

```
For(( initialization;condition;incrementation/decrementation))
```

PROGRAM 1:

```
echo "Enter the number:"
 read number
 i= 1
fact= 1
while [$i –le $number]
 do
 fact= `expr $fact \* $i `
 i= `expr $i +1`
done
echo "The factorial of $number is $fact."
```

OUTPUT:

```
Enter the number: 3
The factorial of 3 is 6.
```

PROGRAM 2:

```
echo "Enter two numbers:"
read a
read b
c=`expr $a + $b `
echo "The sum is $c."
```

OUTPUT:

Enter two numbers:
3
6
The sum is 9.

PROGRAM 3:

```
echo “Enter the number:”
read n
x= $n
sum= 0
while [ $n –gt 0]
do
y= `expr $n % 10`
z= `expr $y \* $y \* $y`
sum= `expr $sum + $z`
n= `expr $n / 10`
done
if [ $x –eq $sum]
then
echo “ $x is an Armstrong number.”
else
echo “$x is not an Armstrong number.”
Fi
```

OUTPUT 1:

Enter a number:
153
153 is an Armstrong number.

OUTPUT 2:

Enter a number: 33
33 is not an Armstrong number.

PROGRAM 4:

```
echo “Enter the first number:”
read a
echo “Enter the second number:”
read b
echo “Enter the third number:”
read c
if [$a –gt $b –a $a –gt $c]
then
echo “$a is greater.”
elif [$b –gt $c]
then
echo “$b is greater.”
else
```

```
echo “$c is greater.”
fi
```

OUTPUT:

```
Enter the first number: 3
Enter the second number:6
Enter the third number: 9
9 is greater.
```

PROGRAM 5:

```
echo “Enter any two numbers:”
read a
read b
echo “The numbers are $a and $b.”
c= $a
a= $b
b= $c
echo “The swapped numbers are $a and $b.”
```

OUTPUT:

```
Enter any two numbers:
3
9
The numbers are 3 and 9.
The swapped numbers are 9 and 3.
```

PROGRAM 6:

```
a= -1
b= 1
c= 0
echo “Enter the limit:”
read n
echo “FIBONACCI SERIES”
for (i =1; i <=n; i++)
do
c= `expr $a + $b`
echo $c
a=$b
b=$c
Done
```

OUTPUT:

```
Enter the limit:
5
FIBONACCI SERIES
0
1
1
2
```

3

PROGRAM 7:

```
echo “enter the day of the week (1-7)
read n
case $n in
1)echo ”1 is Sunday”;;
2)echo ”2 is Monday”;;
3)echo “3.is Tuesday”;;
4)echo ”2 is Wednesday”;;
5)echo “3.is thursday”;;
6)echo ”2 is Friday”;;
7)echo “3.is Saturday”;;
*)echo ”Invalid option”;;
```

OUTPUT 1:

```
Enter the day of the week(1-7) 1
Sunday
```

OUTPUT 2:

```
Enter the day of the week(1-7)
9
Invalid option
```

PROGRAM 8:

```
echo “ enter string 1”
read s1
echo “enter string2”
read s2
if [ $s1 == $s2]
then
echo “ the given string are equal”
else
echo “The given strings are not equal”
fi
```

OUTPUT 1:

```
enter string 1 ram
enter string 2 sam
The given strings are not equal
```

OUTPUT 2:

```
enter string 1
sam
The given strings are equal
```

RESULT:

Thus the above program was created and executed successfully.

IMPLEMENTATION OF CPU SCHEDULING ALGORITHMS

EX. NO. :5

DATE :

(a) Non-Pre-emptive Scheduling

IMPLEMENTATION OF FIRST COME FIRST SERVE SCHEDULING ALGORITHM

AIM:

To write a program to implement the FCFS scheduling algorithm

ALGORITHM:

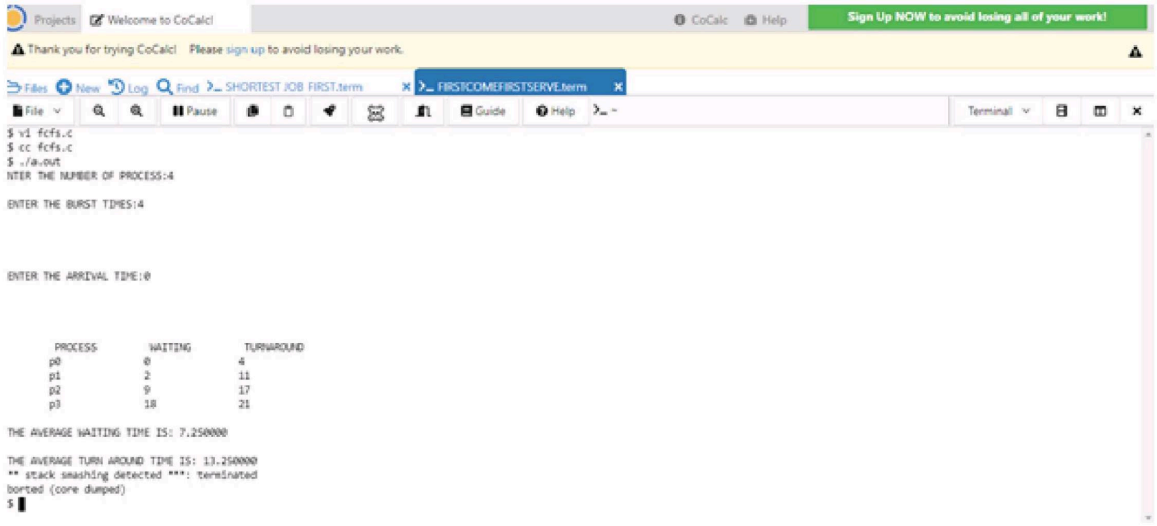
1. Start the process
2. Declare the array size
3. Get the number of processes to be inserted
4. Get the value
5. Start with the first process from its initial position and let the other process to be in queue6. Calculate the total number of burst time
7. Display the values
8. Stop the process

SOURCE CODE:

```
#include<stdio.h>
 main()
{
        int n,a[10],b[10],t[10],w[10],g[10],i,m;
        float att=0,awt=0;
        for(i=0;i<10;i++)
        {
                a[i]=0;
                b[i]=0;
                w[i]=0;
                 g[i]=0;
        }
        printf("enter the number of process");
        scanf("%d",&n);
        printf("enter the burst times");
        for(i=0;i<n;i++)
```

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


   printf("\nenter the arrival times");
for(i=0;i<n;i++)
   scanf("%d",&a[i]);
   g[0]=0;
 for(i=0;i<10;i++)
      g[i+1]=g[i]+b[i];
 for(i=0;i<n;i++)
 {
          w[i]=g[i]-a[i];
          t[i]=g[i+1]-a[i];
          awt=awt+w[i];
          att=att+t[i];
 }
 awt =awt/n;
 att=att/n;
 printf("\n\process\waiting time\turnaround time\n");
 for(i=0;i<n;i++)
 {
          printf("\tp%d\t\t%d\t\t%d\n",i,w[i],t[i]);
 }
 printf("the average waiting time is %f\n",awt); printf("the
average turn around time is %f\n",att); }
```

OUTPUT:



RESULT:

Thus the above program was created and executed successfully.

EX. NO. :5(b) IMPLEMENTATION OF SHORTEST JOB FIRST SCHEDULING ALGORITHM
DATE:

AIM:
To implement the shortest job first scheduling algorithm

ALGORITHM:
1. Start the process
2. Declare the array size
3. Get the number of elements to be inserted
4. Select the process which has the shortest burst will execute the first
5. If two processes have the same burst length then the FCFS scheduling algorithm uses
6. Make the average waiting the length of next process
7. Start with the first process from it's selection as above and let other process tobeinqueue
8. Calculate the total number of burst time
9. Display the values
10. Stop

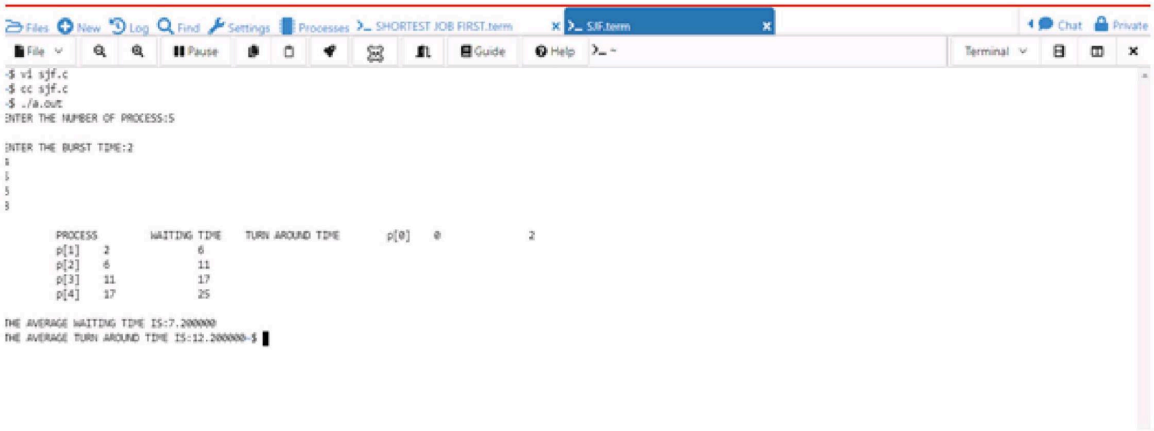
SOURCE CODE:

```
#include<stdio.h>

int main()

{

        int n,j,temp,temp1,pr[10],t[10],b[10],w[10],p[10],i;

        float att=0,awt=0;

        for(i=0;i<10;i++)

        {

                b[i]=0;w[i]=0;

        }

        printf("ENTER THE NUMBER OF PROCESS:");

        scanf("%d",&n);

        printf("\nENTER THE BURST TIME:");

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

        {

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

```
        p[i]=i;
}
for(i=0;i<n;i++)
{
        for(j=i;j<n;j++)
        {
                if(b[i]>b[j])
                {
                        temp=b[i];
                        temp1=p[i];
                        b[i]=b[j];
                        p[i]=p[j];
                        b[j]=temp;
                        p[j]=temp1;
                }
        }
}
w[0]=0;
for(i=0;i<n;i++)
{
        w[i+1]=w[i]+b[i];
}
for(i=0;i<n;i++)
{
        t[i]=w[i]+b[i];
```

```
            awt=awt+w[i];
            att=att+t[i];
        }
        awt=awt/n;
        att=att/n;
        printf("\n \t PROCESS \t WAITING TIME \t TURN AROUND  TIME");
        for(i=0;i<n;i++)
        {
            printf("\t p[%d]\t %d \t\t %d \n",p[i],w[i],t[i]);
        }
        printf("\nTHE AVERAGE WAITING TIME IS:%f",awt);
        printf("\nTHE AVERAGE TURNAROUND TIME IS:%f",att);
        return 0;
}
```

OUTPUT:



RESULT:

Thus the above program was created and executed successfully.

EX. NO. :5(c) IMPLEMENTATION OF ROUND ROBIN SCHEDULING ALGORITHM
DATE :

AIM:

To write a program to implement the round robin scheduling algorithm

ALGORITHM:

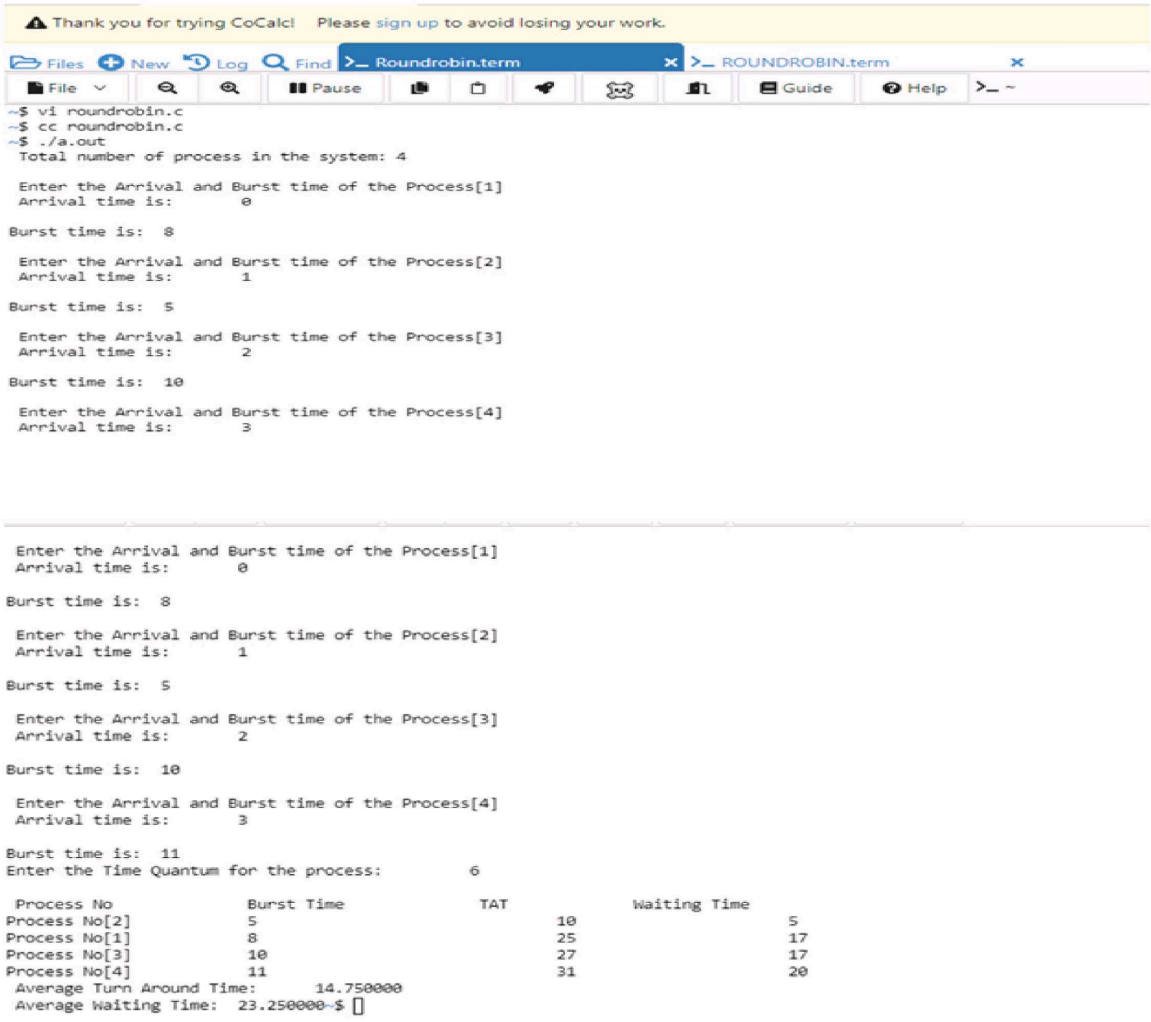
1. Start the process
2. Declare the array size
3. Get the number of elements to be inserted
4. Get the value
5. Set the time sharing system with preemption
6. Define quantum is defined from 10 to 100ms
7. Declare the queue as a circular
8. Make the CPU scheduler goes around the ready queue allocating CPUto each process For the time interval specified
9. Make the CPU scheduler picks the first process and sets time to interrupt after quantum expired dispatches the process
10. If the process have burst less than the time quantum than the process release theCPU
11. If the process have bust greater then time quantum then time will go off and cause interrupt to OS and the process put into the tail of ready queue and the schedule select next process
12. Display the results
13. Stop the process

SOURCE CODE:

```
#include<stdio.h>
int main()
{
        int n,j,temp,temp1,pr[10],t[10],b[10],w[10],p[10],i;
        float att=0,awt=0;
        for(i=0;i<10;i++)
        {
                b[i]=0;w[i]=0;
        }
        printf("ENTER THE NUMBER OF PROCESS:");
        scanf("%d",&n);
        printf("\nENTER THE BURST TIME:");
        for(i=0;i<n;i++)
        {
                scanf("%d",&b[i]);
                p[i]=i;
        }
        for(i=0;i<n;i++)
        {
                for(j=i;j<n;j++)
                {
```

```
                if(b[i]>b[j])
                {
                        temp=b[i];
                        temp1=p[i];
                        b[i]=b[j];
                        p[i]=p[j];
                        b[j]=temp;
                        p[j]=temp1;
                }
        }
    }
    w[0]=0;
    for(i=0;i<n;i++)
    {
        w[i+1]=w[i]+b[i];
    }
    for(i=0;i<n;i++)
    {
        t[i]=w[i]+b[i];
        awt=awt+w[i];
        att=att+t[i];
    }
    awt=awt/n;
    att=att/n;
    printf("\n \t PROCESS \t WAITING TIME \t TURN AROUND  TIME");
    for(i=0;i<n;i++)
    {
        printf("\t p[%d]\t %d \t\t %d \n",p[i],w[i],t[i]);
    }
    printf("\nTHE AVERAGE WAITING TIME IS:%f",awt);
    printf("\nTHE AVERAGE TURNAROUND TIME IS:%f",att);
    return 0;
}
```

OUTPUT:



RESULT:

Thus the above program was created and executed successfully.

EX.NO:5(d) PRIORITY SCHEDULING ALGORITHM

DATE:

AIM:

To write a program for the priority scheduling algorithm.

ALGORITHM:

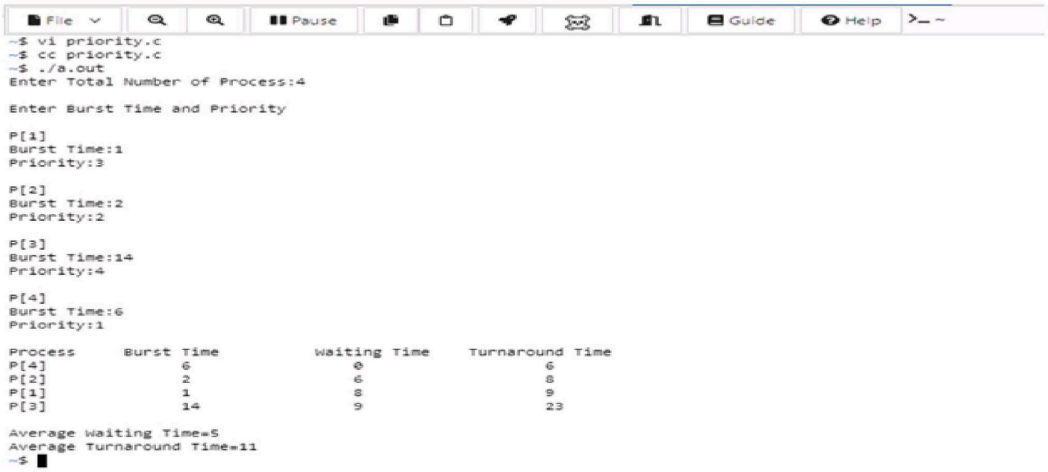
1. Define the process with the process id, process writing time pwt, priority-pr, burst time –by using structure.
2. Initialize the total waiting time (tt) to zero.
3. Read the number of process in the queue
4. Read the process id and process burst time for n jobs
5. Read the process priority
6. Arrange the process according to their priority assigned.

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
int main()
 {
  int x,n,p[10],pp[10],pt[10],w[10],t[10],awt,atat,i;
  printf("Enter the number of process : ");
  scanf("%d",&n);
  printf("\n Enter process : time priorities \n");
  for(i=0;i<n;i++)
   {
    printf("\nProcess no %d : ",i+1);
    scanf("%d  %d",&pt[i],&pp[i]);
    p[i]=i+1;
   }
  for(i=0;i<n-1;i++)
   {
    for(int j=i+1;j<n;j++)
    {
     if(pp[i]<pp[j])
      {
       x=pp[i];
       pp[i]=pp[j];
       pp[j]=x;
       x=pt[i];
       pt[i]=pt[j];
       pt[j]=x;
       x=p[i];
       p[i]=p[j];
       p[j]=x;
     }
```

```
   }
}
w[0]=0;
awt=0;
t[0]=pt[0];
atat=t[0];
for(i=1;i<n;i++)
 {
   w[i]=t[i-1];
   awt+=w[i];
   t[i]=w[i]+pt[i];
   atat+=t[i];
 }
printf("\n\n Job \t Burst Time \t Wait Time \t Turn  Around
 Time   Priority \n");
for(i=0;i<n;i++)
  printf("\n %d \t\t %d  \t\t %d \t\t %d \t\t %d
\n",p[i],pt[i],w[i],t[i],pp[i]);
awt/=n;
atat/=n;
printf("\n Average Wait Time : %d \n",awt);
printf("\n Average Turnaround Time : %d \n",atat);
return 0;
}
```

OUTPUT:



RESULT:

Thus the above program was created and executed successfully.

EX.NO:6 IMPLEMENTATION OF SEMAPHORES PRODUCER CONSUMER PROBLEM
DATE:

AIM:

To implement the producer consumer problem using semaphores.

ALGORITHM:

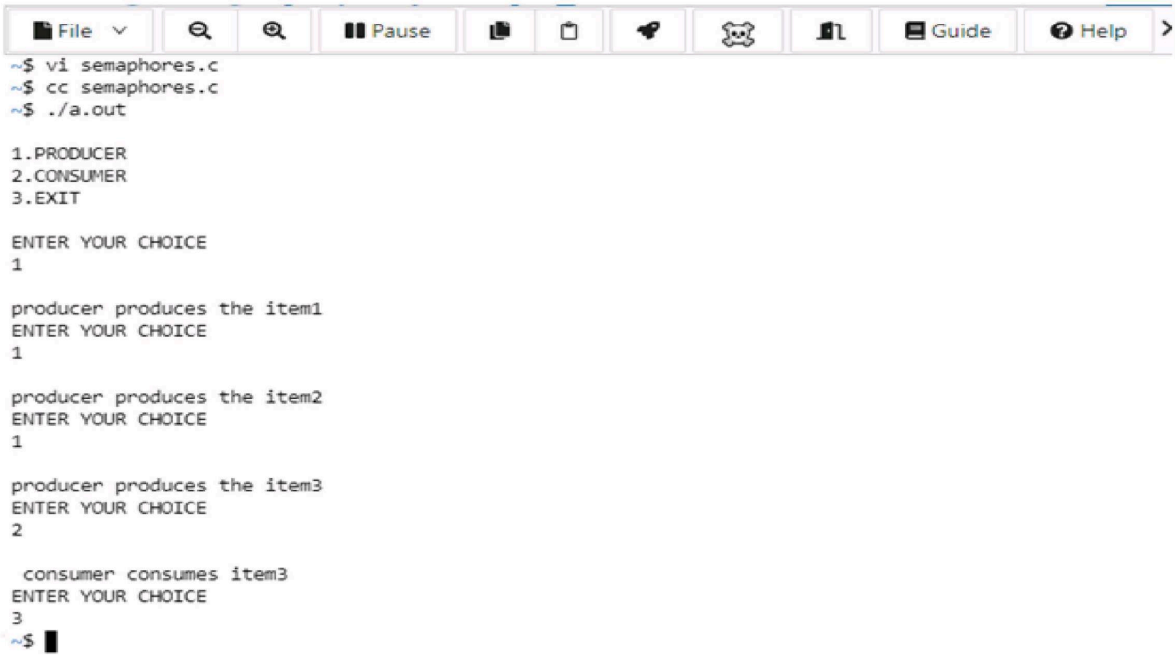
1.Start
2.Initialize buffer by using producer
3.Use two semaphore wait and signal transfer the content
4.Producer side performs the following
4.1:produce an item in steps
4.2:wait and add next item to buffer signal
5.End program.

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
int main()
{
int n;
void producer();
void consumer();
int wait(int);
int signal(int);
printf("\n1.PRODUCER\n2.CONSUMER\n3.EXIT\n");
while(1)
{
printf("\nENTER YOUR CHOICE\n");
scanf("%d",&n);
switch(n)
{
case 1:
if((mutex==1)&&(empty!=0))
producer();
else
printf("BUFFER IS FULL");
break;
case 2:
if((mutex==1)&&(full!=0))
consumer();
else
printf("BUFFER IS EMPTY");
break;
case 3:
exit(0);
```

```
break;
}
}
return 0;
}
int wait(int s)
{
return(--s);
}
int signal(int s)
{
return(++s);
}
void producer()
{
mutex=wait(mutex);
full=signal(full);
empty=wait(empty);
x++;
printf("\nproducer produces the item%d",x);
mutex=signal(mutex);
}
void consumer()
{
mutex=wait(mutex);
full=wait(full);
empty=signal(empty);
printf("\n consumer consumes item%d",x);
x--;
mutex=signal(mutex);
}
```

OUTPUT:



RESULT:

Thus the above program was created and executed successfully.

EX NO: 7 **INTERPROCESS COMMUNICATION**

DATE:

AIM

To write a program for inter process communication using pipes.

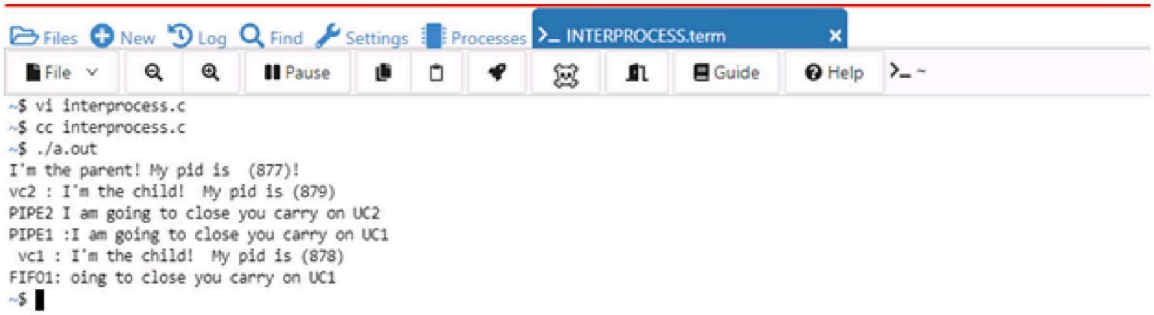
ALGORITHM

1. Create parent and child processes using fork().
2. Create the pipe using pipe() command.
3. Close the write end when read is performed.
4. Close the read end when write is performed.

SOURCE CODE:

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/types.h>
#include<sys/ipc.h>
main()
{
int fd[2],pid;
char *msg="UNIX WORLD\n";
char buff[32];
pipe(fd);
pid=fork();
if(pid==0)
{
printf("\n\t\t\t\tCHILD PROCESS");
printf("\n\t CHILD PROCESS:%d\n\t PIPE ID IS :%d",pid,fd[0]); close(fd[0]);
write(fd[1],msg,32);
}
else
{
sleep(1);
printf("\n\t PARENT PROCESS ID: %d\n",pid,fd[1]);
close(fd[1]);
read(fd[0],buff,32);
printf("\n\n\t THE MESSAGE IS RECEIVED FROM CHILD\n"); printf("\n\n THE
MESSAGE:%s",buff);
}
}
```

OUTPUT:



RESULT:

Thus the above program was created and executed successfully.

EX.NO:8 IMPLEMENTATION OF BANKER'S ALGORITHM FOR DEADLOCK AVOIDANCE
DATE:

AIM: To avoid deadlock using Banker‟s algorithm.

ALGORITHM:
Safety algorithm

1. Start
2. Initialize a temporary vector W (Work) to equal the available vector A. 3. Find an index i (row i) such that = W<Need i If no such row exists, the system will deadlock, since no process can run to completion.
4. If such a row is found, mark this process as finished, and add all its resources toWVector i, W = W + Ci Go to step 2, until either all processes are marked terminated(in this case initial state is safe), or until a deadlock occurs, in which the state is not safe.

Resource – request algorithm

1. If Request i < = Needi , go to step 2. Otherwise, error
2. If Request i < = Available, go to step 3. Otherwise, Pi must wait, since resources are not available
3. Modify the state (the system pretends to have allocated the requested resources to process Pi) Available = Available - Requesti Allocation = Allocation + Request Needi = Needi - Requesti 4. If the resulting state is safe, the transaction is completed and process Pi is allocated its resources. If the new state is unsafe, then I must wait for a Request and the old state is restored.
4. Stop.

SOURCE CODE:

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

int main() {
   int curr[5][5];
   int max_claim[5][5];
   int avl[5];
   int alloc[5] = {0, 0, 0, 0, 0};
   int max_res[5];
   int running[5];

   int i, j, exec, r, p;
   int count = 0;
   bool safe = false;

   printf("\nEnter the number of resources: ");
   scanf("%d", &r);

   printf("\nEnter the number of processes: ");
   scanf("%d", &p);
   for (i = 0; i < p; i++) {
      running[i] = 1;
      count++;
```

```
}

printf("\nEnter Claim Vector: "); for (i = 0; i < r; i++)
    scanf("%d", &max_res[i]);

printf("\nEnter Allocated Resource Table: "); for (i = 0; i <
p; i++) {
    for (j = 0; j < r; j++)
        scanf("%d", &curr[i][j]);
}

printf("\nEnter Maximum Claim table: "); for (i = 0; i < p; i++) {
    for (j = 0; j < r; j++)
        scanf("%d", &max_claim[i][j]);
}

printf("\nThe Claim Vector is: "); for (i = 0; i < r; i++)
    printf("%d ", max_res[i]);

printf("\nThe Allocated Resource Table:\n"); for (i = 0; i <
p; i++) {
    for (j = 0; j < r; j++)
        printf("\t%d", curr[i][j]);
    printf("\n");
}

printf("\nThe Maximum Claim Table:\n"); for (i = 0; i < p; i++) {
    for (j = 0; j < r; j++)
        printf("\t%d", max_claim[i][j]);
    printf("\n");
}

for (i = 0; i < p; i++)
    for (j = 0; j < r; j++)
        alloc[j] += curr[i][j];

printf("\nAllocated resources: ");
for (i = 0; i < r; i++)
    printf("%d ", alloc[i]);
for (i = 0; i < r; i++)
    avl[i] = max_res[i] - alloc[i];
printf("\nAvailable resources: ");
for (i = 0; i < r; i++)
    printf("%d ", avl[i]);
printf("\n");

while (count != 0) {
    safe = false;
    for (i = 0; i < p; i++) {
        if (running[i]) {
```

```
            exec = 1;
            for (j = 0; j < r; j++) {
               if (max_claim[i][j] - curr[i][j] > avl[j]) { exec = 0;
                  break;
               }
            }

            if (exec) {
               printf("\nProcess%d is executing.\n", i + 1); running[i] = 0;
               count--;
               safe = true;
               for (j = 0; j < r; j++)
                  avl[j] += curr[i][j];
               break;
            }
         }
      }

      if (!safe) {
         printf("\nThe processes are in unsafe state.");
         break;
      }

      if (safe)
         printf("\nThe process is in safe state.");

      printf("\nAvailable vector: ");
      for (i = 0; i < r; i++)
         printf("%d ", avl[i]);
   }

   return 0;
}
```

Output:

Enter the number of resources: 4
Enter the number of processes: 5

Enter Claim Vector: 8 5 9 7

Enter Allocated Resource Table: 2 0 1 1 0 1 2 1 4 0 0 3 0 2 1 0 1 0 3 0 Enter Maximum Claim

table: 3 2 1 4 0 2 5 2 5 1 0 5 1 5 3 0 3 0 3 3

The Claim Vector is: 8 5 9 7
The Allocated Resource Table:
2 0 1 1
0 1 2 1

4 0 0 3
0 2 1 0
1 0 3 0

The Maximum Claim Table:
3 2 1 4
0 2 5 2
5 1 0 5
1 5 3 0
3 0 3 3

Allocated resources: 7 3 7 5
Available resources: 1 2 2 2

Process3 is executing.

The process is in a safe state.
Available vector: 5 2 2 5
Process1 is executing.

The process is in a safe state.
Available vector: 7 2 3 6
Process2 is executing.

The process is in a safe state.
Available vector: 7 3 5 7
Process is executing.

The process is in a safe state.
Available vector: 7 5 6 7
Process is executing.
The process is in a safe state.
Available vector: 8 5 9 7

RESULT:
Thus the above program was created and executed successfully.

EX.NO:9 IMPLEMENTATION OF DEADLOCK DETECTION ALGORITHM
DATE:

AIM:
To Detect and Prevent Deadlock using Banker's Algorithm

ALGORITHM:

1. Mark each process that has a row in the Allocation matrix of all zeros. 2. Initialize a temporary vector**W** to equal the Available vector. 3. Find an index such that process*i* is currently unmarked and the ith row Q
   is less than or equal to **W** . That is,*Q ik … Wk*, for 1 *…k …m*. If no such row is found, terminate the algorithm.
4. If such a row is found, mark process*i* and add the corresponding row of the allocation matrix to **W** . That is, set*Wk = Wk* + A*ik*, for 1 *…k …m*. Return To step 3.

SOURCE CODE:

```
#include<stdio.h>
static int mark[20];
int i,j,np,nr;

int main()
{
int alloc[10][10],request[10][10],avail[10],r[10],w[10];

printf("\nEnter the no of process: ");
scanf("%d",&np);
printf("\nEnter the no of resources: ");
scanf("%d",&nr);
for(i=0;i<nr;i++)
{
printf("\nTotal Amount of the Resource R%d: ",i+1);
scanf("%d",&r[i]);
}



printf("\nEnter the request matrix:");

for(i=0;i<np;i++)
for(j=0;j<nr;j++)
scanf("%d",&request[i][j]);

printf("\nEnter the allocation matrix:");
for(i=0;i<np;i++)


for(j=0;j<nr;j++)
scanf("%d",&alloc[i][j]);
```

```
/*Available Resource calculation*/
for(j=0;j<nr;j++)
 {
avail[j]=r[j];
for(i=0;i<np;i++)
 {
avail[j]-=alloc[i][j];

}
}

//marking processes with zero allocation

for(i=0;i<np;i++)
 {
int count=0;
 for(j=0;j<nr;j++)
   {
     if(alloc[i][j]==0)
       count++;
     else
       break;
   }
 if(count==nr)
 mark[i]=1;
}
// initialize W with avail

for(j=0;j<nr;j++)
   w[j]=avail[j];

//mark processes with request less than or equal to W for(i=0;i<np;i++)
{
int canbeprocessed=0;
 if(mark[i]!=1)
 {
  for(j=0;j<nr;j++
{
     if(request[i][j]<=w[j])
       canbeprocessed=1;
     else
        {
        canbeprocessed=0;
        break;
         }
    }
if(canbeprocessed)


 {
```

```
mark[i]=1;

for(j=0;j<nr;j++)
w[j]+=alloc[i][j];
}
}
}
//checking for unmarked processes
int deadlock=0;
for(i=0;i<np;i++)
if(mark[i]!=1)
deadlock=1;

if(deadlock)
printf("\n Deadlock detected");
else
printf("\n No Deadlock possible");
}
```

OUTPUT:

Enter the no of process: 4
Enter the no of resources: 5
Total Amount of the Resource R1: 2 Total Amount of the Resource R2: 1 Total Amount of the Resource R3: 1 Total Amount of the Resource R4: 2 Total Amount of the Resource R5: 1 Enter the request matrix:
0 1 0 0 1
0 0 1 0 1
0 0 0 0 1
1 0 1 0 1
Enter the allocation matrix:1 0 1 1 0 1 1 0 0 0
0 0 0 1 0
0 0 0 0 0
Deadlock detected

RESULT:

Thus the above program was created and executed successfully.

EX.NO. 10 **THREADING & SYNCHRONIZATION**
DATE:

AIM:

To write a C program to implement Threading & Synchronization

ALGORITHM:

1. Start the Program
2. Initialize the process thread array.
3. Print the job started status.
4. Print the job finished status.
5. Start the main function
6. Check for the process creation if not print an error message.
7. Stop

SOURCE CODE:

```
#include<stdio.h>
#include <string.h>
#include<pthread.h>
#include <stdlib.h>
#include <unistd.h>
pthread_t tid[2];
int counter;
void* doSomeThing(void *arg)
{
unsigned long i = 0;
counter += 1;
printf("\n Job %d started\n", counter);
for(i=0; i< (0xFFFFFFFF);i++);
printf("\n Job %d finished\n", counter);
return NULL;
}
int main(void)
{
int i = 0;
int err;
while(i < 2)
{
err = pthread_create(&(tid[i]), NULL, &doSomeThing, NULL); if (err != 0)
printf ("\ncan't create thread :[%s]", strerror(err));
i++;
}
pthread_join(tid[0], NULL);
pthread_join(tid[1], NULL);
return 0;
}
```

OUTPUT:

Job 1 started
Job 1 finished
can't create thread :

RESULT:

Thus the above program was created and executed successfully.

EX.NO:11 IMPLEMENTATION OF MEMORY ALLOCATION METHODS
DATE:

AIM:

To implement the first fit memory management technique.

ALGORITHM:

1. Declare the structure memory with the process id, status start and end of the nodes and their difference.
2. Get the number of nodes and ending address of the node which defines the memory. 3. Select the option from the menu tot insert, delete, display or to exit. 4. If the insert option is selected, get the process id and size of the process. 5. The size of the process files in the memory, then print found, else display as no memory space.
6. Give the status of the process inserted to be 1.
7. If delete option is selected, then get the id of the process to be deleted and change the pid and status to 0.
8. If the display option is selected then display the process with their id, start and end of node, size and status.
9. Exit is chosen then stops the execution.

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
int pr[0],mempart[10],np,nb;
void firstfit();
void bestfit();
void worstfit();
main()
 {
printf("\nMemory Allocation Policy\n");
int i,j,ch;
printf("\nEnter number of Process\t");
scanf("%d",&np);
printf("Enter number of Partition blocks");
scanf("%d",&nb);
printf("\nProcess Information\n");
printf("------------------------\n");
for(i=0;i<np;i++)
 {
printf ("Enter mem required for process P%d = ",i+1);
scanf("%d", &pr[i]);
}
```

```
printf("\nMemory Partition Information");
printf("\n-------------------------\n");


for(j=0;j<nb;j++)
 {
printf("Enter Block size of Block B%d = ",j+1); scanf
("%d",&mempart[j]);
}
while (1)
 {
printf("1.First Fit");
printf("\n2.Best Fit");
printf("\n3.Worst Fit");
printf("\n4.Exit");
printf("\nEnter Your Choice");
 scanf("%d",&ch);
 switch(ch)
 {
case 1:
printf("\nFirst Fit");
printf("\n==============\n");
firstfit();
break;
case 2:
printf("\nBest Fit");
printf("\n==============\n");
bestfit();
break;
case 3:
printf("\nWorst Fit");
printf("\n==============\n");
worstfit();
break;
case 4:
exit(0);
}
printf("\nPress Enter to continue....");
getchar();
}
}
void firstfit()
 {
int i,j,st;
int mem[nb];
for(i=0;i<nb;i++)
 mem[i]=mempart[i];
printf("P/D\tPr.mem\tB/D\tBlock\tNew Block\n");
printf("-----------------------------------------------\n"); for(i=0;i<np;i++)
 {
```

```
st=0;
for(j=0;j<np;j++)
 {
if(pr[i]<=mem[j])
 {
printf("\n%d\t%d\t%d\t%d\n",i+1,pr[i],j+1,mem[j]);
mem[j]=mem[j]-pr[i];
printf("%d",mem[j]);
st=1;
break;
}
}
if(st==0)
printf("\n%d\t%d Insufficient memmory",i+1,pr[i]); }
}
void bestfit()
 {
int mem[nb];
int i,j,st,min,p;
for(i=0;i<nb;i++)
 mem[i]=mempart[i];
min=mem[0];
printf("P/O\tPr.mem\tB/D\tBlock\tNew\tBlock");
printf("-----------------------------------------------\n");
for(i=0;i<np;i++)
 {
st=0;
p=0;
min=mem[0];
for(j=0;j<nb;j++)
 {
if((mem[j]<min)&&(pr[i]<=mem[j]||pr[i]>min)) {min=mem[j];
p=j;
}
}
if(pr[i]<=mem[p])
 {
printf("\t%d\t%d\t%d\t%d\t",i+1,pr[i],p+1,mem[p]);
mem[p]=mem[p]-pr[i];
st=1;
printf("%d",mem[p]);
}
else if(st==0)
printf("%d\t%d Insufficient Memory",i+1,pr[i]); }
}
void worstfit()
 {
int mem[nb];
int i,j,st,max,p;
for(i=0;i<nb;i++)
```

```
 mem[i]=mempart[i];
max=mem[0];
printf("P/D\tPr.mem\tB/D\Block\tNew Block");
printf("\n----------\n"); for(i=0;i<np;i++)
 {
st=0;
p=0;
max=mem[0];
for(j=0;j<nb;j++)
 {
if((mem[j]>max)&&(pr[j]<=mem[j]))
 {
max=mem[j];
p=j;
}
}
if(pr[i]<=mem[p])
 {
printf("\n%d\t%d\t%d\t%d\t",i+1,pr[i],p+1,mem[p]);
mem[p]=mem[p]-pr[i];
st=1;
printf("%d",mem[p]);
}
else if(st==0)
printf("\n%d\t%d Insufficient memory",i+1,pr[i]); }
}
```

OUTPUT:

Memory Allocation Policy
Enter number of Process 4
Enter number of Partition blocks4 Process Information

Enter mem required for process P1 = 500 Enter mem required for process P2 = 50 Enter mem required for process P3 = 110 Enter mem required for process P4 = 300

Memory Partition Information

Enter Block size of Block B1 = 100 Enter Block size of Block B2 = 200 Enter Block size of Block B3 = 150

Enter Block size of Block B4 = 600

1.First Fit
2.Best Fit
3.Worst Fit
4.Exit
Enter Your Choice1

First Fit

P/D Pr.mem B/D Block New Block
---
1 500 4 600
100
2 50 1 100
50
3 110 2 200
90
4 300 Insufficient memory

Press Enter to continue....
1. First fit
2. Best Fit
3. Worst Fit
4. Exit
Enter Your Choice 2

Best Fit
P/O Pr.mem B/D Block New Block

1.500 Insufficient Memory 2 .50 1 100 50 3.110 Insufficient Memory 4.300 Insufficient Memory
Press Enter to continue....1.First fit
2. Best Fit
3. Worst Fit
4. Exit
Enter Your Choice3

Worst Fit

P/D Pr.mem B/D Block New Block
---

1 500 4 600 100
2 50 2 200 150
3 110 2 150 40
4 300 Insufficient memory

Press Enter to continue....
1. First fit
2. Best Fit
3. Worst Fit
4. Exit
Enter Your Choice4

RESULT:

Thus the above program was created and executed successfully.

EX. NO: 12 IMPLEMENTATION OF PAGING TECHNIQUES
DATE:

AIM:

To implement memory allocation with pages

ALGORITHM:

1. Start
2. Declare the structure P_table with variables for page no and frame no
3. Display the status of physical memory
4. Get the number of pages needed for a process
5. Get the contents of the pages
6. Display the contents of logical memory
7. If the physical memory is available then allot the pages of the process
8. Update the physical memory status
9. Update the page table status
10. Display the page table after allocation
11. Display the physical memory after allocation
12. Stop

SOURCE CODE:

```
#include<stdio.h>
void main()
{
int memsize=15;
int pagesize,nofpage;
int p[100];
int frameno,offset;
int logadd,phyadd;
int i;
int choice=0;
printf("\nYour memsize is %d ",memsize);
printf("\nEnter page size:");
scanf("%d",&pagesize);

nofpage=memsize/pagesize;

for(i=0;i<nofpage;i++)
{
printf("\nEnter the frame of page%d:",i+1);
scanf("%d",&p[i]);
}

do
{
printf("\nEnter a logical address:");
scanf("%d",&logadd);
frameno=logadd/pagesize;
offset=logadd%pagesize;
phyadd=(p[frameno]*pagesize)+offset;
```

```
printf("\nPhysical address is:%d",phyadd); printf("\nDo
you want to continue(1/0)?:"); scanf("%d",&choice);
}while(choice==1);
}
```

OUTPUT:

Your memsize is 15
Enter page size:5

Enter the frame of page1:2

Enter the frame of page2:4

Enter the frame of page3:7

Enter a logical address:3

Physical address is:13
Do you want to continue(1/0)?:1 Enter a logical

address:1

\

RESULT:

Thus the above program was created and executed successfully.

Ex.No:13(a) IMPLEMENTATION OF FIFO PAGE REPLACEMENT ALGORITHM

DATE:

AIM

To write a program to implement FIFO page replacement algorithms.

ALGORITHM

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

SOURCE CODE:

```
#include<stdio.h>
int main()
{
        int i,j,n,a[50],frame[10],no,k,avail,count=0;
        printf("\n ENTER THE NUMBER OF PAGES:\n");
        scanf("%d",&n);
        printf("\n ENTER THE PAGE NUMBER :\n");
        for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
        printf("\n ENTER THE NUMBER OF FRAMES :");
        scanf("%d",&no);
        for(i=0;i<no;i++)
        frame[i]= -1;
                j=0;
                printf("\tref string\t page frames\n");
        for(i=1;i<=n;i++)
                {
                        printf("%d\t\t",a[i]);
                        avail=0;
                        for(k=0;k<no;k++)
                        if(frame[k]==a[i])
                                avail=1;
                        if (avail==0)
                        {
                                frame[j]=a[i];
                                j=(j+1)%no;
                                count++;
```

```
                        for(k=0;k<no;k++)
                        printf("%d\t",frame[k]);
                }
                printf("\n");
        }
        printf("Page Fault Is %d",count);
        return 0;
}
```

OUTPUT:

ENTER THE NUMBER OF PAGES: 20

ENTER THE PAGE NUMBER : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 ENTER THE NUMBER OF FRAMES :3

```
ref string page frames
   7 7 -1 -1
   0 7 0 -1
   1 7 0 1
   2 2 0 1
   0
   3 2 3 1
   0 2 3 0
   4 4 3 0
   2 4 2 0
   3 4 2 3
   0 0 2 3
   3
   2
   1 0 1 3
   2 0 1 2
   0
   1
   7 7 1 2
   0 7 0 2
   1 7 0 1

   Page Fault Is 15
```

RESULT:

Thus the above program was created and executed successfully.

EX.NO:13(b) IMPLEMENTATION OF LRU PAGE REPLACEMENT ALGORITHM
DATE:

AIM:

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

ALGORITHM :

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

SOURCE CODE:

```
#include<stdio.h>
main()
{
        int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];
        printf("Enter no of pages:");
        scanf("%d",&n);
        printf("Enter the reference string:");
        for(i=0;i<n;i++)
        scanf("%d",&p[i]);
        printf("Enter no of frames:");
        scanf("%d",&f);
        q[k]=p[k];
        printf("\n\t%d\n",q[k]);
        c++;
        k++;
        for(i=1;i<n;i++)
        {
                c1=0;
                for(j=0;j<f;j++)
                {
                        if(p[i]!=q[j])
                        c1++;
                }
                if(c1==f)
                {
                        c++;
                        if(k<f)
                        {
                                q[k]=p[i];
```

```
                        k++;
                        for(j=0;j<k;j++)
                        printf("\t%d",q[j]);
                        printf("\n");
                }
                else
                {
                        for(r=0;r<f;r++)
                        {
                                c2[r]=0;
                                                for(j=i-1;j<n;j--)
                                {
                                if(q[r]!=p[j])
                                c2[r]++;
                                else
                                break;
                        } }
                for(r=0;r<f;r++)
                 b[r]=c2[r];
                for(r=0;r<f;r++)
                {
                        for(j=r;j<f;j++)
                        {
                                if(b[r]<b[j])
                                {
                                                        t=b[r];
                                                        b[r]=b[j];
                                                        b[j]=t;
                                                } } }
                for(r=0;r<f;r++)
                {
                        if(c2[r]==b[0])
                        q[r]=p[i];
                        printf("\t%d",q[r]);
                }
                printf("\n");
        } } }
printf("\nThe no of page faults is %d",c);

}
```

OUTPUT

Enter no of pages:10

Enter the reference string:7 5 9 4 3 7 9 6 2 1 Enter no of frames:3

7

7 5

7 5 9

4 5 9

4 3 9

4 3 7

9 3 7

9 6 7

9 6 2

1 6 2

The no of page faults is 10

RESULT:

Thus the above program was created and executed successfully.

EX.NO:14 IMPLEMENTATION OF THE VARIOUS FILE ORGANIZATION
DATE: TECHNIQUES.

AIM:

Write a C program to simulate the following file organization techniques a) Single level directory b) Two level directory c) Hierarchical

ALGORITHM:
SINGLE LEVEL DIRECTORY ORGANIZATION:

1. Start
2. Declare the number, names and size of the directories and file names. 3. Get the values for the declared variables.
4. Display the files that are available in the directories.
5. Stop

TWO LEVEL DIRECTORY ORGANIZATION:

1. Start
2. Declare the number, names and size of the directories and subdirectories and filenames.
3. Get the values for the declared variables.
4. Display the files that are available in the directories and subdirectories. 5. Stop

HIERARCHICAL DIRECTORY ORGANIZATION:

1. Start
2. Declare the number, names and size of the directories and subdirectories and filenames.
3. Get the values for the declared variables.
4. Display the files that are available in the directories and subdirectories. **5.** Stop.

SOURCE CODE:

1. SINGLE LEVEL DIRECTORY ORGANIZATION

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir;
void main()
{
int i,ch;
char f[30];
clrscr();
dir.fcnt = 0;
printf("\nEnter name of directory -- ");
scanf("%s", dir.dname);
while(1)
```

```
{
printf("\n\n 1. Create File\t2. Delete File\t3. Search File \n 4. Display Files\t5. Exit\nEnter your choice --
");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\n Enter the name of the file -- ");
scanf("%s",dir.fname[dir.fcnt]);
dir.fcnt++;
break;
case 2: printf("\n Enter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is deleted ",f);
strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);
break;
}
}
if(i==dir.fcnt)
printf("File %s not found",f);
else
dir.fcnt--;
break;
case 3: printf("\n Enter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is found ", f);
break;
}
}
if(i==dir.fcnt)
printf("File %s not found",f);
break;
case 4: if(dir.fcnt==0)
printf("\n Directory Empty");
else
{
printf("\n The Files are -- ");
for(i=0;i<dir.fcnt;i++)
```



```
printf("\t%s",dir.fname[i]);
}
break;
```

```
default: exit(0);
}
}
getch();
}
```

OUTPUT:

Enter name of directory -- CSE

1. Create File 2. Delete File 3. Search File 4. Display Files 5. Exit
Enter your choice – 1

Enter the name of the file -- A

1. Create File 2. Delete File 3. Search File 4. Display Files 5. Exit
Enter your choice – 1

Enter the name of the file -- B

1. Create File 2. Delete File 3. Search File 4. Display Files 5. Exit
Enter your choice – 1

Enter the name of the file -- C

1. Create File 2. Delete File 3. Search File 4. Display Files 5. Exit
Enter your choice – 4

The Files are -- A B C

1. Create File 2. Delete File 3. Search File 4. Display Files 5. Exit
Enter your choice – 3 Enter the name of the file – ABC File ABC not found
1. Create File 2. Delete File 3. Search File 4. Display Files 5. Exit
Enter your choice – 2 Enter the name of the file – B File B is deleted
1. Create File 2. Delete File 3. Search File 4. Display Files 5. Exit
Enter your choice – 5

2. TWO LEVEL DIRECTORY ORGANIZATION

```
#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];
clrscr();
```

```
dcnt=0;
while(1)
 {
printf("\n\n 1. Create Directory\t 2. Create File\t 3. Delete File"); printf("\n 4. Search File
\t \t 5. Display \t 6. Exit \t Enter your choice -- "); scanf("%d",&ch);
 switch(ch)
 {
case 1: printf("\n Enter name of directory -- ");
 scanf("%s", dir[dcnt].dname);
dir[dcnt].fcnt=0;
dcnt++;
printf("Directory created");
break;
case 2: printf("\n Enter 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].fcnt]);
dir[i].fcnt++;
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]); goto jmp;
}
}
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].dname);
for(k=0;k<dir[i].fcnt;k++)
printf("\t%s",dir[i].fname[k]);
}
}
break;
default:exit(0);
}
}getch();
}
```

OUTPUT:

```
 1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display 6.
Exit Enter your choice -- 1

Enter name of directory -- DIR1
Directory created


 1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display 6.
Exit Enter your choice -- 1
```

Enter name of directory -- DIR2

Directory created

1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display 6. Exit Enter your choice -- 2

Enter name of the directory – DIR1 Enter name of the file
-- A1
File created

1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display 6. Exit Enter your choice -- 2

Enter name of the directory – DIR1 Enter name of the file
-- A2
File created

1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display 6. Exit Enter your choice -- 2

Enter name of the directory – DIR2 Enter name of the file
-- B1
File created

1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display 6. Exit Enter your choice -- 5 Directory Files
DIR1 A1 A2
DIR2 B1

1. Create Directory 2. Create File 3. Delete File 4. Search File 5. Display 6. Exit Enter your choice -- 4
Enter name of the directory – DIR
Directory not found

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 3 Enter name of the directory – DIR1
Enter name of the file -- A2

File A2 is deleted

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice – 6

3. HIERARCHICAL DIRECTORY ORGANIZATION

```
#include<stdio.h>
#include<graphics.h>
struct tree_element
{
char name[20];
int x,y,ftype,lx,rx,nc,level;
struct tree_element *link[5];
};
typedef struct tree_element
node; void main()
{
int gd=DETECT,gm;
node *root;
root=NULL;
clrscr();
create(&root,0,"root",0,639,320);
clrscr();
initgraph(&gd,&gm,"c:\\tc\\BGI");
display(root);
getch();
closegraph();
}
create(node **root,int lev,char *dname,int lx,int rx,int x)
{
int i,gap;
if(*root==NULL)
{
(*root)=(node *)malloc(sizeof(node));
printf("Enter name of dir/file(under %s) :",dname);
fflush(stdin);
gets((*root)->name);
printf("enter 1 for Dir/2 forfile :");
scanf("%d",&(*root)->ftype);
(*root)->level=lev;
(*root)->y=50+lev*50;
(*root)->x=x;
(*root)->lx=lx;
(*root)->rx=rx;
for(i=0;i<5;i++)
(*root)->link[i]=NULL;
if((*root)->ftype==1)
{
printf("No of sub directories/files(for %s):",(*root)->name); scanf("%d",&(*root)->nc);
if((*root)->nc==0)
gap=rx-lx;
```

```
else gap=(rx-lx)/(*root)->nc;
for(i=0;i<(*root)->nc;i++)
create(&((*root)->link[i]),lev+1,(*root)->name,lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2); }
else (*root)->nc=0;
}
}
display(node *root)
{
int i;
settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
setcolor(14); if(root!=NULL)
{
for(i=0;i<root->nc;i++)
{
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
}
if(root->ftype==1) bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0); else
fillellipse(root->x,root->y,20,20);
outtextxy(root->x,root->y,root->name); for(i=0;i<root->nc;i++) {
display(root->link[i]);
}
}
}
```

OUTPUT:

```
Enter Name of dir/file (under root): ROOT Enter 1 for Dir / 2 For File :
1
No of subdirectories / files (for ROOT) :2 Enter Name of dir/file (under
ROOT):USER 1 Enter 1 for Dir /2 for file:1
No of subdirectories /files (for USER 1):1 Enter Name of dir/file (under USER
1):SUBDIR Enter 1 for Dir /2 for file:1
No of subdirectories /files (for SUBDIR):2 Enter Name of dir/file
(under USER 1): JAVA Enter 1 for Dir /2 for file:1
No of subdirectories /files (for JAVA): 0 Enter Name of dir/file (under
SUBDIR):VB Enter 1 for Dir /2 for file:1
No of subdirectories /files (for VB): 0

Enter Name of dir/file (under ROOT):USER2 Enter 1 for Dir /2 for
file:1
No of subdirectories /files (for USER2):2 Enter Name of dir/file (under
ROOT):A Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under USER2):SUBDIR 2 Enter 1 for Dir /2 for file:1
No of subdirectories /files (for SUBDIR 2):2

Enter Name of dir/file (under SUBDIR2):PPL Enter 1 for Dir /2 for file:1
```

No of subdirectories /files (for PPL):2 Enter Name of dir/file (under PPL):B Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under PPL):C Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under SUBDIR):AI
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for AI): 2 Enter Name of dir/file (under AI):D Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under AI):E Enter 1 for Dir /2 for file

RESULT:

Thus the above program was created and executed successfully.

EX.NO:15(a) IMPLEMENTATION OF CONTIGUOUS FILE ALLOCATION
DATE:

AIM:

To write a program to implement contiguous file allocation.

ALGORITHM:

1. Start the process.
2. Define linear ordering on the disk.
3..Assume one job in the disk.
4. For sequential access, the file system remembers the disk address of the last block referenced.
5. Display the results.
6. Stop the process.

SOURCE CODE:

```
#include<stdlib.h>
#include<string.h>
struct block
 {
int b_id;
int b_alloted;
};
struct block b[50];
int main()
 {
int i,n,n0,bname,sblock,j,count=0,cblock;
int psize,flag=1,pname;
printf("\n Enter the no of blocks");
scanf("%d",&n);
for(i=0;i<=n;i++)
 {
b[i].b_id=i;
b[i].b_alloted=0;
 }
printf("Enter the no of block already alloted");
scanf("%d",&n0);
for(i=0;i<n0;i++)
 {
printf("\n Enter the block");
scanf("%d",&bname);
b[bname].b_alloted=100;
 }
printf("\n Enter process name \n");
scanf("%d",&pname);
printf("Enter process size");
```

```
scanf("%d",&psize);
for(i=0;i<n;i++)
 {
if(b[i].b_alloted==0)
 {
count=count+1;
if(count==1)
 {
sblock=i;
}
if(count==psize)
 {
cblock=i;
for(j=sblock;j<=cblock;j++)
b[j].b_alloted=pname;
printf("\n\n Process %d is alloted blocks from %d to %d \n \n",pname,sblock,cblock); i=n+1;
flag=1;
}
}
else
 {
count=0;
flag=0;
}
}
if(flag==0)
printf("\n Process not allocated \n");
count=0;
for(i=0;i<n;i++)
 {
count=count+1;
if(count<4)
printf("b[%d]-- %d \t \t",b[i].b_id,b[i].b_alloted);
else
 {
count=0;
printf("\n\n\n b[%d]--%d\t\t",b[i].b_id,b[i].b_alloted);
count=count+1;
}
}
return 0;
}
```

OUTPUT:

```
[44208104008@localhost oslab]$ cc contiguous.c
[44208104008@localhost oslab]$ ./a.out Enter the no of blocks3
Enter the no of block already allotted 2 Enter the block1
 Enter the block2
 Enter process name
5
Enter process size
2
 Process not allocated
b[0]-- 0 b[1]-- 100 b[2]—100
```

RESULT:

Thus the above program was created and executed successfully.

EX.NO :15(b) IMPLEMENTATION OF LINKED FILE ALLOCATION TECHNIQUE
DATE:

AIM:

To write a program to implement linked file allocation.

ALGORITHM:

1. Start the process.
2. Declare the pointer.
3. Declare the values.
4. Perform the process.
5. Display the result.
6. Stop the process.

SOURCE CODE:

```
# include <stdio.h>
# include <string.h>
struct block
{
int b_id;
char b_allocated;
};
struct block b[50];
void main()
{
 int i,n0,n,bname,j_count=0,larr[90],cnt=0,k=0;
int psize,flag=1,flg=0,flg2=0;
char pname;
printf("\n Enter no of blocks");
scanf("%d",&n);
for(i=0;i<n;i++)
{
b[i].b_id=i;
b[i].b_allocated='-';
}
do
{
if(flg==1)
{
printf("Not Enough blocks in total ..Re enter\n");
flg=0;
}
printf("Enter no of block allocated");
scanf("%d",&n0);
if(n0>n)
flg=1;
```

```
}
while(flg=1);
for(i=0;i<n0;i++)
{
printf("Enter the block no"); do
{
if(flg2==1)
{
printf("Block no does not exist..Re-enter\n"); flg2=0;
}
scanf("%d",&bname);
if(bname>=n)
flg2=1;
}
while(flg2==1);
b[bname].b_allocated='z';
}
printf("\n Process name is A"); pname='A';
printf("\n Enter process size");
scanf("%d",&psize);
for(i=0;i<n;i++)
{
if(b[i].b_allocated=='-')
cnt++;
}
if(cnt>=psize)
{
for(i=0;i<n;i++)
{
if(b[i].b_allocated=='-')
{
larr[k]=i;
b[i].b_allocated=pname;
k++;
}
if(k==psize)
break;
}
printf("Linkage structure of blocks \n");
for(i=0;i<k;i++)
{
printf("%d --->",larr[i]);
}
printf("\n");
count++;
for(i=0;i<n;i++)
{
count=count+1;
if(count<4)
printf("b[%d]---%c \t",b[i].b_id,b[i].b_allocated); else
```

```
{
count=0;
printf("\n b[%d] -- %c\t",b[i].b_id,b[i].b_allocated);
count=count+1;
}}}
else
{
printf("NO space to allocated\n");
}
printf("\n");
}
```

OUTPUT:

```
Enter No.of blocks 3
Enter No. of blocks allocate 1
Enter block no 2
Process Name is A
Enter process size 2
Linkage structure of blocks
0 ---> 1 --->
b[0]--A b[1]--A
b[2]--Z
```

RESULT:

Thus the above program was created and executed successfully.

EX.NO: 15(C) IMPLEMENTATION OF INDEX FILE ALLOCATION TECHNIQUE
DATE:

AIM:

To write a program to implement index file allocation technique

ALGORITHM:

1. Start the process
2. Enter number of blocks
3. Check if flag=1,then print total blocks are not enough and re-enter 4. Display number of blocks allocated to the process
5. Enter the block number that is allocated to process
6. If flag2=1,print the block number does not exit
7. Display process name and process size
8. Check if cnt>psize,initialize larr[k]=I and compute b[i].b_allocated=pname 9. Print the linkage structure of blocks,directory structure and process index 10. Else print there is no space to allocate
11. Stop the process

SOURCE CODE:

```
# include <stdio.h>
# include <string.h>
struct block
 {
int b_id;
char b_allocated;
};
struct block b[50];
void main()
 {
 int i,n0,n,bname,j_count=0,larr[90],cnt=0,k=0;
int psize,flag=1,flg=0,flg2=0;
char pname;
printf("\n Enter no of blocks");
scanf("%d",&n);
for(i=0;i<n;i++)
 {
b[i].b_id=i;
b[i].b_allocated='-';
 }
do
 {
if(flg==1)
```

```
{
printf("Not Enough blocks in total ..Re enter\n"); flg=0;
}
printf("Enter no of block allocated");
scanf("%d",&n0);
if(n0>n)
flg=1;
}
while(flg=1);
for(i=0;i<n0;i++)
{
printf("Enter the block no");
do
{
if(flg2==1)
{
printf("Block no does not exits..Re-enter\n"); flg2=0;
}
scanf("%d",&bname);
if(bname>=n)
flg2=1;
}
while(flg2==1);
b[bname].b_allocated='z';
}
printf("\n Process name is A");
pname='A';
printf("\n Enter process size");
scanf("%d",&psize);
for(i=0;i<n;i++)
{
if(b[i].b_allocated=='-')
cnt++;
}
if(cnt>=psize)
{
for(i=0;i<n;i++)
{
if(b[i].b_allocated=='-')
{
larr[k]=i;
b[i].b_allocated=pname;
k++;
}
if(k==psize)
{
end=i;
break;
}}
printf("Linkage structure of blocks \n");
```

```
for(i=0;i<k;i++)
 {
printf("%d --->".larr[i]);
 }
printf("\n");
count++;
for(i=end;i<n;i++)
 {
if(b[i].b_allocated=='-')
 {
b[i].b_allocated='#';
end=i;
break;
}}
for(i=0;i<n;i++)
 {
count=count+1;
if(count<4)
printf("b[%d]-- %c \t",b[i].b_id,b[i].b_allocated); else
 {
count=0;
printf("\n b[%d]-- %c \t",b[i].b_id,b[i].b_allocated); count=count+1;
}}
printf("\n Directory Structure \n Process \t Index \n"); printf("%d \t %c
\n",b[end].b_id,b[end].b_allocated); }
else
 {
printf("No space to allocate \n");
 }
printf("\n");
 }
```

OUTPUT:

```
Enter no.of Block 8
Enter no. of blocks allocated 2
Enter Block no 2
Enter block no 4
Process Name is A
Enter process size 3
Linkage Structure of Blocks
0---> 1--->3 --->
b[0]--A b[i]--A
b[2]--Z b[3]--A b[4]==z
b[5]==# b[6]-- b[7]--
Directory structure
Process Index
5#
```

RESULT:

Thus the above program was created and executed successfully.