Dr. M.G.R.
EDUCATIONAL AND RESEARCH INSTITUTE
DEEMED TO BE UNIVERSITY
University with Graded Autonomy Status
(An ISO 21001 : 2018 Certified Institution)
Periyar E.V.R. High Road, Maduravoyal, Chennai-95. Tamilnadu, India.

RECORD NOTEBOOK

BCS18L06- OPERATING SYSTEMS LAB

2023-2024(ODD SEMESTER)

DEPARTMENT

Of

COMPUTER SCIENCE AND ENGINEERING

NAME : **HARICHSELVAM**
REGISTER NO : **211191101045**
COURSE : **B.TECH CSE-DS&AI**
YEAR/SEM/SEC : **III / V / A**

Dr. M.G.R.
EDUCATIONAL AND RESEARCH INSTITUTE
DEEMED TO BE UNIVERSITY
University with Graded Autonomy Status
(An ISO 21001 : 2018 Certified Institution)
Periyar E.V.R. High Road, Maduravoyal, Chennai-95. Tamilnadu, India.

BONAFIDE CERTIFICATE

Register No: **211191101045**

Name of Lab: **OPERATING SYSTEMS LAB(BCS18L06)**

Department: **COMPUTER SCIENCE AND ENGINEERING**

Certified that this is the bonafide record of work done by **HARICHSELVAM** of III Year B.Tech. (CSE – DS & AI),Sec-A in the **OPERATING SYSTEMS LAB** during the year 2023-2024.

Signature of Lab-in-Charge **Signature of Head of Dept**

Submitted for the Practical Examination held on ----------

Internal Examiner **External Examiner**

TABLE OF CONTEXT

Ex.No:1 **UNIX INTRODUCTION**

DATE:

AIM:

To study the basics of UNIX operating system, commands and vi editor

UNIX Commands

General Commands:

1. **date**: This tells the current date and time.
   $ date
   Thu Oct 15 09:34:50 PST 2005

2. **who**: Gives the details of the user who have logged into the system.
   $ who
   abc tty() oct 15 11:17
   Xyz tty4 oct 15 11:30

3. **whoami**: Gives the details regarding the login time and system's name for the connection being used.
   $ whoami.
   Raghu

4. **man**: It displays the manual page of our terminal with the command 'man' command name.
   $ man who

5. head and tail: 'head' is used to display the initial part of the text file and 'tail' is used to display the last part of the file.
$ head [-count] [filename]
$ tail [-count] [filename]

6. pwd: It displays the full path name for the current directory we are working in.
$ pwd
/home/raghu

7. is : It displays the list of files in a current working directory.
$ ls
$ ls -l = lists files in long format.

$ ls –t = lists in order of last modification time.
$ ls –a = lists all entries , including the hidden files.
$ ls –d = lists directory files instead of its contents.
$ ls –p = puts a slash after each directory.
$ ls –u = lists in order of last access time.

8. mkdir: It is used to create a new directory.

$ mkdir directory name.

9. cd : It is used to change from the working directory to any other directory specified.

$ cd changes to home directory.
$ cd.. changes to parent directory.
$ cd / changes to root directory.
$ cd dirl changes to directory dirl.

10. rmdir: It is use to remove the directory specified in the command line.

$ rmdir directory name

11.cat : This command helps us to specify the contents of the file we specify.

$ cat [option….[file….]]

12. cp : This command is used to create duplicate copied of ordinary files.

$ cp file target
$ cp file1 file2 [file1 is copied to file2]

13.mv: This command is used to rename and move ordinary and directory files.

$ mv file1 file2
$ mv directory1 directory2

14. ln: This is used to link file.

$ ln file1 [file2….] target

15. rm: This command is used to remove one or more files from the directory.This can be used to delete all files as well as directory.

$ rm [option…..] file

16. chmod: Change the access permissions of a file or a directory.

$ chmod mode file
$ chmod [who] [+/-/=] [permission…] file

who = a – all users
g – group
o – others
u –user
[+/-/=] + adds
- removes
= assigns

[permission] r = read
w =write
x =execute
Ex.:: $ chmod 754 prog1.

17. chown: change the owner ID of the files or directories.
Owner may be decimal user ID or a login name found in the file/etc/passwd.
This utility is governed by the chown kernel authorization. If it is not granted,ownership can only be changed by root.
Ex:: $ chown tutor test

18. wc:counts and displays the lines,words and characters in the files specified.
$ wc
$ wc
$ wc
$ wc
Ex:: $ wc prog2.
3 9 60 prog2.

19. grep : searches the file for the pattern.
$ grep [option…] pattern [file….].
Display the lines containing the pattern on the standard output.
$ grep c report only the number of matching lines.
$ grep-l list only the names of files containing pattern.
$grep-v display all lines expert those containing pattern.

Ex: grep c”the”prog2.

20. cut: cuts out selected fields of each line of a file.
$ cut -first [-d char] [file1 file2…].
-d = it is delimiter. Default is tab.

Ex: cut –f1, 3 –d”w”prog2.

21. **paste**: merges the corresponding lines of the given files.

$ paste –d file1 file2

Option – d allows replacing tab character by one or more alternate characters.

Ex:: paste prog1 prog2

22. sort : arranges lines in alphabetic or numeric order.

$ sort [option]file.

Option –d dictionary order

Option –n arithmetic order

Option –r reverse order

Ex :: $ ls – l | sort –n

The Vi Editor

The vi editor is a line-oriented editor and is not very easy to use. But it is simple and you can learn enough commands to use it for editing your Java programs.

Command mode and edit mode

In vi, you will need to shift from command mode to edit mode and back again.

You need to be in command mode to move the cursor around from one place to another.

Esc-Shift to command mode

You need to be in edit mode to input or change text. You will automatically shift to the edit mode when you enter a command that affects text . You then need to hit Escape to go back to the command mode or else whatever you type will be taken as the input

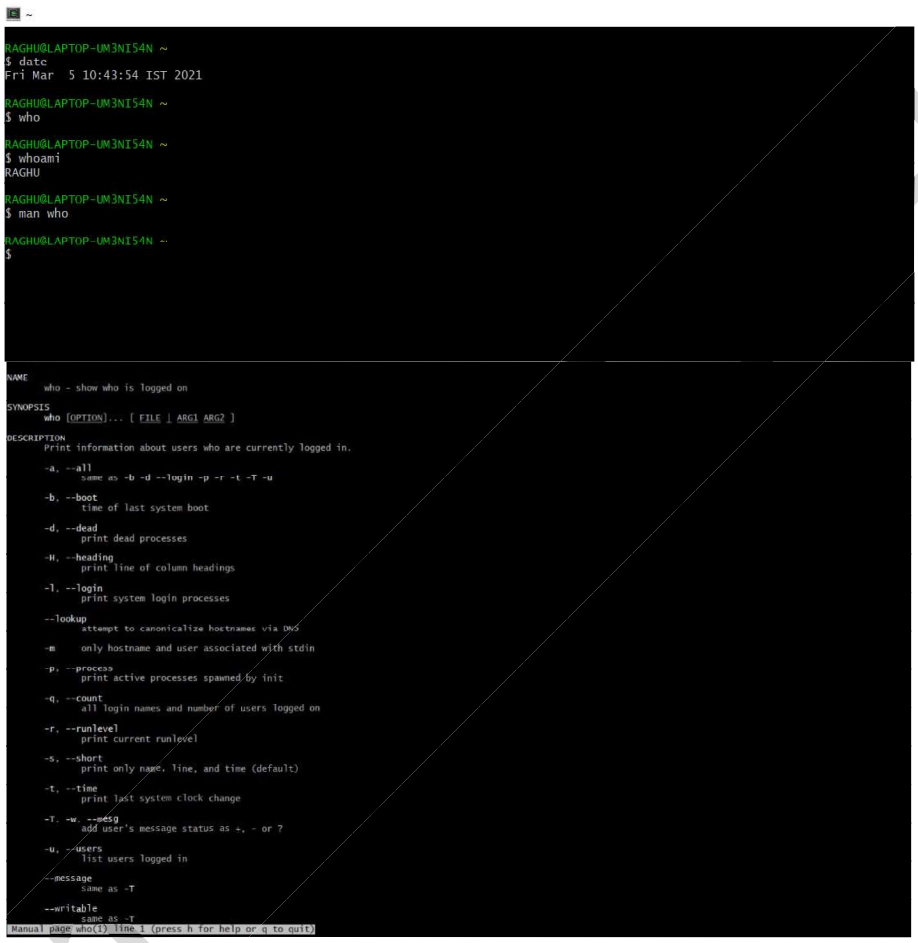
i	Insert text before character.
a	Add text after a character.
I	Insert text at the beginning of a line.
A	Insert text at the end of a line.
x	Delete the current character.
X	Delete the previous character.
o	Open a new blank line after the current line.
O	Open a new blank line before the current line.
dd	Delete the current line.
cc	change the current line.
r	Replace one character.
R	Replace several characters.
s	Substitute new characters for a old one.

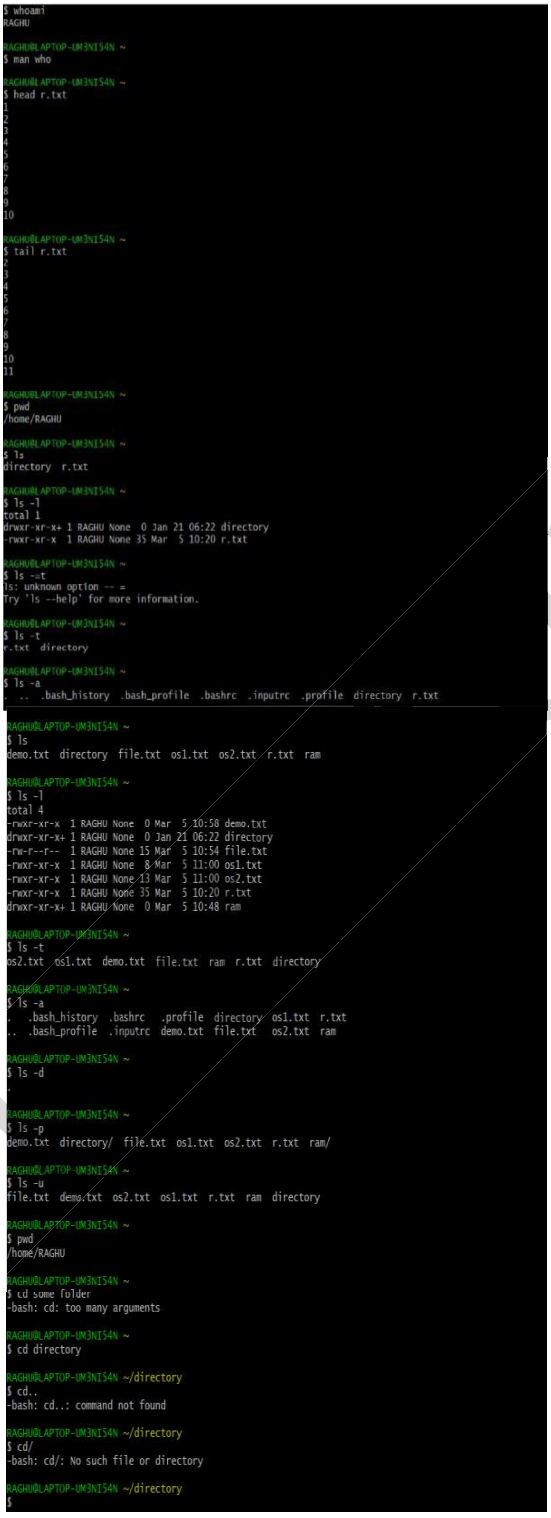
File access commands:

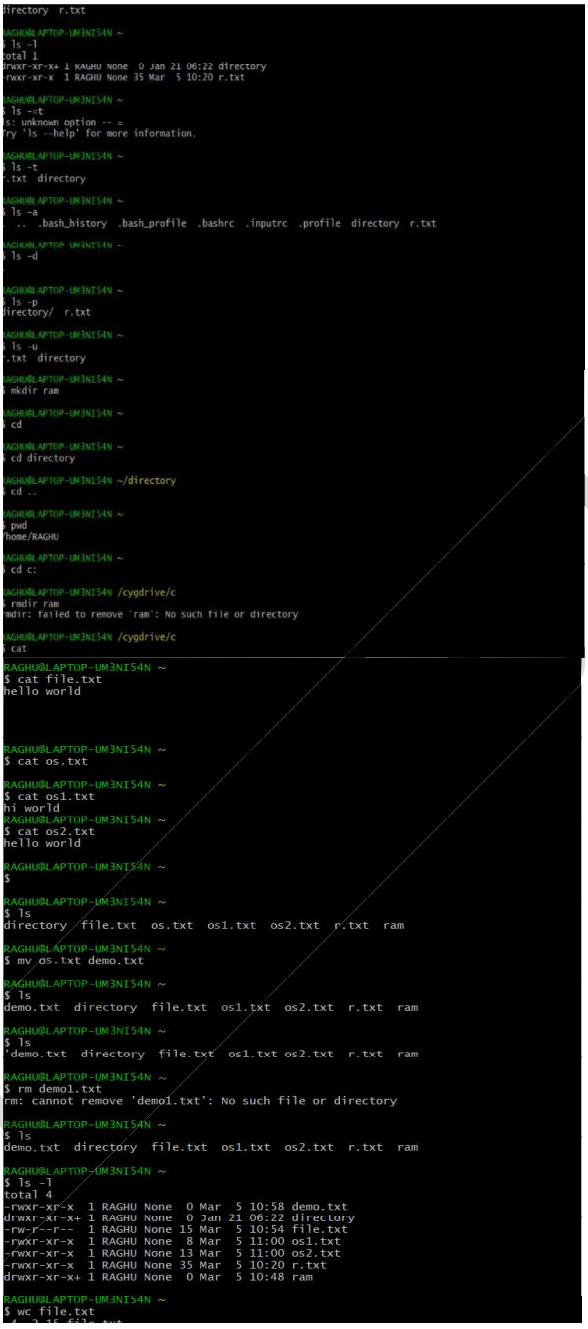
You need some way to open an existing file , to save your work , to save and exit and to abandon an attempt to edit a file without making any changes(just in case you really mess up). Precede these with an Esc.

ZZ or :wq	Save the edited file and exit from the editor.
:w	Save the edited file and stay in the editor.
:q	Quit from the editor.
:q!	Quit the editor under any conditions.

OUTPUT:


~
RAGHU@LAPTOP-UM3NI54N ~
$ date
Fri Mar 5 10:43:54 IST 2021
RAGHU@LAPTOP-UM3NI54N ~
$ who
RAGHU@LAPTOP-UM3NI54N ~
$ whoami
RAGHU
RAGHU@LAPTOP-UM3NI54N ~
$ man who
RAGHU@LAPTOP-UM3NI54N ~
$
NAME
who - show who is logged on
SYNOPSIS
who [OPTION]... [FILE | ARG1 ARG2]
DESCRIPTION
Print information about users who are currently logged in.
-a, --all
same as -b -d --login -p -r -t -T -u
-b, --boot
time of last system boot
-d, --dead
print dead processes
-H, --heading
print line of column headings
-l, --login
print system login processes
--lookup
attempt to canonicalize hostnames via DNS
-m
only hostname and user associated with stdin
-p, --process
print active processes spawned by init
-q, --count
all login names and number of users logged on
-r, --runlevel
print current runlevel
-s, --short
print only name, line, and time (default)
-t, --time
print last system clock change
-T, -w, --mesg
add user's message status as +, - or ?
-u, --users
list users logged in
--message
same as -T
--writable
same as -T
Manual page who(1) line 1 (press h for help or q to quit)


$ whoami
RAGHU
RAGHU@LAPTOP-UM3NI54N ~
$ man who
RAGHU@LAPTOP-UM3NI54N ~
$ head r.txt
1
2
3
4
5
6
7
8
9
10
RAGHU@LAPTOP-UM3NI54N ~
$ tail r.txt
2
3
4
5
6
7
8
9
10
11
RAGHU@LAPTOP-UM3NI54N ~
$ pwd
/home/RAGHU
RAGHU@LAPTOP-UM3NI54N ~
$ ls
directory r.txt
RAGHU@LAPTOP-UM3NI54N ~
$ ls -l
total 1
drwxr-xr-x+ 1 RAGHU None 0 Jan 21 06:22 directory
-rwxr-xr-x 1 RAGHU None 35 Mar 5 10:20 r.txt
RAGHU@LAPTOP-UM3NI54N ~
$ ls -kt
ls: unknown option -- =
Try 'ls --help' for more information.
RAGHU@LAPTOP-UM3NI54N ~
$ ls -t
r.txt directory
RAGHU@LAPTOP-UM3NI54N ~
$ ls -a
. .. .bash_history .bash_profile .bashrc .inputrc .profile directory r.txt
RAGHU@LAPTOP-UM3NI54N ~
$ ls
demo.txt directory file.txt os1.txt os2.txt r.txt ram
RAGHU@LAPTOP-UM3NI54N ~
$ ls -l
total 4
-rwxr-xr-x 1 RAGHU None 0 Mar 5 10:58 demo.txt
drwxr-xr-x+ 1 RAGHU None 0 Jan 21 06:22 directory
-rw-r--r-- 1 RAGHU None 15 Mar 5 10:54 file.txt
-rwxr-xr-x 1 RAGHU None 8 Mar 5 11:00 os1.txt
-rwxr-xr-x 1 RAGHU None 13 Mar 5 11:00 os2.txt
-rwxr-xr-x 1 RAGHU None 35 Mar 5 10:20 r.txt
drwxr-xr-x+ 1 RAGHU None 0 Mar 5 10:48 ram
RAGHU@LAPTOP-UM3NI54N ~
$ ls -t
os2.txt os1.txt demo.txt file.txt ram r.txt directory
RAGHU@LAPTOP-UM3NI54N ~
$ ls -a
. .bash_history .bashrc .profile directory os1.txt r.txt
.. .bash_profile .inputrc demo.txt file.txt os2.txt ram
RAGHU@LAPTOP-UM3NI54N ~
$ ls -d
.
RAGHU@LAPTOP-UM3NI54N ~
$ ls -p
demo.txt directory/ file.txt os1.txt os2.txt r.txt ram/
RAGHU@LAPTOP-UM3NI54N ~
$ ls -u
file.txt demo.txt os2.txt os1.txt r.txt ram directory
RAGHU@LAPTOP-UM3NI54N ~
$ pwd
/home/RAGHU
RAGHU@LAPTOP-UM3NI54N ~
$ cd some folder
-bash: cd: too many arguments
RAGHU@LAPTOP-UM3NI54N ~
$ cd directory
RAGHU@LAPTOP-UM3NI54N ~/directory
$ cd..
-bash: cd..: command not found
RAGHU@LAPTOP-UM3NI54N ~/directory
$ cd/
-bash: cd/: No such file or directory
RAGHU@LAPTOP-UM3NI54N ~/directory
$


directory r.txt
RAGHU@LAPTOP-UM3NI54N ~
$ ls -l
total 1
drwxr-xr-x+ 1 RAGHU None 0 Jan 21 06:22 directory
-rwxr-xr-x 1 RAGHU None 35 Mar 5 10:20 r.txt
RAGHU@LAPTOP-UM3NI54N ~
$ ls -=t
ls: unknown option -- =
Try 'ls --help' for more information.
RAGHU@LAPTOP-UM3NI54N ~
$ ls -t
r.txt directory
RAGHU@LAPTOP-UM3NI54N ~
$ ls -a
. .. .bash_history .bash_profile .bashrc .inputrc .profile directory r.txt
RAGHU@LAPTOP-UM3NI54N ~
$ ls -d
RAGHU@LAPTOP-UM3NI54N ~
$ ls -p
directory/ r.txt
RAGHU@LAPTOP-UM3NI54N ~
$ ls -u
r.txt directory
RAGHU@LAPTOP-UM3NI54N ~
$ mkdir ram
RAGHU@LAPTOP-UM3NI54N ~
$ cd
RAGHU@LAPTOP-UM3NI54N ~
$ cd directory
RAGHU@LAPTOP-UM3NI54N ~/directory
$ cd ..
RAGHU@LAPTOP-UM3NI54N ~
$ pwd
/home/RAGHU
RAGHU@LAPTOP-UM3NI54N ~
$ cd c:
RAGHU@LAPTOP-UM3NI54N /cygdrive/c
$ rmdir ram
rmdir: failed to remove 'ram': No such file or directory
RAGHU@LAPTOP-UM3NI54N /cygdrive/c
$ cat
RAGHU@LAPTOP-UM3NI54N ~
$ cat file.txt
hello world
RAGHU@LAPTOP-UM3NI54N ~
$ cat os.txt
RAGHU@LAPTOP-UM3NI54N ~
$ cat os1.txt
hi world
RAGHU@LAPTOP-UM3NI54N ~
$ cat os2.txt
hello world
RAGHU@LAPTOP-UM3NI54N ~
$
RAGHU@LAPTOP-UM3NI54N ~
$ ls
directory file.txt os.txt os1.txt os2.txt r.txt ram
RAGHU@LAPTOP-UM3NI54N ~
$ mv os.txt demo.txt
RAGHU@LAPTOP-UM3NI54N ~
$ ls
demo.txt directory file.txt os1.txt os2.txt r.txt ram
RAGHU@LAPTOP-UM3NI54N ~
$ ls
'demo.txt directory file.txt os1.txt os2.txt r.txt ram
RAGHU@LAPTOP-UM3NI54N ~
$ rm demo1.txt
rm: cannot remove 'demo1.txt': No such file or directory
RAGHU@LAPTOP-UM3NI54N ~
$ ls
demo.txt directory file.txt os1.txt os2.txt r.txt ram
RAGHU@LAPTOP-UM3NI54N ~
$ ls -l
total 4
-rwxr-xr-x 1 RAGHU None 0 Mar 5 10:58 demo.txt
drwxr-xr-x+ 1 RAGHU None 0 Jan 21 06:22 directory
-rw-r--r-- 1 RAGHU None 15 Mar 5 10:54 file.txt
-rwxr-xr-x 1 RAGHU None 8 Mar 5 11:00 os1.txt
-rwxr-xr-x 1 RAGHU None 13 Mar 5 11:00 os2.txt
-rwxr-xr-x 1 RAGHU None 35 Mar 5 10:20 r.txt
drwxr-xr-x+ 1 RAGHU None 0 Mar 5 10:48 ram
RAGHU@LAPTOP-UM3NI54N ~
$ wc file.txt

Result:

Thus the Basics of Unix operating system , commands and vi have been studied successfully.

Ex.No:2 **SHELL PROGRAMMING**

DATE:

2.a SUM OF TWO NUMBERS

Aim:

To find the sum of two given numbers.

Algorithm:

Step 1: Start

Step 2: Declare variables a, b and sum.

Step 3: Read values a and b.

Step 4: Add the two numbers and assign the result to sum.

Step 5: Display sum

Step 6: Stop

Program

```
echo " enter first no "
read a
echo " enter second no "
read b
c= expr $a + $b
```

INPUT:

```
 [student@localhost student]$ sh sum
enter first no
3
 enter second no
4
```

OUTPUT:

RESULT:

Thus the sum of two numbers were implemented and output was verified successfully.

2.b) FIBONACCI SERIES

Aim:

To find the Fibonacci series of the given number.

Algorithm:

Step 1: Start

Step 2: Declare variables first_term,second_term and temp.

Step 3: Initialize variables first_term←0 second_term←1

Step 4: Display first_term and second_term

Step 5: Repeat the steps

5.1: temp←second_term

5.2: second_term←second_term+first term

5.3: first_term←temp

5.4: Display second_term

Step 6: Stop

Program:

```
echo "enter a number"
read n
i=0
a=0
b=1
c=0
echo "fibonacci series is"
echo "$a"
echo "$b"
n1=` expr $n - 2`
while [ $i -lt $n1 ]
do
c=` expr $a + $b`
echo "$c"
a=$b
b=$c
i=` expr $i + 1`
done
```

INPUT:

[student@localhost student]$ sh fibo
enter a number
5

OUTPUT:

RESULT:

Thus the Fibonacci series were implemented and output was verified successfully.

2.c) POSITIVE OR NEGATIVE

Aim:

To find whether the given number is positive or negative.

Algorithm:

Step 1: Start

Step 2: Enter the value

Step 3: Read the value

Step 4: If Number is >0
Then
Print “positive”
else
if Number is<0
Print “Negative”
Else
Print “zero”.

Step 5: Stop

Program:

```
echo " enter a number "
read a
if [ $a -eq 0 ]
then
echo " no is zero "
elif [ $a -gt 0 ]
then
echo " no is positive "
else
echo " no is negative "
fi
```

INPUT:

[student@localhost student]$ sh positive
enter a number
-5

OUTPUT:

RESULT:

Thus the positive or negative were implemented and output was verified successfully.

2.d) GREATEST OF THREE NUMBERS

Aim:

To find the Greatest of the given three numbers.

Algorithm:

Step 1: Start
Step 2: Declare variables a,b and c.
Step 3: Read variables a,b and c.
Step 4: If a>b and If a>c
Display a is the largest number.
Else
Display c is the largest number.
Else
If b>c
Display b is the largest number.
Else
Display c is the greatest number.
Step 5: Stop

Program:

```
echo " enter the three numbers "
read x
read y
read z
if [ $x -gt $y -a $x -gt $z ]
then
echo "$x is greater"
elif [ $y -gt $x -a $y -gt $z ]
then
echo "$y is greater"
else
echo "$z is greater"
fi
```

INPUT:

[student@localhost student]$ sh greatest
 enter the three numbers
5
8
9

OUTPUT:

RESULT:

Thus the greatest of three numbers were implemented and ouput was verified successfully.

2.e) ARMSTRONG NUMBER

Aim:

To find whether the given number is an Armstrong number or not.

Algorithm:

Step 1: Start
Step 2: Input the value N
Step 3: Set SUM = 0
Step 4: Number = N
Step 5: Repeat While Number ≠ 0
SUM = (Number%10)**3 + SUM
Number = Number/10

Step 6: If SUM == N then
Print N is an Armstrong Number.
Else
Print N is not an Armstrong Number.
Step 7: Stop

Program:

```
echo "enter the number"
read num
ans=0
n=$num
while [ $n -gt 0 ]
do
q=` expr $n % 10`
ans=` expr $ans + $q \* $q \* $q`
n=` expr $n / 10`
done
if [ $ans -eq $num ]
then
echo "number is armstrong"
else
echo "number is not armstrong"
fi
```

INPUT:

[student@localhost student]$ sh armstrong
enter the number
153

OUTPUT:

RESULT:

Thus the Armstrong number were implemented and output was verified successfully.

2.f) FACTORIAL NUMBER

Aim :

To write a program to generate the factorial of given number.

Algorithm:

Step 1: Start

Step 2: Declare variables n, fact and i.

Step 3: Initialize variables fact←1 i←1

Step 4: Read value of n

Step 5: Repeat the steps until i=n 5.1: fact←fact* i 5.2: i←i+1

Step 6: Display factorial

Step 7: Stop.....

Program:

```
n=0
fact=1
g=0
y=1
echo "enter no to find the factorial:"
read n
g=$n
while [ $n -ge $y ]
do
fact=` expr $fact \* $n`
n=` expr $n - 1 `
done
echo "factorial for $g is $fact"
```

INPUT:

[student@localhost student]$ sh fact
enter no to find the factorial:
5

OUTPUT:

RESULT:

Thus the shell Programs were implemented and output was verified successfully.

Ex.No:3 **SYSTEM CALLS FORK**
DATE:

Aim:

To write a Program to create a process using the fork system calls.

Algorithm:

1. Start the program.
2. Read the input from the command line.
3. Use fork() system call to create process, getppid() system call used to get the parent process ID and getpid() system call used to get the current process ID

6. If the pid > 0 Print parent and its process id otherwise print child and its
   Process id along with system date
8. Stop the program.

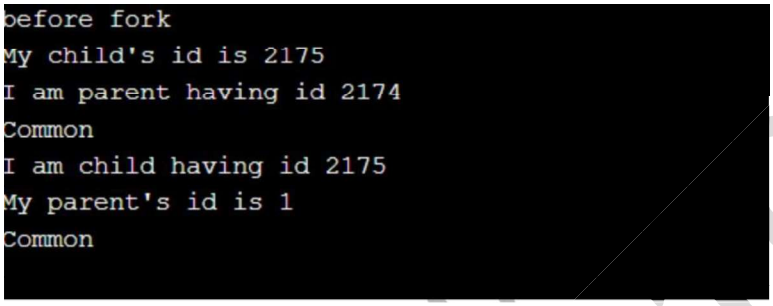
Program :

```
#include<iostream.h>
#include<sys/types.h>
#include<errno.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
pid_t child;
cout<<"pid:"<<getpid()<<"patent:"<<getppid()<<endl;
switch(child=fork())
        {
case (pid_t) -1: perror ("fork");
break;
case (pid_t) 0: cout<<"child
created:pid:"<<getpid()<<"patent:"<<getppid()<<endl;
exit(0);
default:cout<<"parent after fork pid:"<<"child pid:"<<child<<endl;
}
return 0;
}
```

INPUT:

[student@localhost student]$ cc proc.c

OUTPUT:



RESULT:

Thus the Program was implemented and output was verified successfully

Ex.No:4 **CPU SCHEDULING POLICIES**
DATE:

AIM:

To Implement CPU Scheduling Algorithms

1. FCFS (First Come First Served),
2. Shortest Job First,
3. Priority Based Scheduling,
4. Round Robin,
5. Comparative Study.

Ex No.4 (a) **FIRST COME FIRST SERVED CPU SCHEDULING**
DATE:

Aim:

To Implement CPU Scheduling Algorithms using first come first served

Algorithm:

1. Start the process.
2. Declare the array size.
3. Get the number of elements to be inserted.
4. Select the process that first arrived in the ready queue
5. Make the average waiting the length of next process.
6. Start with the first process from it's selection as above and let other process to be in queue.
7. Calculate the total number of burst time.
8. Display the values.
9. Stop the process.

Program:

```
#include<stdio.h>
main()
{
int i,n,w[10],e[10],b[10];
float wa=0,ea=0;
printf("\nEnter the no of jobs: ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\n Enter the burst time of job %d :",i+1);
scanf("%d",&b[i]);
if(i==0)
{
w[0]=0;
e[0]=b[0];
}
else
{
e[i]=e[i-1]+b[i];
w[i]=e[i-1];
}
}
printf("\n\n\tJobs\tWaiting time   \tBursttime\tExecution time\n");
printf("\t_____\n");
for(i=0;i<n;i++)
{
printf("\t%d\t\t%d\t\t%d\t\t%d\n",i+1,w[i],b[i],e[i]);
wa+=w[i];
ea+=e[i];
}
wa=wa/n;
ea=ea/n;
printf("\n\nAverage waiting time is :%2.2f ms\n",wa);
printf("\nAverage execution time is:%2.2f ms\n\n",ea);}
```

INPUT:

Enter the no of jobs: 4

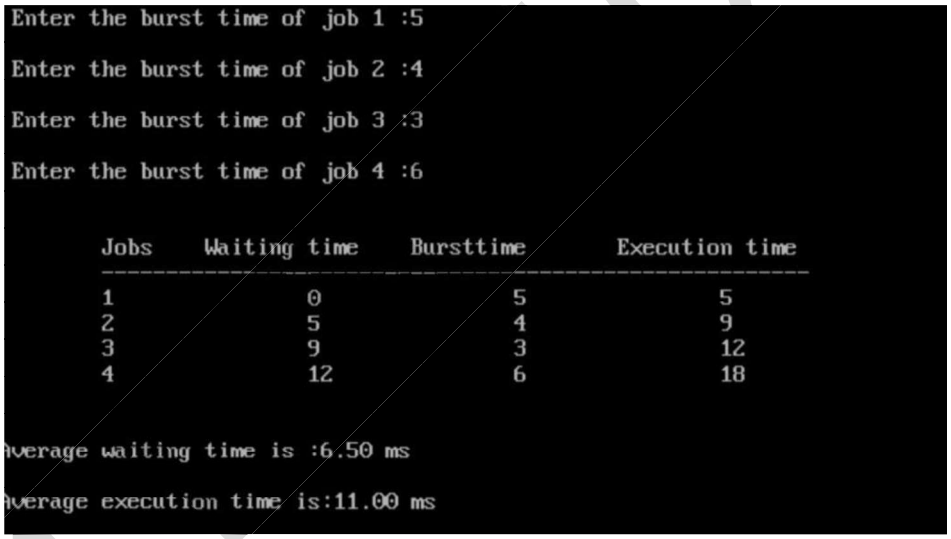
Enter the burst time of job 1 : 5

Enter the burst time of job 2 : 4

Enter the burst time of job 3 : 3

Enter the burst time of job 4 : 6

OUTPUT:



RESULT:

Thus the Program was implemented and output was verified successfully.

Ex No.4 (b) **SHORTEST JOB FIRST CPU SCHEDULING**

DATE:

Aim:

To Implement CPU Scheduling Algorithms using shortest job first.

Algorithm:

1. Start the process.
2. Declare the array size.
3. Get the number of elements to be inserted.
4. Select the process which have shortest burst will execute first.
5. If two process have same burst length then FCFS scheduling algorithm used.
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 to be in queue.
8. Calculate the total number of burst time.
9. Display the values.
10. Stop the process.

Program:

```
#include<stdio.h>
 struct sjfs
 {
  char pname[10];
  int btime;
 }proc[10],a;
void main( )
 {
  struct sjfs proc[10];
  int n,i,j;
  int temp=0,temp1=0,temp2;
  char name[20];
  float tt,awt;
      printf("Enter the number of processes:\n");
  scanf("%d", &n);
  for(i=0;i<n;i++)
   {
   printf("Enter the process name: \n");
   scanf("%s",&proc[i].pname);
   printf("Enter the Burst time:\n");
   scanf("%d",&proc[i].btime);
   }
  for(i=0;i<n;i++)
  {
```

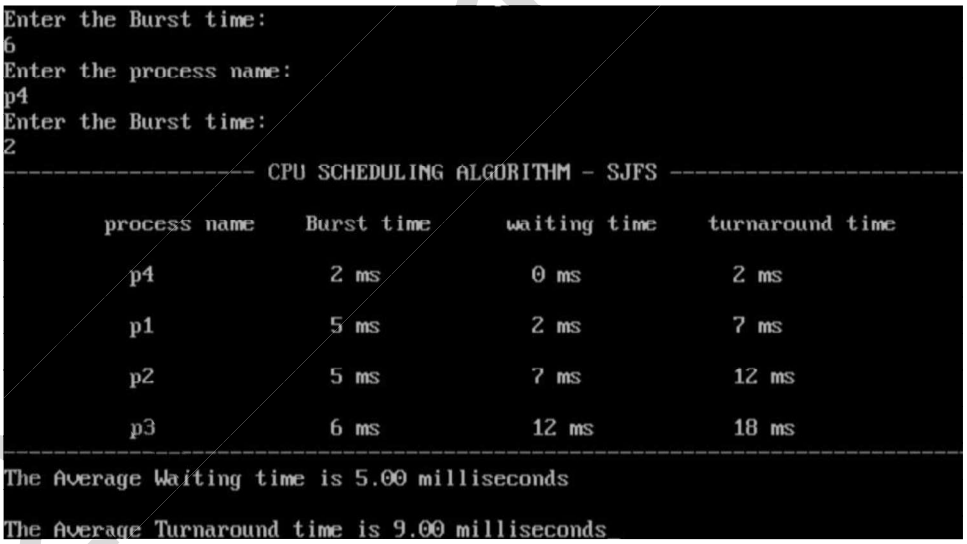
```
 for(j=0;j<n;j++)
  {
  if(proc[i].btime<proc[j].btime)
   {
    a=proc[i];
    proc[i]=proc[j];
    proc[j]=a;
   }
  }
 }
 printf("---------- CPU SCHEDULING ALGORITHM - SJFS ----------");
 printf("\n\tprocess name \tBurst time \twaiting time \tturnaround time\n");
 temp=0;
 for(i=0;i<n;i++)
 {
  temp=temp1+temp+proc[i].btime;
  temp1=temp1+proc[i].btime;
  temp2=temp1-proc[i].btime;
  printf("\n\t  %s     \t  %d ms      \t %d ms \t %d ms\n",proc[i].pname,proc[i].btime,temp2,temp1);
 }
 printf("--------------------------------------------------");
 awt=(temp-temp1)/n;
```

```
tt=temp/n;
printf("\nThe Average Waiting time is %4.2f milliseconds\n",awt);
printf("\nThe Average Turnaround time is %4.2f",tt); }
```

INPUT:

Enter the number of processes:
4
Enter the process name:
p1
Enter the Burst time:
5
Enter the process name:
p2
Enter the Burst time:
5
Enter the process name:
p3
Enter the Burst time:
6
Enter the process name:
p4
Enter the Burst time:
2

OUTPUT:



RESULT:

Thus the Program was implemented and output was verified successfully.

Ex. No.4 (c) **ROUND ROBIN CPU SCHEDULING**

DATE:

Aim:

To Implement CPU Scheduling Algorithms using Round Robin

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.
    i. Make the CPU scheduler goes around the ready queue allocating CPU to each process for the time interval specified.
8. Make the CPU scheduler picks the first process and sets time to interrupt after quantum expired dispatches the process.
9. If the process has burst less than the time quantum than the processreleases the CPU

Program:

```
#include<stdio.h>
#include<malloc.h>
void line(int i)
{
 int j;
 for(j=1; j<=i; j++)
   printf("-");
 printf("\n");
}
 struct process
{
 int p_id;
 int etime, wtime, tatime;
 };
 struct process *p, *tmp;
 int i, j, k, l, n, time_slice, ctime;
 float awtime=0, atatime=0;
 int main()
{
 printf("Process Scheduling - Round Robin \n");
 line(29);
 printf("Enter the no.of processes : ");
 scanf("%d", &n);
 printf("Enter the time slice : ");
 scanf("%d", &time_slice);
 printf("Enter the context switch time : ");
 scanf("%d", &ctime);
 p=(struct process*) calloc(n+1, sizeof(struct process));
 tmp=(struct process*) calloc(n+1, sizeof(struct process));
 for(i=1; i<=n; i++)
 {
   printf("Enter the execution time of process %d : ", i-1);
   scanf("%d", &p[i].etime);
   p[i].wtime=(time_slice+ctime)*i-1;
   awtime += p[i].wtime;
   p[i].p_id=i-1;
   tmp[i]=p[i];
 }
 i=0; j=1; k=0;
 while(i<n)
```

```
{
 for(j=1; j<=n; j++)
 {
       if(tmp[j].etime <= time_slice && tmp[j].etime!=0)
  {
     k=k+tmp[j].etime;
     tmp[j].etime=0;
     p[j].tatime=k;
     atatime += p[j].tatime;
     k=k+ctime;
     i++;
  }
  if(tmp[j].etime>time_slice && tmp[j].etime!=0)
  {
     k=k+time_slice+ctime;
     tmp[j].etime -= time_slice;
  }
 }
}

 awtime=awtime/n;
 atatime=atatime/n;
 printf("\nShedule \n");
 line(60);
 printf("Process\t\tExecution\tWait\t\tTurnaround\n");
 printf("Id No\t\ttime\t\ttime\t\ttime\n");
 line(60);
 for(i=1;i<=n;i++)
     printf("%7d\t%14d\t%8d\t%14d \n", p[i].p_id, p[i].etime,
p[i].wtime,
 p[i].tatime);
 line(60);
 printf("Avg waiting time     :\t%2f \n",awtime);
 printf("Avg turn around time :\t%2f\n",atatime);
 line(60);
 return(0);
}
```

Input :

Round Robin Scheduling

Enter the no of processes : 3
Enter the time slice : 4
Enter the context switch time : 5
Enter the execution time of process 0 : 6
Enter the execution time of process 1 : 6
Enter the execution time of process 2 : 5

Output:

```
C:\TURBOC3\BIN>TC
Process Scheduling - Round Robin
-----------------------------
Enter the no.of processes : 3
Enter the time slice : 4
Enter the context switch time : 5
Enter the execution time of process 0 : 6
Enter the execution time of process 1 : 6
Enter the execution time of process 2 : 5

Shedule
-------------------------------------------------------------
Process         Execution       Wait            Turnaround
Id No           time            time            time
-------------------------------------------------------------
      0             6         8                   29
      1             6        17                   36
      2             5        26                   42
-------------------------------------------------------------
Avg waiting time     :  17.000000
Avg turn around time :  35.666668
```

RESULT:

Thus the Program was implemented and output was verified successfully.

Ex No.4 (d) **PRIORITY CPU SCHEDULING**

DATE:

Aim:

To Implement CPU Scheduling Algorithms using priority.

Algorithm:

1. Start the process.
2. Declare the array size.
3. Get the number of elements to be inserted.
4. Get the priority for each process and value
5. start with the higher priority process from it's initial position let other process to be queue.
6. Calculate the total number of burst time.
7. Display the values
8. Stop the process.

Program:

```
#include<stdio.h>
#include<malloc.h>
void line(int i)
{
 int j;
 for(j=1;j<=i;j++)
 printf("-");
 printf("\n");
}
struct process
{
 int p_id,priority;
 int etime,wtime,tatime;
};
struct process *p,temp;
int i,j,k,l,n;
float awtime=0,atatime=0;
int main()
{
 printf("Priority Scheduling\n");
 line(29);
 printf("Enter the no of processes : ");
 scanf("%d",&n);
 p=(struct process *) calloc(n+1,sizeof(struct process));
 p[0].wtime=0;
 p[0].tatime=0;
 for(i=1;i<=n;i++)
 {
  printf("Enter the execution time of process %d:",i-1);
  scanf("%d",&p[i].etime);
  printf("Enter the priority of process %d:",i-1);
  scanf("%d",&p[i].priority);
  p[i].p_id=i;
 }
 for(i=1;i<=n;i++)
 for(j=1;j<=n-i;j++)
  if(p[i].priority>p[j+1].priority)
   {
    temp=p[j];
    p[j]=p[j+1];
```

```
    p[j+1]=temp;
   }
 for(i=1;i<=n;i++)
 {
  p[i].wtime=p[i-1].tatime;
  p[i].tatime=p[i-1].tatime+p[i].etime;
  awtime+=p[i].wtime;
  atatime+=p[i].tatime;
}
 awtime=awtime/n;
 awtime=atatime/n;
 printf("\nSchedule\n");
 line(60);
 printf("Process\t\texection\twait\t\tturnaround\n");
 printf("Id No\t\t time\t\ttime\t\ttime\n");
 line(60);
 for(i=1;i<=n;i++)
  printf("%7d\t%14d\t%8d\t%14d\n",p[i].p_id,p[i].etime,p[i].tatime);
 line(60);
 printf("Avg waiting time:\t %2f\n",awtime);
 printf("Avg turnaround time:\t %2f\n",atatime);
 line(60);
 return(0);
}
```

Input:

Priority Scheduling

Enter the no of processes : 3

Enter the execution time of process0 : 5
Enter the priority of process0 : 6
Enter the execution time of process1: 3
Enter the priority of process1: 5
Enter the execution time of processes2: 1
Enter the priority of process2: 3

Output:

```
Enter the no of processes : 3
Enter the execution time of process 0:5
Enter the priority of process 0:6
Enter the execution time of process 1:3
Enter the priority of process 1:5
Enter the execution time of process 2:1
Enter the priority of process 2:3

Schedule
-------------------------------------------------------------
Process         exection        wait            turnaround
Id No            time           time            time
-------------------------------------------------------------
     2             3         3                     3
     3             1         4                     3
     1             5         9                     3
-------------------------------------------------------------
Avg waiting time:       5.333333
Avg turnaround time:    16.000000
-------------------------------------------------------------
```

RESULT:

Thus the Program was implemented and output was verified successfully

Ex.No: 5

INTERPROCESS COMMUNICATION USING SHARED MEMORY

DATE:

AIM:

To implement Inter Process Communication using Message queue.

Algorithm:

1. Start the program
2. Use shmget() to allocate a shared memory
3. Use shmat() to attach a shared memory to an address space
4. Write the message in the shared memory area using the parent process
5. Read the message from the shared memory area using the child process and print it
6. Stop the program

Program:

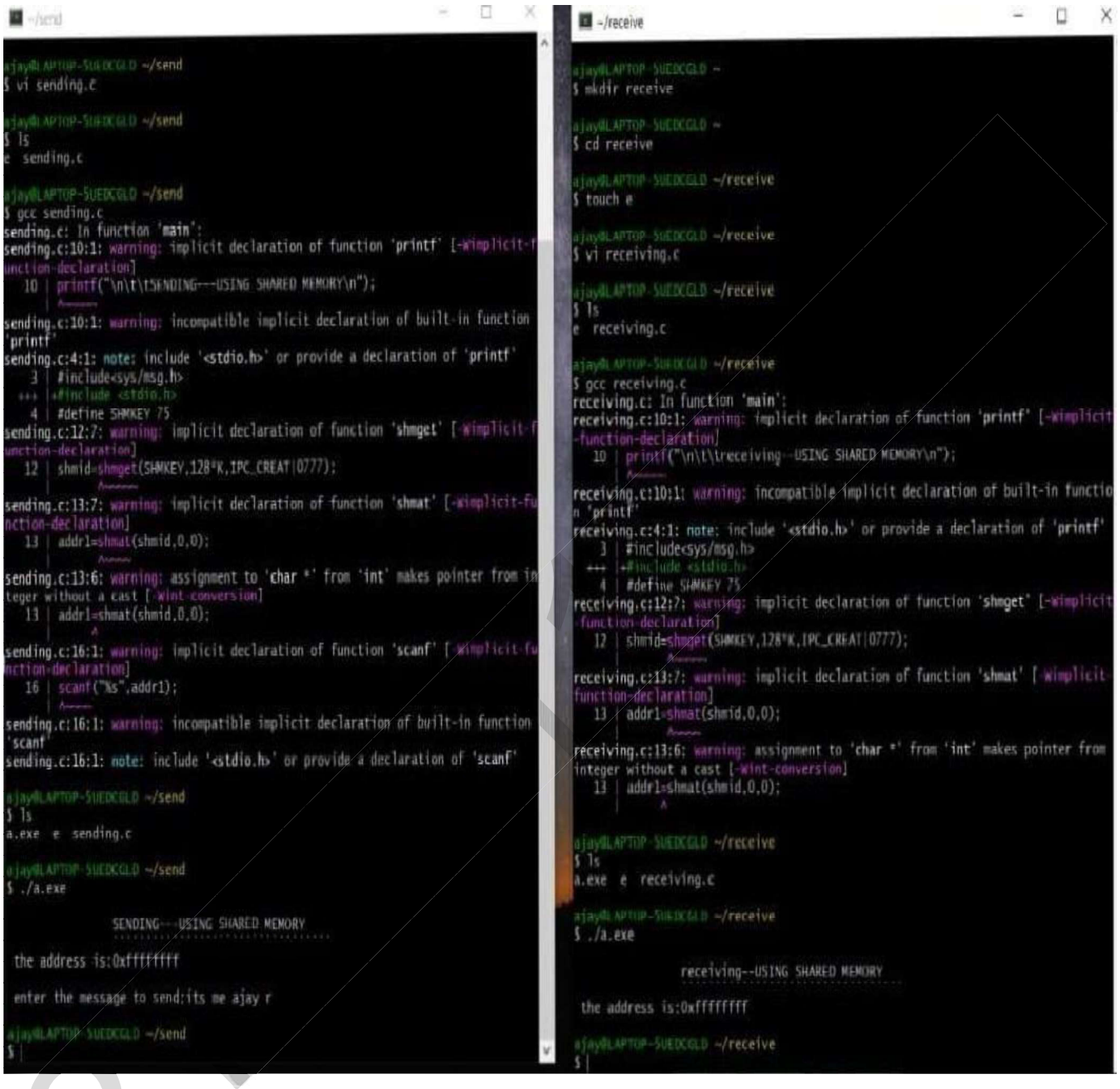
SHARED MEMORY –SENDING

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#define SHMKEY 75
#define K 1024
int main()
{
int shmid;
char *addr1;
printf("\n\t\tSENDING---USING SHARED MEMORY\n");
printf("\t\t``````````````````````````````");
shmid=shmget(SHMKEY,128*K,IPC_CREAT|0777);
addr1=shmat(shmid,0,0);
printf("\n the address is:0x%x\n",addr1);
printf("\n enter the message to send:");
scanf("%s",addr1);
return(0);
}
```

SHARED MEMORY –RECEIVING

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#define SHMKEY 75
#define K 1024
int main()
{
int shmid;
char *addr1;
printf("\n\t\treceiving--USING SHARED MEMORY\n");
printf("\t\t``````````````````````````````");
shmid=shmget(SHMKEY,128*K,IPC_CREAT|0777);
addr1=shmat(shmid,0,0);
printf("\n the address is:0x%x\n",addr1);
printf("\n the received message is:%s\n",addr1);
return(0);
}
```

OUTPUT:



```
~/send
ajay@LAPTOP-5UEDCGLD ~/send
$ vi sending.c

ajay@LAPTOP-5UEDCGLD ~/send
$ ls
e  sending.c

ajay@LAPTOP-5UEDCGLD ~/send
$ gcc sending.c
sending.c: In function 'main':
sending.c:10:1: warning: implicit declaration of function 'printf' [-wimplicit-function-declaration]
   10 | printf("\n\t\tSENDING---USING SHARED MEMORY\n");
      | ^~~~~~
sending.c:10:1: warning: incompatible implicit declaration of built-in function 'printf'
sending.c:4:1: note: include '<stdio.h>' or provide a declaration of 'printf'
    3 | #include<sys/msg.h>
  +++ |+#include <stdio.h>
    4 | #define SHMKEY 75
sending.c:12:7: warning: implicit declaration of function 'shmget' [-wimplicit-function-declaration]
   12 | shmid=shmget(SHMKEY,128*K,IPC_CREAT|0777);
      |       ^~~~~~
sending.c:13:7: warning: implicit declaration of function 'shmat' [-wimplicit-function-declaration]
   13 | addr1=shmat(shmid,0,0);
      |       ^~~~~
sending.c:13:6: warning: assignment to 'char *' from 'int' makes pointer from integer without a cast [-Wint-conversion]
   13 | addr1=shmat(shmid,0,0);
      |      ^
sending.c:16:1: warning: implicit declaration of function 'scanf' [-wimplicit-function-declaration]
   16 | scanf("%s",addr1);
      | ^~~~~
sending.c:16:1: warning: incompatible implicit declaration of built-in function 'scanf'
sending.c:16:1: note: include '<stdio.h>' or provide a declaration of 'scanf'

ajay@LAPTOP-5UEDCGLD ~/send
$ ls
a.exe  e  sending.c

ajay@LAPTOP-5UEDCGLD ~/send
$ ./a.exe

                SENDING---USING SHARED MEMORY
                ..............................

 the address is:0xffffffff

 enter the message to send:its me ajay r

ajay@LAPTOP-5UEDCGLD ~/send
$
```

```
~/receive
ajay@LAPTOP-5UEDCGLD ~
$ mkdir receive

ajay@LAPTOP-5UEDCGLD ~
$ cd receive

ajay@LAPTOP-5UEDCGLD ~/receive
$ touch e

ajay@LAPTOP-5UEDCGLD ~/receive
$ vi receiving.c

ajay@LAPTOP-5UEDCGLD ~/receive
$ ls
e  receiving.c

ajay@LAPTOP-5UEDCGLD ~/receive
$ gcc receiving.c
receiving.c: In function 'main':
receiving.c:10:1: warning: implicit declaration of function 'printf' [-wimplicit-function-declaration]
   10 | printf("\n\t\treceiving---USING SHARED MEMORY\n");
      | ^~~~~~
receiving.c:10:1: warning: incompatible implicit declaration of built-in function 'printf'
receiving.c:4:1: note: include '<stdio.h>' or provide a declaration of 'printf'
    3 | #include<sys/msg.h>
  +++ |+#include <stdio.h>
    4 | #define SHMKEY 75
receiving.c:12:7: warning: implicit declaration of function 'shmget' [-wimplicit-function-declaration]
   12 | shmid=shmget(SHMKEY,128*K,IPC_CREAT|0777);
      |       ^~~~~~
receiving.c:13:7: warning: implicit declaration of function 'shmat' [-wimplicit-function-declaration]
   13 | addr1=shmat(shmid,0,0);
      |       ^~~~~
receiving.c:13:6: warning: assignment to 'char *' from 'int' makes pointer from integer without a cast [-Wint-conversion]
   13 | addr1=shmat(shmid,0,0);
      |      ^

ajay@LAPTOP-5UEDCGLD ~/receive
$ ls
a.exe  e  receiving.c

ajay@LAPTOP-5UEDCGLD ~/receive
$ ./a.exe

                receiving---USING SHARED MEMORY
                ................................

 the address is:0xffffffff

ajay@LAPTOP-5UEDCGLD ~/receive
$
```

RESULT:

The IPC using Shared memory segment is implemented and verified successfully.

Ex.No: 6 IMPLEMENTATION OF BANKER'S ALGORITHM

DATE:

Aim:

To write a C program to implement Bankers Algorithm.

Algorithm:

1. Start
2. Declare available(k) where is the instances of resource type , max[i,j]=k where process i request at most k instances of resource type j and allocation[i,j]=k where for process i k instances of resource type j are allocated
3. Compute Need[i,j]=Max[i,j]-allocation[i,j]
4. Let work and finish be vectors of length i and j respectively.
5. Initialize Work=available
6. if finish[i]=false for all processes

7. Find a process i such that $Finish\ [i] = false$ and $Need_i \leq Work$ and if no such i exists go to step 10

8. $Work := Work + Allocation_i$

9. If Finish[i]=true go to step 7

10 . If finish[i]=true for all processes then display the system is in safe state else unsafe state

Program:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int i,j,z;
int res[10];
int resource,process,boolean=0;
int
allocation[10][10],sel[10],max[10][10],need[10][10],work[10],work1[10]
;
int check=0;
int  adpro;
int ar[10];
printf("Welcome to Bankers Algorithms");
printf("\n Enter the no .of processes:");
scanf("%d",&process);
printf("\n Enter the number of Resources");
scanf("%d",&resource);
for(i=0;i<resource;i++)
{
printf("Enter the instances of Resource %d:",i+1);
scanf("%d",&res[i]);
}
for(i=0;i<process;i++)
{
      printf("\n Enter allocated resources for process %d:",i+1);
      for(j=0;j<resource;j++)
      {
      printf("\n Resource %d:",j+1);
      scanf("%d",&allocation[i][j]);
}
}
      for(i=0;i<10;i++)
      sel[i]=-1;
for(i=0;i<process;i++)
{
      printf("Enter maximum need of process: %d",i+1);
      for(j=0;j<resource;j++)
      {
      printf("\n Resource %d:",j+1);
      scanf("%d",&max[i][j]);
```

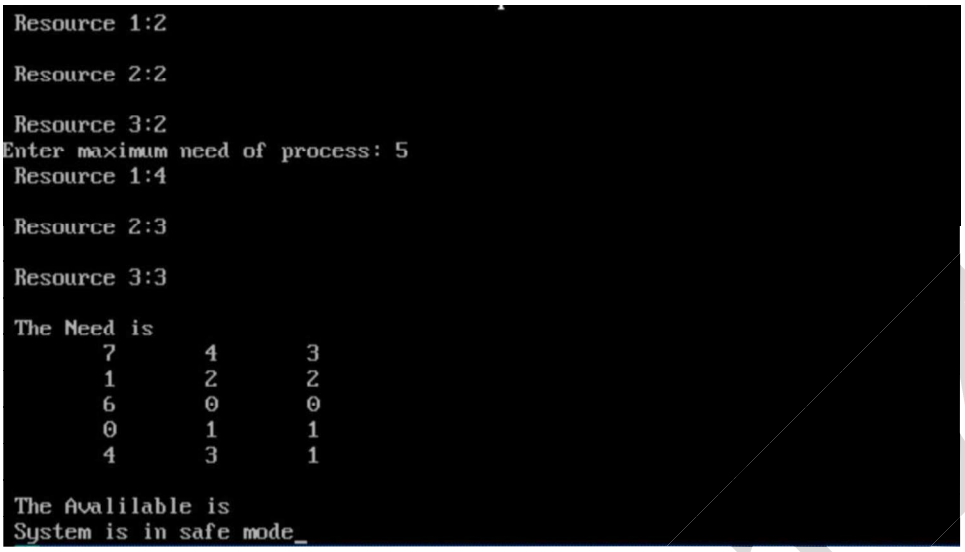
```
        }
    }
    for(i=0;i<process;i++)
        for(j=0;j<resource;j++)
            need[i][j]=max[i][j]-allocation[i][j];
    printf("\n The Need is \n");
    for(i=0;i<process;i++)
    {
        for(j=0;j<resource;j++)
        printf("\t%d",need[i][j]);
        printf("\n");
    }
    printf("\n The Avalilable is ");
    for(i=0;i<resource;i++)
    {
    work[i]=0;
    for(j=0;j<process;j++)
    work[i]=work[i]+allocation[j][i];
    work[i]=res[i]-work[i];
    }
    for(z=0;z<process;z++)
    {
    for(i=0;i<process;i++)
    {
        for(j=0;j<resource;j++)
        {
        if(work[j]>=need[i][j]&&sel[i]==-1)
        {
        boolean=1;
        }
        else
        {
        boolean=0;
        break;
        }
    }
    if(boolean==1)
    {
        sel[i]=1;
        for(j=0;j<resource;j++)
        {
        work[j]=work[j]+allocation[i][j];
        }
```

```
  }
}
}
if(check==1)
      printf("\n System is in Unsafe mode");
else
      printf("\n System is in safe mode");
}
```

Input:

Enter the no .of processes:5
Enter the number of Resources3
Enter the instances of Resource 1:10
Enter the instances of Resource 2:5
Enter the instances of Resource 3:7
Enter allocated resources for process 1:
Resource 1:0
Resource 2:1
Resource 3:0
Enter allocated resources for process 2:
Resource 1:2
Resource 2:0
Resource 3:0
Enter allocated resources for process 3:
Resource 1:3
Resource 2:0
Resource 3:2
Enter allocated resources for process 4:
Resource 1:2
Resource 2:1
Resource 3:1
Enter allocated resources for process 5:
Resource 1:0
Resource 2:0
Resource 3:2
Enter maximum need of process: 2
Resource 1:3
Resource 2:2
Resource 3:2
Enter maximum need of process: 3
Resource 1:9
Resource 2:0
Resource 3:2
Enter maximum need of process: 4
Resource 1:2
Resource 2:2
Resource 3:2
Enter maximum need of process: 5
Resource 1:4
Resource 2:3
Resource 3:3

Output:



Result:

The Banker's Algorithm was implemented and verified successfully.

Ex.No: 7

IMPLEMENTAION OF DINING PHILOSOPHER'S PROBLEM

DATE:

Aim:

Write a C program to implement Dining Philosophers problem.

Program:

```
#include<stdio.h>
char state[10],self[10],spoon[10];
void test(int k)
{
if((state[(k+4)%5]!='e')&&(state[k]=='h')&&(state[(k+1)%5]!='e'))
{
state[k]='e';
self[k]='s';
spoon[k]='n';
spoon[(k+4)%5]='n';
}
}
void pickup(int i)
{
state[i]='h';
test(i);
if(state[i]=='h')
{
self[i]='w';
}
}
void putdown(int i)
{
state[i]='t';
spoon[i]='s';
spoon[i-1]='s';
test((i+4)%5);
test((i+1)%5);
}

int main()
{
```

```
int ch,a,n,i;
printf("\t\t Dining Philosopher Problem\n");
for(i=0;i<5;i++)
{
state[i]='t';
self[i]='s';
spoon[i]='s';
}
printf("\t\t Initial State of Each Philosopher\n");
printf("\n\t Philosopher No.\t Think/Eat  \tStatus    \tspoon");
for(i=0;i<5;i++)
{
      printf("\n\t\t %d\t\t%c\t\t%c\t%c\n",i+1,state[i],self[i],spoon[i]);
}
printf("\n 1.Exit \n 2.Hungry\n3.Thinking\n");
printf("\n Enter your choice\n");
printf("\t\t");
scanf("%d",&ch);
while(ch!=1)
{
      switch(ch)
      {
      case 2:
      {
      printf("\n\t Enter which philosopher is hungry\n");
      printf("\t\t");
      scanf("%d",&n);
      n=n-1;
      pickup(n);
      break;
      }
      case 3:
      {
      printf("\n\t Enter which philosopher is thinking\n");
      printf("\t\t");
      scanf("%d",&n);
      n=n-1;
      putdown(n);
      break;
      }
}
printf("\n\t State of Each philosopher\n\n");
printf("\n\t Philosoper No.\t Thinking\t Hungry");
```

```
for(i=0;i<5;i++)
{
      printf("\n\t\t %d\t\t%c\t\t%c\t%c\n",i+1,state[i],self[i],spoon[i]);
}
printf("\n 1.Exit\n 2.Hungry\n 3.Thinking\n");
printf("\n Enter your choice\n");
printf("\t\t");
scanf("%d",&ch);
}}
```

Output:

```
C:\TURBOC3\BIN>TC
                Dining Philosopher Problem
                Initial State of Each Philosopher

        Philosopher No.         Think/Eat       Status          spoon
                1               t               s       s

                2               t               s       s

                3               t               s       s

                4               t               s       s

                5               t               s       s

 1.Exit
 2.Hungry
3.Thinking

 Enter your choice
                2

        Enter which philosopher is hungry


        Enter which philosopher is hungry
                4

        State of Each philosopher


        Philosoper No.  Thinking        Hungry
                1               t               s       s

                2               t               s       s

                3               t               s       n

                4               e               s       n

                5               t               s       s

 1.Exit
 2.Hungry
 3.Thinking

 Enter your choice
```

Result:

The Dining philosophers Problem was implemented and verified.

MEMORY ALLOCATION

Ex.No:8(a) **FIRST FIT MEMORY ALLOCATION**
DATE:

Aim:

To write a C++ program to implement First Fit Memory allocation technique.

Algorithm:

1. Start
2. Declare the no.of free blocks(holes) and their sizes.
3. Get the size of the file to be loaded.
4. Have to allocatet the first blocks that is big enough to hold the file.
5. Start search from the beginning of the set of holes.
6. If hole is large enough is found then stop searching and allocate the hole to the file.
7. Otherwise display file that cannot be allocated.

Program:

```
#include<stdio.h>
#include<string.h>
main()
{
int n,j,i,size[10],sub[10],f[10],m,x,ch,t;
int cho;
printf("\t\t MEMORY MANAGEMENT \n");
printf("\t\t ==================\n");
printf("\tEnter the total no of blocks: ");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
 printf("\n Enter the size of blocks: ");
 scanf("%d",&size[i]);
 }
cho=0;
while(cho==0)
{
printf("\n Enter the size of the file: ");
scanf("%d",&m);
x=0;
for(i=1;i<=n;i++)
{
if(size[i]>=m)
{
printf("\n size can occupy %d",size[i]);
size[i]-=m;
x=i;
break;
}
}
if(x==0)
{
printf("\n\nBlock can't occupy\n\n");
}
printf("\n\nSNO\t\tAvailable block list\n") ;
for(i=1;i<=n;i++)
printf("\n\n%d\t\t\t%d",i,size[i]);
printf("\n\n Do u want to continue.... (0-->yes/1-->no): ");
scanf("%d",&cho);
}}
```

Input:

Enter the total no of blocks: 4

Enter the size of blocks: 50

Enter the size of blocks: 20

Enter the size of blocks: 30

Enter the size of blocks: 40

Enter the size of the file: 25

Output:

```
 Enter the size of blocks: 50
 Enter the size of blocks: 20
 Enter the size of blocks: 30
 Enter the size of blocks: 40
 Enter the size of the file: 25
 size can occupy 50
SNO            Available block list

1                    25
2                    20
3                    30
4                    40
 Do u want to continue.....(0-->yes/1-->no):
2                    20
3                    30
4                    40
 Do u want to continue.....(0-->yes/1-->no): 0
 Enter the size of the file: 28
 size can occupy 30
SNO            Available block list

1                    25
2                    20
3                    2
4                    40
 Do u want to continue.....(0-->yes/1-->no):
```

RESULT:

Thus the memory allocation techniques first fit was implemented.

Ex.No. 8(b) **BEST FIT MEMORY ALLOCATION**
DATE:

Aim:

To write a C program to implement Best Fit Memory allocation technique.

Algorithm:

1. Start
2. Declare the no.of free blocks(holes) and their sizes.
3. Get the size of the file to be loaded.
4. Find the hole that is sufficiently enough for the file.
5. Start search from the beginning of the set of holes.
6. If the hole that is large enough is found then st6op searching and allocate the hole to the file.
7. Otherwise display file cannot be allocated.
8. Stop.

Program:

```
#include<stdio.h>
 main()
{
int n,j,i,size[10],sub[10],f[10],m,x,ch,t;
int cho;

printf("\t\t MEMORY MANAGEMENT \n");
printf("\t\t ==================\n");
printf("\tENTER THE TOTAL NO OF blocks : ");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
 printf("\n Enter the size of blocks : ");
 scanf("%d",&size[i]);
 }
cho=0;
while(cho==0)
{
printf("\n Enter the size of the file : ");
scanf("%d",&m);
for(i=1;i<=n;i++)
{
sub[i]=size[i];
f[i]=i;
}
for(i=1;i<=n;i++)
{
```

```
for(j=i+1;j<=n;j++)
if(sub[i]>sub[j])
{
t=sub[i];
sub[i]=sub[j];
sub[j]=t;
t=f[i];f[i]=f[j];
f[j]=t;
}
}
for(i=1;i<=n;i++)
{
if(size[f[i]]>=m)
{
 printf("size can occupy %d : ",size[f[i]]);
size[f[i]]-=m;
x=i;
break;
}
}
if(x==0)
{
printf("block can't occupy");
}
printf("\n\nSNO\t\t Available Block size\n") ;
for(i=1;i<=n;i++)
printf("\n%d\t\t%d",i,size[i]);
printf("\n\n Do u want to continue.... (0-->yes\t/1-->no)");
scanf("%d",&cho);
```

Input:

ENTER THE TOTAL NO OF blocks : 4

Enter the size of blocks : 50

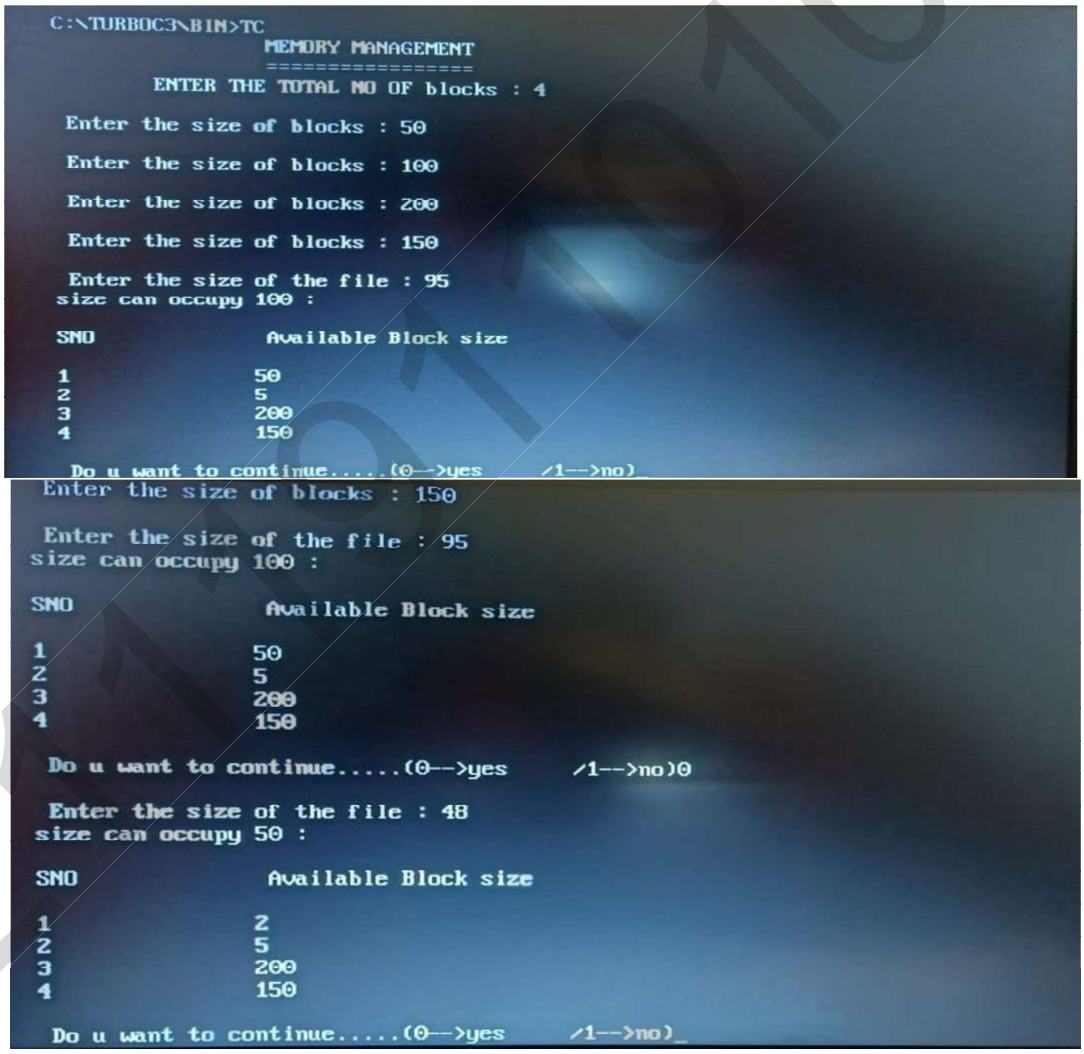
Enter the size of blocks : 100

Enter the size of blocks : 200

Enter the size of blocks : 150

Enter the size of the file : 95

Output:



RESULT:

Thus the best fit memory allocation technique was implemented.

Ex.No. 8(c) **WORST FIT MEMORY ALLOCATION**
DATE:

Aim:

To write a C program to implement Worst Fit Memory allocation technique.

Algorithm:

1. START
2. Declare the number of free blocks(holes) and their size.
3. Get the file size to be loaded.
4. Among the holes find the largest hole that is enough for the file.
5. Start search from the beginning of the set of holes.
6. If the hole is large enough is found then stoop searching and allocate the hole to the file.
7. Otherwise display file cannot be allocated.
8. stop

Program:

```
#include<stdio.h>
int p[10]={0},m=0,x=0,b[10]={0},a[10]={0};
main()
{
int j=0,n=0,pra[10]={0},pro[10]={0},z[10],ch,flag,flag2,sum=0,sum2=0;
int c=0,i=0,k=0;
printf("\n\t\tMEMORY MANAGEMENT POLICIES\n");
printf("\n enter the no of process:\t");
scanf("%d",&n);
printf("\n enter the no of partition:\t");
scanf("%d",&m);
printf("\nprocess information\n");
for(i=0;i<n;i++)
{
printf("\n enter the memory required for process P%d:",i+1);
scanf("\t%d",&a[i]);
pro[i]=a[i];
}
printf("\n memory partition information\n");
for(j=0;j<m;j++)
{
printf("\n enter the block size of block B%d:",j+1);
scanf("\t%d",&p[j]);
pra[j]=p[j];
}
 arrange();
printf("\n process partition\n\n");
```

```
for(i=0;i<n;i++)
{
for(j=0;j<m;j++)
{
if(a[i]<p[j])
{
printf("%d\t%d\n",a[i],p[j]);
p[j]=0;
flag=i;
break;
}
}
if(flag!=i)
printf("%d\t%s\n",a[i],"waiting");
arrange();
}
}
 arrange()
{
int i,j,t;
for(i=0;i<m-1;i++)
for(j=0;j<m-i-1;j++)
{
if(p[j]<p[j+1])
{
t=p[j+1];
p[j+1]=p[j];
p[j]=t;
}}}
```

Input:

MEMORY MANAGEMENT POLICIES

enter the no of process: 5

enter the no of partition: 5

process information

enter the memory required for process P1:212

enter the memory required for process P2:417

enter the memory required for process P3:112

enter the memory required for process P4:321

enter the memory required for process P5:460

memory partition information

enter the block size of block B1:400

enter the block size of block B2:300

enter the block size of block B3:500

enter the block size of block B4:200

enter the block size of block B5:600

Output:

```
C:\TURBOC3\BIN>TC

                MEMORY MANAGEMENT POLICIES

 enter the no of process:       5

 enter the no of partition:     5

process information

 enter the memory required for process P1:212

 enter the memory required for process P2:417

 enter the memory required for process P3:112

 enter the memory required for process P4:321

 enter the memory required for process P5:460

 memory partition information

 enter the block size of block B1:

memory partition information

enter the block size of block B1:400

enter the block size of block B2:300

enter the block size of block B3:500

enter the block size of block B4:200

enter the block size of block B5:600

process partition

212     600
417     500
112     400
321     waiting
460     waiting
```

RESULT:

Thus the Worst fit memory allocation technique was implemented.

PAGE REPLACEMENT

Ex.No: 9(a) **FIRST IN FIRST OUT PAGE REPLACEMENT**
DATE:

Aim:

To write a C Program to implement FIFO page replacement algorithm.

Algorithm:

1. Start
2. Declare the no.of blocks and no. of page references.
3. If the referred page is in the main memory then refer it.
4. If the referred page is not in main memory and free blocks are available then more the referred page to the existing free blocks.
5. Otherwise replace the first page that entered in main memory with the referred page and increment page fault.
6. Stop

Program:

```
#include<stdio.h>
 main()
{
int main_mem,cur=0,i=0,j,fault=0;
static int page[100],page_mem[100],flag,num;
for(i=0;i<100;i++)
      page_mem[i]=-2;
printf("\n\t\t\t\t paging->fifo\n");
printf("\n\n Enter the number of pages in main memory ");
scanf("%d",&main_mem);
printf("\n enter no of page references");
scanf("%d",&num);
for(i=0;i<num;i++)
{
      printf("\n Enter page reference:");
      scanf("%d",&page[i]);
}
printf("\n\t\t\t\t Fifo-> paging \n\n\n");
for(i=0;i<main_mem;i++)
      printf("\t page %d",i+1);
for(i=0;i<num;i++)
{
for(j=0;j<main_mem;j++)
if(page[i]==page_mem[j])
{
flag=1;
break;
}
```

```
        if(!flag)
        {
        page_mem[cur]=page[i];
        fault++;
        }
        printf("\n\n");
        for(j=0;j<main_mem;j++)
              printf("\t%d",page_mem[j]);
        if(!flag&&cur<main_mem-1)
        cur++;
        else if(!flag)
        cur=0;
        flag=0;
        }
        printf("\n\n-2 refers to empty blocks\n\n")
        printf("\n\n No of page faults:%d\n",fault);
}
```

Input:

Enter the number of pages in main memory: 3

enter no of page references: 8

Enter page reference: 2

Enter page reference: 0

Enter page reference: 3

Enter page reference: 0

Enter page reference: 2

Enter page reference: 3

Enter page reference: 5

Enter page reference: 9

Output:

```
 page 1  page 2  page 3
2       -2      -2
2       0       -2
2       0       3
2       0       3
2       0       3
2       0       3
5       0       3
5       9       3
-2 refers to empty blocks
```

RESULT:

Thus the FIFO page replacement algorithm was implemented.

Ex.No: 9(b) LEAST RECENTLY USED PAGE REPLACEMENT

DATE:

Aim: To write a C Program to implement LRU page replacement algorithm.

Algorithm:

1. Start
2. Declare the no.of blocks and no.of page references.
3. If the referred page is in the main memory then refer it.
4. If the referred page is not in main memory and free blocks are available then more the referred page to the existing free blocks.
5. Otherwise replace the page which is not referred for a long time with the referred page and increment page fault.
6. Stop

Program:

```
#include<stdio.h>
#include<stdlib.h>
#define max 100
int frame[10],count[10],cstr[max],tot,nof,fault;
main()
{
 getdata();
push();
}
getdata()
{
      int pno,i=0;
      printf("\n\t\tL R U - Page Replacement Algorithm\n");
      printf("\nEnter No. of Pages in main memory:");
      scanf("%d",&nof);
      printf("\nEnter the no of page references:\n");
      scanf("%d",&pno);
      for(i=0;i<pno;i++)
      {printf("Enter page reference%d:",i);
      scanf("%d",&cstr[i]);
      }
      tot=i;
      for(i=0;i<nof;i++)
      printf("\tpage%d\t",i);
}
push()
{
      int x,i,j,k,flag=0,fault=0,nc=0,mark=0,maximum,maxpos=-1;
      for(i=0;i<nof;i++)
      {
            frame[i]=-1;
            count[i]=mark--;
      }
      for(i=0;i<tot;i++)
      {
            flag=0;
            x=cstr[i];
            nc++;
            for(j=0; j<nof; j++)
               {
                  for(k=0; k<nof;k++)
```

```
                        count[k]++;
                    if(frame[j]==x)
                    {
                        flag=1;
                        count[j]=1;
                        break;
                    }
                }
                if(flag==0)
                {
                    maximum = 0;
                    for(k=0;k<nof;k++)
                    {
                        if(count[k]>maximum && nc>nof)
                        {
                            maximum=count[k];
                            maxpos = k;
                        }
                    }
                    if(nc>nof)
                    {
                        frame[maxpos]=x;
                        count[maxpos]=1;
                     }
                     else
                            frame[nc-1]=x;
                    fault++;
                    dis();
                }
            }
            printf("\nTotal Page Faults :%d",fault);
        }
        dis()
        {
            int i=0;
            printf("\n\n");
            while(i<nof)
            {
                printf("\t%d\t",frame[i]);
                i++;
            }
}
```

Input:

Enter the number of pages in main memory: 3

Enter the no of page references: 8

Enter page reference0: 0

Enter page reference1: 1

Enter page reference2: 2

Enter page reference3: 1

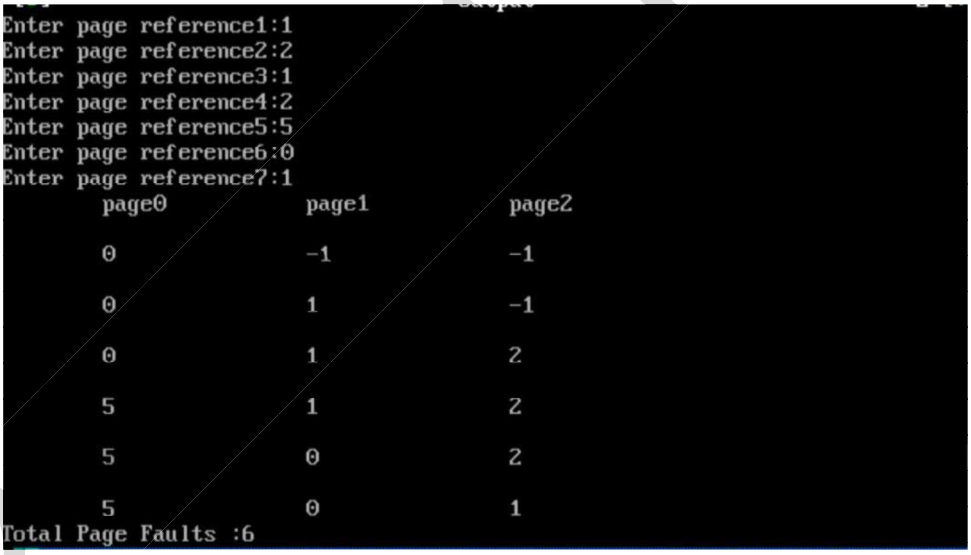
Enter page reference4: 2

Enter page reference5: 5

Enter page reference6: 0

Enter page reference7: 1

Output:



Result:

Thus the page replacement algorithm LRU was implemented.