BE semester-5 Computer Engineering and Information Technology
Academic year – 2018-19 (Odd Semester)

Subject Name: Operating System
Subject Code: CE-503 / IT-503

List of Practicals

Sr. No	Title	Date	Initial of faculty	Remarks
1	To Study and hands on upon various UNIX, VI Editor and Shell's Commands			
2	Implementation of Shell's Scripts using basic UNIX commands and control statements			
3	Implementation of Shell's Scripts demonstrating Loops			
4	Implementation of Shell's Scripts demonstrating the use of String operations & Command line Arguments			
5	Implementation of FCFS (First Come First Serve) CPU Scheduling algorithm			
6	Implementation of SJF (Shortest Job First) CPU Scheduling algorithm			
7	Implementation of RR(Round Robin) CPU Scheduling algorithm			
8	Implementation of Priority CPU Scheduling algorithm			
9	Implementation of First- Fit Memory Allocation algorithm			
10	Implementation of Best-Fit Memory Allocation algorithm			
11	Implementation of FIFO Page Replacement algorithm			
12	Implementation of LRU Page Replacement algorithm by Stack method			
13	Implementation of Optical Page Replacement algorithm			

PRACTICAL – 1

AIM: To Study and hands on upon various UNIX, VI Editor and Shell's Commands.

1) cal:
   a) Print the calendar of any year in the range 1 to 9999
   b) The system not stores these calendar in memory but generate through the calculation when we invoke command

   Options

   c) If we want to see the calendar of any year then
      - $ Cal 2006
   d) If we want to see the calendar for particular month then
      - $ Cal 3 2006

2) who:-
   a) List all the users who are currently logged in, with their terminal and their log in time.

   Option:

   b) $ who am i:

      It displays our login name, terminal name and log in time.

3) ls:

It list all the filenames in alphabetical order which are present in current directory.

Options:

ls –a : display hidden files also

ls [aeiou]* : display all files which has first letter any from the bracket.

ls –l : It list the files with 7 columns having information about it's permission ,number of links, owner name, group name, size of file in bytes, date and time when file was last modified & filename.

4) cat:

   a) It is used to create a new file. Press ctrl + d to indicate EOF.

   b) $ cat > test

      Crete a filename with test

   c) $ cat test

      Display the content of test file

   c) $ cat file1 file2 > file3

      This would create a new file and content of file1 & file2 are copied into it.

5) wc:

   a) It counts the number of lines , words and character from the specified file or files.

   b) Wc –l filename : display the number of lines

   c) Wc-w filename: display the number of words

   d) Wc-c filename: display the number of character.

6) pipe :

   a) We can join the commands using a pipe, means it sends the output of one command as input of another.

   b) $ ls | wc-l

      Here the output of ls becomes the input to wc which promptly counts the number of lines it receives as input and display this count on the Screen.

7) echo:

   It is used to display output. If we use " " in echo then the shell treats the content within " " as a single string to be displayed.

8) exit:

It will stop the execution of shell script and exit form the program.

9) type:

It returns the location of the given command

type command

10) man:

a) This provide the help manual for every commands
b) $ Man cd

It display the help manual for cd command.

11) date:

a) It display date on screen
b) $ date

It display Wed mar 08 04:40:10 IST 2006

Where IST means Indian standard time.

c) The output of the date command can be modified by a variety of Switch.

For e.g. $date '+DATE:%d-%m-%y%n TIME : %H:%M:%S'

Where %d, %m, %y indicate day, month, year and %h, %m, %s Indicate hour, minute and second.

12) bc:

a) Calculator can be invoked by typing bc at shell prompt.
b) $ bc

```
Sqrt(25)
 5
```

13) script:

a) When you write script on the prompt, all the commands which you run after it will be stored in a file named type script.
b) You can close the file by writing exit on the prompt.

14) ln:

a) It is used to create another link for a same file.

b) $ ln file1 file2

This command establish one more link for the file1 in the form of the name file2.

15) nl:

a) This command displays the content of the file with the line number.

b) $ nl abc

This command display the content of files abc with line number before every line.

16) head:

a) It helps in viewing lines at the beginning of file.

b) If we not specify anything then this command display first 10 lines

For e.g. $ head abc

c) $ head -15 abc

This command display first 15 lines of file abc.

17) tail:

It helps in viewing lines from the end of file
For e.g. $ tail -20 abc will display the last 20 lines from the file abc.

18) grep:

a) grep stands for “globally search a regular expression and print it.”

This command search for the specified input fully for a match with The supplied pattern and display it.

b) $ grep xy abc

This would search the word xy in a file called abc.

19) sort:

a) It is used to sort the content of a file.

b) $ sort abc

It shows the content of file abc in sorted format

c) $ sort file1 file2 file3

This will sort the contents of several files at once.

20) cut:

a) It cuts or picks up a given number of character or fields from the specified file.

b) suppose we have a large database of student information from that we Want only specific field say name (second field) and division (fifth field) then we can write the cut command as

```
$ cut –c 2 ,7  empinfo
```

If we want to view the field 2 through 7 then

```
$ cut –f1 -d “ “  empinfo
```

21) passwd:

a) You can change your password whenever you are logged in by using the passwd command as below.

```
$ passwd
```

This command asks you to enter old password to prove that you are the authorized person and then after it ask for the new password.

22) pwd:

Pwd stands for ‘present working directory’. When you write pwd on prompt it will display current working directory.

For e.g. `$ pwd`

23) cd:

a) This command is used to change the directory.

b) `$ cd newdir`

This command would take you in new directory.

c) `$ cd`

when given without any argument is interpreted by the shell as a request to change over the current user’s home directory.

24) mkdir:

a) This command is used to create a new directory.

b) `$ mkdir xyz`

This will create directory named xyz.

c) `$ mkdir –p  xyz/abc`

-p option tell to create xyz directory first and then create directory abc.

25) rm:

This command removes the given file or files supplied to it.

a) `$ rm -i file1`

where –i is a switch , removes file interactively; means you are asked for confirmation before deleting the file.

b) `$  rm -r dir1`

This command recursively removes all content of dir1 and also dir1 itself

26) cp:

This command is used to copy a file.

a) `$ cp file1 file2`

This will copy file1 in to file2. if file2 does not exit then it will be created.

b) We can copy more than one file into a directory.

`$ cp file1 file2 dir1`

27) mv:

This command is used to rename the file

a) `$ mv file1 file2`

This command renames the file1 to file2

b) `$mv file1 file2 dir1`

This command moves the file1 and file2 from its original location to the directory dir.

28) cmp:

The cmp utilities compares two files of any type and writes the result to the Standard o/p

29) diff

The diff utility used to compare the contents of two files.

```
diff file1 file2
```

30) ulimit

unix system has resource limits such as limits on number of processes, maximum allowed file size , etc

a) ulimit -a
b)ulimit -f 121212

<u>Common vi editor command list</u>

To insert new text	esc + i (You have to press 'escape' key then 'i')
To save file	esc + : + w (Press 'escape' key then 'colon' and finally 'w')
To save file with file name (save as)	esc + : + w "filename"
To quit the vi editor	esc + : + q
To quit without saving	esc +: + q!
o save and quit vi editor	esc + : + wq
To search for specified word in forward direction	esc + /word (Press 'escape' key, type /word-to-find, for e.g. to find word 'shri', type as/shri)
To continue with search	n
To search for specified word in backward direction	esc + ?word (Press 'escape' key, type word-to-find)
To copy the line where cursor is located	esc + yy
To paste the text just deleted or copied at the cursor	esc + p
To delete entire line where cursor is located	esc + dd
To delete word from cursor position	esc + dw
Undo last change	u
Undo all changes to the entire line	U

Write after cursor (goes into insert mode)	a
Write at the end of line (goes into insert mode)	A
Terminate insert mode	ESC
Open a new line (goes into insert mode)	o
Three lines delete	3dd
To Find all occurrence of given word and Replace them globally without confirmation	esc + :$s/word-to-find/word-to-replace/g For. e.g. :$s/mumbai/pune/g Here word "mumbai" is replace with "pune"
To Find all occurrence of given word and Replace then globally with confirmation	esc + :$s/word-to-find/word-to-replace/cg
To run shell command like ls, cp or date etc within vi	esc + :!shell-command For e.g. :!pwd

PRACTICAL – 2

AIM: Implementation of Shell’s Scripts using basic UNIX commands and control statements

1. Write shell script to take cost price and selling price of an item is input through the keyboard, and determine whether the seller has made profit or incurred loss.

Program:-

```
echo “Enter cost price of item:”;
read c;
echo “Enter selling price of item:”;
read s;
if [ $s –eq $c ]
then
echo “No profit or No loss has incurred.”;
elif [ $s –lt $c ]
then
echo “Loss of Rs. `expr $c - $s` has incurred.”;
else
echo “Profit of Rs. `expr $s - $c` has incurred.”;
fi
```

Output:-

```
Enter cost price of item:
2000

Enter selling price of item:
2250

Profit of Rs. 250 has incurred.
```

2. Write a script to find largest number out of two inputted number.

Program:-

```
echo “Enter two numbers:”;
read a;
read b;
```

```
if [ $a –gt $b ]
then
echo “$a is a largest number.”;
elif [ $a –lt $b ]
then
echo “$b is a largest number.”;
else
echo “Both number are equal number.”;
fi
```

Output:-

```
Enter two numbers:
12
10

12 is a largest number.
```

3. Write a script to enter any year through the keyboard and to determine whether the year is leap year or not.

Program:-

```
echo “Enter the year in 4 digits:”;
read year;
if  [ $year –gt 1000 –a $year –lt 9999 ]
then
if  [  `expr $year % 4` -eq 0 ]
then
echo “The year $year is a leap year.”;
else
echo “The year $year is not a leap year.”;
fi
else
echo “Invalid year format.”;
fi
```

Output:-

```
Enter the year in 4 digits:
2014
```

The year 2014 is not a leap year.

4. Write a script on write a menu which has following option:

1). Present working directory
2). Calendar
3). List of user who have currently logged in
4). Exit
Make the use of case statement.

Program:-

```
echo “Menu:1. Present working directory.\n2. Calendar.\n3. List of user who have currently logged in.\n4. Exit”

echo “Enter your choice:”

read choice

case $choice in

1) pwd ;;
2) cal ;;
3) who am i ;;
4) exit ;;
*)  echo “Invalid Choice.”

esac
```

Output:-

```
Manu:

1. Present working directory.
2. Calendar.
3. List  of user who have currently logged in.
4. Exit

Enter your choice:
2

April 2014

Su   Mo  Tu   We  Th   Fr    Sa

1     2     3     4     5
6     7     8     9     10    11    12
```

```
13   14   15   16   17   18   19
20   21   22   23   24   25   26
27   28   29   30
```

5. Write a script to accept number and perform addition, subtraction, multiplication & division.

Program:-

```
echo “Menu:\n 1.Addition\n 2. subtraction\n 3.Multiplication \n 4. Division \n”;
echo “Enter your choice:”;
read choice;
if [ $choice –lt 5 ]
then
echo “Enter a & b :”;
read a;
read b;
else
fi
case $choice in
1) echo “Addition of this two number: $sum = `expr $a + $b`” ;;
2) echo “Subtraction of this two number: $sub = `expr $a - $b`” ;;
3) echo “Multiplication of this two number: $mul = `expr $a \* $b`” ;;
4) echo “Division of this two number: $div = `expr $a / $b`” ;;
*)  echo “Invalid Choice” ;;
esac
```

Output:-

```
Menu:
1.Addition
2. subtraction
3.Multiplication
4. Division

Enter your choice:
1

Enter a & b :
10
20
```

```
Addition of this two number:
30
```

6. Write a script for accept a string and check whether it is file or directory if it exists. (null string is not allowed).

Program:-

```
echo “Enter string:”;
read str;
if [ -d $str ]
then
echo “ It is a directory.”;
elif [ -f $str ]
then
echo “ It is a file.”;
else
echo “The file or directory does not exist”;
fi
```

Output:-

```
Enter string:
abc
It is a directory.
```

7. Write a script to accept file name & display last modification time if file exists otherwise display appropriate massage.

Program:-

```
echo “Enter file name :”
read fn;
if [ -f $fn ]
then
        echo “Last modification time:”;
        date -r $fn | date +%T
else
        echo “File does not exist.”;
fi
```

Output:-

```
Enter file name :
abc.sh


Last modification time:
14:34:67
```

8. Write a script to display the name of all executable files in the given directory.

Program:-

```
echo “Enter directory name:”;
read dir;
if [ -d $dir ]
then
echo “Enter extension you want to search: ”;
read exe;
ls -l $exe
else
        echo “Directory doesn't exists.”;

fi
```

Output:-

```
Enter directory name:
abc


Enter extension you want to search:
*.c
-rw-rw-r-- 1 uspa uspa 85 May 13 13:26 a1.c

-rw-rw-r-- 1 uspa uspa 59 May 20 12:49 a2.c
```

9. Write a script to display the date, time & a welcome massage (like Good Morning etc.) the time should be displayed with ‘a.m’ or ‘p.m’ & not in 24 hours notation.

Program:-

```
echo “Current Time: date +%T”;
HH =`date +%H`
if [ $HH -gt 5 -a $HH -lt 10 ]
then
echo “Good Morning”;

elif [ $HH -ge 10 -a $HH -lt 16 ]
then
echo "Good Afternoon"
elif [ $HH -ge 16 -a $HH -lt 20 ]
then
echo "Good Evening";
elif [ $HH -ge 20 -a $HH -lt 5 ]
then
echo “Good Night”
fi
```

Output:-

```
Current Time:
07:36:32

Good Morning
```

10. Write a script to display the directory in the descending order of the size of each file.

Program:-

```
echo "Enter the file extention :"
read dir
ls -lS $dir
```

Output:-

```
Enter the file extention :
*.sh
-rw-r--r-- 1 ubuntu ubuntu 510 Apr 22 11:41 vishal1.sh
-rw-r--r-- 1 ubuntu ubuntu 420 Apr 22 13:44 dir.sh
-rw-r--r-- 1 ubuntu ubuntu 265 Apr 22 11:35 vishal.sh
-rw-r--r-- 1 ubuntu ubuntu 214 Apr 22 14:17 aaa.sh
-rw-r--r-- 1 ubuntu ubuntu 185 Apr 22 12:31 v3.sh
```

```
-rw-r--r-- 1 ubuntu ubuntu 138 Apr 22 11:52 time.sh
-rw-r--r-- 1 ubuntu ubuntu  83 Apr 22 14:46 aaaa.sh
```

11. Write a script to make following file and directory management operation.

1) **Display current directory**
2) **List directory**
3) **Make directory**
4) **Change directory**
5) **Copy directory**
6) **Rename a file**
7) **Delete a file**
8) **Edit a file**

Program:-

```
echo “Manu:\n1. Display current directory\n2. List directory\n3. Make directory\n4. Change directory\n5. Copy directory\n6. Rename a file\n7. Delete a file and Edit a file”
echo “Enter your choice: ”
read choice
if [ $choice –eq 1 ];
then
echo “Current directory is ”
pwd
elif [ $choice –eq 2 ];
then
echo “List of directory:”
ls
elif [ $choice –eq 3 ];
then
echo “Enter new directory name to create:”
read dir
mkdir $dir
echo “$dir directory is created successfully.”
elif [ $choice –eq 4 ];
then
echo “Enter directory name to change: “
read cdir
cd $cdir
elif [ $choice –eq 5 ];
then
echo “Enter file name to copy:”
```

```
read file
echo “Enter directory name to which file to be copied:”
read dir1
cp $file $dir1
elif [ $choice –eq 6 ];
then
echo “Enter file name to rename: ”
read rname
echo “Enter new name:”
read nname
mv $rname $nname
elif [ $choice –eq 7 ]
then
echo “Enter file name to delete:”
read dfile
rm $dfile
elif [ $choice –eq 8 ];
then
echo “Enter file name to edit:”
read efile
echo “Enter new data to write:”
read newdata
echo $newdata > $efile
fi
```

Output:-

```
Menu:

1. Display current directory
2. List directory
3. Make directory
4. Change directory
5. Copy directory
6. Rename a file
7. Delete a file
8. Edit a file

Enter your choice:
1
/home/ubuntu
```

12. Write a script which reads a text file & output the following count of character , words & lines

Program :-

```
echo "Enter the file name "
read fname
echo "Total character "
 wc -c $fanme
echo "Total words "
wc -w $fname
echo "Total line "
wc -l $fanme
```

Output:-

```
Enter the file name: abc.sh

Total character
243

Total words
67

Total line
22
```

PRACTICAL – 3

AIM: Implementation of Shell’s Scripts demonstrating Loops

1. Write a script for accept a five-digit number through keyword, then reverse this five-digit number.

Program:-

```
echo “Enter five-digit number:”;
read n;
rev=0; rem=0; temp=0;
while [ $n -ne 0 ]
do
rem=`expr $n % 10`;
temp=`expr $rev \* 10`;
rev=`expr $temp + $rem`;
 n=`expr $n / 10`;
done
echo “Reverse number is $rev.”;
```

Output:-

```
Enter five-digit number:
12345
```

A Shell Script To Print A Number In Reverse Order:

```
echo "Enter a number"
read n
sd=0
rev=0
while [ $n -gt 0 ]
do
    sd=$(( $n % 10 ))
    rev=`expr $rev \* 10 + $sd`
    n=$(( $n / 10 ))
done
echo "Reverse number of entered digit is $rev"
```

```
Reverse number is 54321
```

2. Write a script to generate sum of all odd number between 1 to 20.

Program:-

```
echo “Sum of all odd number between 1 to 20:”;
sum = 0;
i = 1;
while [ $i –le 20]
do
rem=`expr $i % 2`;
if [ $rem -eq 1]
then
sum = `expr $i + $sum`
fi
done
i=`expr $i + 1`;
echo “ $sum ”
```

Output:-

```
Sum of all odd number between 1 to 20:

100
```

3. Write a script to demonstrate the use of for.. loop

Program:

```
$ cat for1.sh
i=0
for day in Mon Tue Wed Thu Fri
do
 echo "Weekday $((i++)) : $day"
done
```

Output:

```
$ ./for1.sh
Weekday 1 : Mon
Weekday 2 : Tue
Weekday 3 : Wed
Weekday 4 : Thu
Weekday 5 : Fri
```

4. Write a script to fetch the data from file & display data.

Program:-

```
while read line
do
echo $line;
done
```

Output:- sh file2.sh > file1.sh

```
LDRP-ITR, Gandhinagar.
```

5. write a shell script to print any two pattern.

```
# Program in Bash to
# print pyramid

# Static input to the
# number
p=7;

for((m=1; m<=p; m++))
do
   # This loop print spaces
   # required
   for((a=m; a<=p; a++))
   do
     echo -ne " ";
   done

   # This loop print the left
   # side of the pyramid
   for((n=1; n<=m; n++))
   do
     echo -ne "#";
   done

   # This loop print right
   # side of the pryamid.
   for((i=1; i<m; i++))
   do
```

```
      echo -ne "#";
    done

    # New line
    echo;
done
```

```
       #
      ###
     #####
    #######
   #########
  ###########
 #############
```

6.write a script to check whether entered number is palindrome or not.

Reverse logic then if else reverse == original no than palindrome

7. write a script to print the Fibonacci series.

8. write a shell script to print table of a given number.

```
echo "Enter a Number"
read n
i=1

while [ $i -le 10 ]
do
	echo " $n x $i = $(( n * i ))"
	i=$(( i + 1 ))
done
```

PRACTICAL – 4

AIM: Implementation of Shell's Scripts demonstrating the use of String operations & Command line Arguments

String Length

```
$ vi len.sh
```

```
var="Welcome to the geekstuff"
echo ${#var}
```

Output:

```
sh len.sh

24
```

Substring of String

Program:

```
$ cat substr.sh

#! /bin/bash


var="Welcome to the geekstuff"

echo ${var:15}

echo ${var:15:4}
```

Output:

```
$ ./substr.sh

geekstuff

geek
```

String Concatenate

Program:

```
a='hello'
```

```
b='world'

c=$a$b

echo $c
```

Output

```
> helloworld
```

String Compare

Program:

```
Sourcesystem="ABC"

if [ 'XYZ' -eq "$Sourcesystem" ]; then

   echo "Sourcesystem Matched"

else

   echo "Sourcesystem is NOT Matched $Sourcesystem"

fi;


echo Sourcesystem Value is  $Sourcesystem ;
```

Output:

Command-line Arguments:

Program:

```
$ vi myscript
#!/bin/bash

echo "First arg: $1"
echo "Second arg: $2"
```

Output:

Run by writing command:

```
$ ./myscript hello world

First arg: hello
Second arg: world
```

PRACTICAL – 5

AIM: Implementation of FCFS (First Come First Serve) CPU Scheduling

Algorithm

1. Define an array of structure *process* with members *pid*, *btime, wtime* & *ttime*.
2. Get length of the ready queue, i.e., number of process (say *n*)
3. Obtain *btime* for each process.
4. The *wtime* for first process is 0.
5. Compute *wtime* and *ttime* for each process as:
   a. $wtime_{i+1} = wtime_i + btime_i$
   b. $ttime_i = wtime_i + btime_i$
6. Compute average waiting time *awat* and average turnaround time *atur*
7. Display the *btime, ttime* and *wtime* for each process.
8. Display GANTT chart for the above scheduling
9. Display *awat* time and *atur*
10. Stop

Result

Thus waiting time & turnaround time for processes based on FCFS scheduling was computed and the average waiting time was determined.

Program

```
/* FCFS Scheduling  - fcfs.c */

#include <stdio.h>

struct process
{
   int
   pid;
   int
   btime;
   int
   wtime;
   int
   ttime;
} p[10];

main()
{
   int i,j,k,n,ttur,twat;
   float awat,atur;
```

```
printf("Enter no. of process : ");
scanf("%d", &n);
for(i=0; i<n; i++)
{
   printf("Burst time for process P%d (in ms) :
   ",(i+1));
   scanf("%d", &p[i].btime);
   p[i].pid = i+1;
}

p[0].wtime = 0;
for(i=0; i<n; i++)
{
   p[i+1].wtime = p[i].wtime + p[i].btime;
   p[i].ttime = p[i].wtime + p[i].btime;
}
ttur = twat = 0;
for(i=0; i<n; i++)
{
   ttur += p[i].ttime;
   twat += p[i].wtime;
}
awat = (float)twat / n;
atur = (float)ttur / n;

printf("\n       FCFS Scheduling\n\n");
for(i=0; i<28; i++)
   printf("-
      ");
printf("\nProcess B-Time T-Time W-Time\n");
for(i=0; i<28; i++)
   printf("-
      ");
for(i=0; i<n; i++)
   printf("\n  P%d\t%4d\t%3d\t%2d",
           p[i].pid,p[i].btime,p[i].ttime,p[i].wt
           ime);
printf("\n");
for(i=0; i<28; i++)
   printf("-
      ");

printf("\n\nGANTT Chart\n");
printf("-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
   printf("-");
printf("\n");
printf("|");
for(i=0; i<n;
i++)
{
   k = p[i].btime/2;
   for(j=0; j<k; j++)
```

```
         printf(" ");
      printf("P%d",p[i].pid);
      for(j=k+1; j<p[i].btime;
      j++)
         printf("
            ");
      printf("|");
   }
   printf("\n");
   printf("-");
   for(i=0; i<(p[n-1].ttime + 2*n); i++)
      printf("-");
   printf("\n");
   printf("0");
   for(i=0; i<n;
   i++)
   {
      for(j=0; j<p[i].btime; j++)
         printf(" ");
      printf("%2d",p[i].ttime);
   }

   printf("\n\nAverage waiting time     : %5.2fms", awat);
   printf("\nAverage turn around time : %5.2fms\n", atur);
}
```

Output

```
$ gcc fcfs.c

$./a.out
Enter no. of process : 4
Burst time for process P1 (in ms) : 10
Burst time for process P2 (in ms) : 4
Burst time for process P3 (in ms) : 11
Burst time for process P4 (in ms) : 6

          FCFS
       Scheduling

------------------------
---- Process B-Time T-
Time W-Time
----------------------------
    P1      10      10
            0
    P2       4      14
           10
    P3      11      25
           14
    P4       6      31
           25
```

```
----------------------------

GANTT Chart
----------------------------------------
|     P1     |  P2 |     P3      |   P4  |
----------------------------------------
0          10    14           25      31

Average waiting time     : 12.25ms
Average turnaround time : 20.00ms
```

PRACTICAL – 6

AIM: Implementation of SJF (Shortest Job First) CPU Scheduling

Algorithm

1. Define an array of structure *process* with members *pid*, *btime*, *wtime* & *ttime*.
2. Get length of the ready queue, i.e., number of process (say *n*)
3. Obtain *btime* for each process.
4. *Sort* the processes according to their *btime* in ascending order.

a. If two process have same *btime*, then FCFS is used to resolve the tie.

5. The *wtime* for first process is 0.
6. Compute *wtime* and *ttime* for each process as:

a. $wtime_{i+1} = wtime_i + btime_i$

b. $ttime_i = wtime_i + btime_i$

7. Compute average waiting time *awat* and average turn around time *atur*.
8. Display *btime*, *ttime* and *wtime* for each process.
9. Display GANTT chart for the above scheduling
10. Display *awat* and *atur*
11. Stop

Result

Thus waiting time & turnaround time for processes based on SJF scheduling was computed and the average waiting time was determined.

Program

```
/* SJF Scheduling – sjf.c */

#include <stdio.h>

struct process
{
int pid; int
btime; int
wtime; int
ttime;
} p[10], temp;

main()
{
int i,j,k,n,ttur,twat;
float awat,atur;

printf("Enter no. of process : ");
scanf("%d", &n);
for(i=0; i<n; i++)
{
      printf("Burst time for process P%d (in ms) : ",(i+1));
      scanf ("%d", &p[i].btime);
```

```
     p[i].pid = i+1;
}

for(i=0; i<n-1; i++)
{
     for(j=i+1; j<n; j++)
     {
          if((p[i].btime > p[j].btime) ||
          (p[i].btime == p[j].btime && p[i].pid > p[j].pid))
          {
               temp = p[i];
               p[i]=p[j];
               p[j] = temp;
          }

     }
}

p[0].wtime = 0;
for(i=0; i<n; i++)
{
p[i+1].wtime = p[i].wtime + p[i].btime;
p[i].ttime = p[i].wtime + p[i].btime;
}
ttur = twat = 0;
for(i=0; i<n; i++)
{
ttur += p[i].ttime;
twat += p[i].wtime;
}
awat = (float)twat / n;
atur = (float)ttur / n;

printf("\n       SJF Scheduling\n\n");
for(i=0; i<28; i++)
  printf("-");
printf("\nProcess B-Time T-Time W-Time\n");
for(i=0; i<28; i++)
  printf("-");
for(i=0; i<n; i++)
printf("\n  P%-4d\t%4d\t%3d\t%2d",
p[i].pid,p[i].btime,p[i].ttime,p[i].wtime);
printf("\n");
for(i=0; i<28; i++)
  printf("-");

printf("\n\nGANTT Chart\n");
printf("-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
printf("-");
printf("\n|");
for(i=0; i<n; i++)
{
k = p[i].btime/2;
for(j=0; j<k; j++)
printf(" ");
printf("P%d",p[i].pid); for(j=k+1;
j<p[i].btime; j++)
```

```
    printf(" ");
printf("|");
}
printf("\n-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
printf("-");
printf("\n0");
for(i=0; i<n; i++)
{
for(j=0; j<p[i].btime; j++)
printf(" ");
printf("%2d",p[i].ttime);
}

printf("\n\nAverage waiting time     : %5.2fms", awat);
printf("\nAverage turn around time : %5.2fms\n", atur);
}
```

Output

```
$ gcc sjf.c

$./a.out
Enter no. of process : 5
Burst time for process P1 (in ms) : 10
Burst time for process P2 (in ms) : 6
Burst time for process P3 (in ms) : 5
Burst time for process P4 (in ms) : 6
Burst time for process P5 (in ms) : 9

   SJF Scheduling

---------------------------
- Process B-Time T-Time W-
Time
----------------------------
   P3   5         5        0
   P2      6     11        5
   P4      6     17   11
   P5      9     26   17
   P1     10     36   26
----------------------------

GANTT Chart
-----------------------------------------------
|  P3  |   P2  |   P4  |    P5    |     P1    |
-----------------------------------------------
0      5      11      17         26          36

Average waiting time     : 11.80ms
Average turnaround time : 19.00ms
```

PRACTICAL – 7

AIM: Implementation of Round Robin (RR) CPU Scheduling

Algorithm

1. Get length of the ready queue, i.e., number of process (say *n*)
2. Obtain *Burst* time B_i for each processes P_i.
3. Get the *time slice* per round, say *TS*
4. Determine the number of rounds for each process.
5. The wait time for first process is 0.
6. If $B_i > TS$ then process takes more than one round. Therefore turnaround and waiting time should include the time spent for other remaining processes in the same round.
7. Calculate *average* waiting time and turnaround time
8. Display the GANTT chart that includes
    a. order in which the processes were processed in progression of rounds

    b. Turnaround time T_i for each process in progression of rounds.
9. Display the *burst* time, *turnaround* time and *wait* time for each process (in order of Rounds they were processed).
10. Display *average* wait time and turnaround time
11. Stop

Result

Thus waiting time and turnaround time for processes based on Round robin scheduling was computed and the average waiting time was determined.

Program

```
/* Round robin scheduling - rr.c */

#include <stdio.h>

main()
{
int i,x=-1,k[10],m=0,n,t,s=0;
int a[50],temp,b[50],p[10],bur[10],bur1[10];
int wat[10],tur[10],ttur=0,twat=0,j=0;
float awat,atur;

printf("Enter no. of process : ");
scanf("%d", &n);
for(i=0; i<n; i++)
{
      printf("Burst time for process P%d : ", (i+1));
      scanf("%d", &bur[i]);
      bur1[i] = bur[i];
}
printf("Enter the time slice (in ms) : ");
scanf("%d", &t);
```

```
for(i=0; i<n; i++)
{
      b[i] = bur[i]/t;
      if((bur[i]%t)!= 0)
            b[i]+=1;
      m += b[i];
}

printf("\n\t\tRound Robin Scheduling\n");

printf("\nGANTT Chart\n");
for(i=0; i<m; i++)
   printf("--------");
printf("\n");

a[0] = 0;
while(j < m)
{
      if(x == n-1)
       x = 0;
      else
            x+
      +;
      if(bur[x] >= t)
      {
            bur[x] -= t;
            a[j+1] = a[j] + t;

            if(b[x] == 1)
            {
                  p[s] = x;
                  k[s] = a[j+1];
                  s++;
            }
            j++;
            b[x] -= 1;
            printf("  P%d    |", x+1);
      }
      else if(bur[x] != 0)
      {
            a[j+1] = a[j] + bur[x];
            bur[x] = 0;
            if(b[x] == 1)
            {
                  p[s] = x;
                  k[s] = a[j+1];
                  s++;
            }
            j++;
            b[x] -= 1;
            printf("  P%d  |",x+1);
      }
}

printf("\n");
for(i=0;i<m;i++)
```

```
printf("--------");
printf("\n");

for(j=0; j<=m; j++)
      printf("%d\t", a[j]);

for(i=0; i<n; i++)
{
      for(j=i+1; j<n; j++)
      {
            if(p[i] > p[j])
            {
                  temp = p[i];
                  p[i] = p[j];
                  p[j] = temp;

                  temp = k[i];
                  k[i] =k[j];
                  k[j] = temp;
            }
      }
}

for(i=0; i<n; i++)
{
wat[i] = k[i] - bur1[i];
tur[i] = k[i];
}
for(i=0; i<n; i++)
{
ttur += tur[i];
twat += wat[i];
}

printf("\n\n");
for(i=0; i<30; i++)
printf("-");
printf("\nProcess\tBurst\tTrnd\tWait\n");
for(i=0; i<30; i++)
   printf("-");
for (i=0; i<n; i++)
printf("\nP%-4d\t%4d\t%4d\t%4d", p[i]+1, bur1[i],
tur[i],wat[i]);
printf("\n");
for(i=0; i<30; i++)
   printf("-");

awat = (float)twat / n;
atur = (float)ttur / n;
printf("\n\nAverage waiting time      : %.2f ms", awat);
printf("\nAverage turn around time : %.2f ms\n", atur);
}
```

Output

```
$ gcc rr.c

$ ./a.out
Enter no. of process : 5
Burst time for process P1 : 10
Burst time for process P2 : 29
Burst time for process P3 : 3
Burst time for process P4 : 7
Burst time for process P5 : 12
Enter the time slice (in ms) : 10

Round Robin Scheduling

GANTT Chart
--------------------------------------------------------------
P1    |  P2   |  P3  |  P4  |  P5  |  P2   |  P5   |   P2  |
--------------------------------------------------------------
0    10      20     23      30      40      50      52      61

-----------------------------
- Process Burst   Trnd
Wait
------------------------------
 P1      10       10          0
 P2      29       61     32
 P3    3          23     20
 P4    7          30     23
 P5      12       52     40
------------------------------

Average waiting time     : 23.00 ms
Average turn around time : 35.20 ms
```

PRACTICAL – 8

AIM: Implementation of Priority CPU Scheduling Algorithm

Algorithm

1. Define an array of structure *process* with members *pid*, *btime*, *pri*, *wtime* & *ttime*.
2. Get length of the ready queue, i.e., number of process (say *n*)
3. Obtain *btime* and *pri* for each process.
4. *Sort* the processes according to their *pri* in ascending order.

a. If two process have same *pri*, then FCFS is used to resolve the tie.

5. The *wtime* for first process is 0.
6. Compute *wtime* and *ttime* for each process as:

a. $wtime_{i+1} = wtime_i + btime_i$

b. $ttime_i = wtime_i + btime_i$

7. Compute average waiting time *awat* and average turn around time *atur*
8. Display the *btime*, *pri*, *ttime* and *wtime* for each process.
9. Display GANTT chart for the above scheduling
10. Display *awat* and *atur*
11. Stop

Result

Thus waiting time & turnaround time for processes based on Priority scheduling was computed and the average waiting time was determined.

Program

```
/* Priority Scheduling  - pri.c */

#include <stdio.h>

struct process
{
int pid; int
btime; int
pri; int
wtime; int
ttime;
} p[10], temp;

main()
{
int i,j,k,n,ttur,twat;
float awat,atur;

printf("Enter no. of process : ");
scanf("%d", &n);
for(i=0; i<n; i++)
{
```

```
printf("Burst time for process P%d (in ms) : ", (i+1));
scanf("%d", &p[i].btime);
printf("Priority for process P%d : ", (i+1));
scanf("%d", &p[i].pri);
p[i].pid = i+1;
}

for(i=0; i<n-1; i++)
{
for(j=i+1; j<n; j++)
{
if((p[i].pri > p[j].pri) ||
(p[i].pri == p[j].pri && p[i].pid > p[j].pid) )
{
temp  =  p[i];  p[i]  =
p[j]; p[j] = temp;
}

}
}
p[0].wtime = 0;
for(i=0; i<n; i++)
{
p[i+1].wtime = p[i].wtime + p[i].btime;
p[i].ttime = p[i].wtime + p[i].btime;
}

ttur = twat = 0;
for(i=0; i<n; i++)
{
ttur += p[i].ttime;
twat += p[i].wtime;
}
awat = (float)twat / n;
atur = (float)ttur / n;

printf("\n\t Priority Scheduling\n\n");
for(i=0; i<38; i++)
   printf("-");
printf("\nProcess B-Time Priority T-Time  W-Time\n");
for(i=0; i<38; i++)
   printf("-");
for (i=0; i<n; i++)
printf("\n  P%-4d\t%4d\t%3d\t%4d\t%4d",
p[i].pid,p[i].btime,p[i].pri,p[i].ttime,p[i].wtime);
printf("\n");
for(i=0; i<38; i++)
   printf("-");

printf("\n\nGANTT Chart\n");
printf("-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
printf("-");
printf("\n|");
for(i=0; i<n; i++)
{
k = p[i].btime/2;
```

```
for(j=0; j<k; j++)
printf(" ");
printf("P%d",p[i].pid); for(j=k+1;
j<p[i].btime; j++)
    printf(" ");
printf("|");
}
printf("\n-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
printf("-");
printf("\n0");
for(i=0; i<n; i++)
{
for(j=0; j<p[i].btime; j++)
printf(" ");
printf("%2d",p[i].ttime);
}

printf("\n\nAverage waiting time      : %5.2fms", awat);
printf("\nAverage turn around time : %5.2fms\n", atur);
}
```

Output

```
$ gcc pri.c

$ ./a.out
Enter no. of process : 5
Burst time for process P1 (in ms) : 10
Priority for process P1 : 3
Burst time for process P2 (in ms) : 7
Priority for process P2 : 1
Burst time for process P3 (in ms) : 6
Priority for process P3 : 3
Burst time for process P4 (in ms) : 13
Priority for process P4 : 4
Burst time for process P5 (in ms) : 5
Priority for process P5 : 2
    Priority Scheduling

-------------------------------------
- Process B-Time Priority T-Time  W-
Time
--------------------------------------
   P2      7       1       7          0
   P5      5       2       12         7
   P1      10      3       22     12
   P3      6       3       28     22
   P4      13      4       41     28
--------------------------------------

GANTT Chart
-----------------------------------------------------
|   P2   |  P5  |     P1     |  P3  |      P4       |
-----------------------------------------------------
0        7     12          22      28              41
```

```
Average waiting time     : 13.80ms
Average turn around time : 22.00ms
```

PRACTICAL – 9

AIM: Implementation of First-Fit Memory Allocation Algorithm

Algorithm

1. Declare structures *hole* and *process* to hold information about set of holes and processes respectively.
2. Get number of holes, say *nh*.
3. Get the size of each hole
4. Get number of processes, say *np*.
5. Get the memory requirements for each process.
6. Allocate processes to holes, by examining each hole as follows:

a. If hole size > process size then

i. Mark process as allocated to that hole. ii. Decrement hole size by process size.

b. Otherwise check the next from the set of hole

7. Print the list of process and their allocated holes or unallocated status.
8. Print the list of holes, their actual and current availability.
9. Stop

Result

Thus processes were allocated memory using first fit method.

Program

```
/* First fit allocation - ffit.c */

#include <stdio.h>

struct process
{
      int
size;
      int
flag;
      int holeid;
} p[10];

struct hole
{
      int size;
      int actual;
} h[10];

main()
{
int i, np, nh, j;

printf("Enter the number of Holes : ");
scanf("%d", &nh);
for(i=0; i<nh; i++)
{
    printf("Enter size for hole H%d : ",i);
    scanf("%d", &h[i].size);
    h[i].actual =  h[i].size;
}

printf("\nEnter number of process : " );
scanf("%d",&np);
for(i=0;i<np;i++)
{
    printf("enter the size of process P%d : ",i);
    scanf("%d", &p[i].size);
    p[i].flag = 0;
}

for (i=0; i<np; i++)
{
    for(j=0; j<nh; j++)
    {
        if(p[i].flag != 1)
        {
            if(p[i].size <= h[j].size)
            {
                p[i].flag = 1;
                p[i].holeid = j;
                h[j].size -= p[i].size;
            }
        }
    }
}

printf("\n\tFirst fit\n");
printf("\nProcess\tPSize\tHole");
for(i=0; i<np; i++)
{
    if(p[i].flag != 1)
        printf("\nP%d\t%d\tNot allocated", i, p[i].size);
    else
        printf("\nP%d\t%d\tH%d", i, p[i].size, p[i].holeid);
}

printf("\n\nHole\tActual\tAvailable");
for(i=0; i<nh ;i++)
    printf("\nH%d\t%d\t%d", i, h[i].actual, h[i].size);
printf("\n");
}
```

Outp
ut

```
$       gcc
ffit.c

$
./a.out
Enter the number of Holes : 5
Enter size for hole H0 : 100
Enter size for hole H1 : 500
Enter size for hole H2 : 200
Enter size for hole H3 : 300
Enter size for hole H4 : 600

Enter number of process : 4
enter the size of process P0 : 212
enter the size of process P1 : 417
enter the size of process P2 : 112
enter the size of process P3 : 426

First fit
Process PSize   Hole
P0      212     H1
P1      417     H4
P2      112     H1
P3       426     Not allocated

Hole    Actual  Available
H0      100     100
H1      500     176
H2      200     200
H3      300     300
H4      600     183
```

PRACTICAL – 10

AIM: Implementation of Best-Fit Memory Allocation Algorithm

Algorithm

1. Declare structures *hole* and *process* to hold information about set of holes and processes respectively.
2. Get number of holes, say *nh*.
3. Get the size of each hole
4. Get number of processes, say *np*.
5. Get the memory requirements for each process.
6. Allocate processes to holes, by examining each hole as follows:

a. Sort the holes according to their sizes in ascending order b. If hole

size > process size then

i. Mark process as allocated to that hole. ii. Decrement

hole size by process size.

c. Otherwise check the next from the set of sorted hole

7. Print the list of process and their allocated holes or unallocated status.
8. Print the list of holes, their actual and current availability.
9. Stop

Result

Thus processes were allocated memory using best fit method.

Program

```
/* Best fit allocation - bfit.c */

#include <stdio.h>

struct process
{
int size; int
flag; int
holeid;
} p[10];
struct hole
{
int hid; int
size; int
actual;
} h[10];

main()
{
int i, np, nh, j;
void bsort(struct hole[], int);

printf("Enter the number of Holes : ");
scanf("%d", &nh);
```

```
for(i=0; i<nh; i++)
{
printf("Enter size for hole H%d : ",i);
scanf("%d", &h[i].size);
h[i].actual =  h[i].size;
h[i].hid = i;
}

printf("\nEnter number of process : " );
scanf("%d",&np);
for(i=0;i<np;i++)
{
printf("enter the size of process P%d : ",i);
scanf("%d", &p[i].size);
p[i].flag = 0;
}

for(i=0; i<np; i++)
{
bsort(h, nh);

for(j=0; j<nh; j++)
{
if(p[i].flag != 1)
{
if(p[i].size <= h[j].size)
{
p[i].flag = 1; p[i].holeid =
h[j].hid; h[j].size -= p[i].size;
}
}
}
}

printf("\n\tBest fit\n");
printf("\nProcess\tPSize\tHole");
for(i=0; i<np; i++)
{
if(p[i].flag != 1)
printf("\nP%d\t%d\tNot allocated", i, p[i].size);
else
printf("\nP%d\t%d\tH%d", i, p[i].size, p[i].holeid);
}

printf("\n\nHole\tActual\tAvailable");
for(i=0; i<nh ;i++)
printf("\nH%d\t%d\t%d", h[i].hid, h[i].actual,
h[i].size);
printf("\n");
}

void bsort(struct hole bh[], int n)
{
struct hole temp;
int i,j;
for(i=0; i<n-1; i++)
{
```

```
for(j=i+1; j<n; j++)
{
if(bh[i].size > bh[j].size)
{
temp = bh[i]; bh[i] =
bh[j]; bh[j] = temp;
}
}
}
}
```

Outp
ut

```
$       gcc
bfit.c

$
./a.out
Enter the number of Holes : 5
Enter size for hole H0 : 100
Enter size for hole H1 : 500
Enter size for hole H2 : 200
Enter size for hole H3 : 300
Enter size for hole H4 : 600

Enter number of process : 4
enter the size of process P0 : 212
enter the size of process P1 : 417
enter the size of process P2 : 112
enter the size of process P3 : 426

Best fit

 Process PSize   Hole
 P0      212     H3
 P1      417     H1
 P2      112     H2
 P3      426     H4

 Hole    Actual  Available
 H1      500     83
 H3      300     88
 H2      200     88
 H0      100     100
 H4      600     174
```

PRACTICAL – 11

AIM: Implementation of FIFO Replacement Algorithm

Algorithm

1. Get length of the reference string, say *l*.
2. Get reference string and store it in an array, say *rs*.
3. Get number of frames, say *nf*.
4. Initalize *frame* array upto length *nf* to -1.
5. Initialize position of the oldest page, say *j* to 0.
6. Initialize no. of page faults, say *count* to 0.
7. For each page in reference string in the given order, examine:
   a. Check whether page exist in the *frame*
   array b. If it does not exist then
      i. Replace page in position *j*.
      ii. Compute page replacement position as (*j*+1)
      iii. Increment *count* by 1.
      iv. Display pages in *frame* array.
8. Print
9. Stop

Result

Thus page replacement was implemented using FIFO algorithm.

Program

```
/* FIFO page replacement - fifopr.c */

#include <stdio.h>

main()
{
   int i,j,l,rs[50],frame[10],nf,k,avail,count=0;

   printf("Enter length of ref. string : ");
   scanf("%d", &l);
   printf("Enter reference string :\n");
   for(i=1; i<=l; i++)
      scanf("%d", &rs[i]);
   printf("Enter number of frames : ");
   scanf("%d", &nf);

   for(i=0; i<nf; i++)
      frame[i] = -
          1;
   j = 0;

   printf("\nRef. str  Page frames");
   for(i=1; i<=l; i++)
   {
      printf("\n%4d\t", rs[i]);
      avail = 0;
      for(k=0; k<nf; k++)
         if(frame[k] == rs[i])
            avail = 1;
      if(avail == 0)
      {
         frame[j] =
         rs[i]; j =
         (j+1) % nf;
         count++;
         for(k=0; k<nf; k++)
            printf("%4d", frame[k]);
      }
   }
   printf("\n\nTotal no. of page faults : %d\n",count);
}
```

Output

```
 $ gcc fifopr.c

 $ ./a.out
 Enter length of ref. string :  20
 Enter reference string :
 1 2 3 4 2 1 5 6 2 1 2 3 7 6 3  2 1 2 3 6
 Enter number of frames : 5

 Ref. str  Page frames
    1       1  -1  -1  -1  -1
    2       1   2  -1  -1  -1
    3       1   2   3  -1  -1
    4       1   2   3   4  -1
    2
    1
    5       1   2   3   4   5
    6       6   2   3   4   5
    2
    1       6   1   3   4   5
    2       6   1   2   4   5
    3       6   1   2   3   5
    7       6   1   2   3   7
    6
    3
    2
    1
    2
    3
    6

Total  no.  of  page
faults : 10
```

PRACTICAL – 12

AIM: Implementation of LRU Page Replacement Algorithm by Stack method

Algorithm

1. Get length of the reference string, say *len*.
2. Get reference string and store it in an array, say *rs*.
3. Get number of frames, say *nf*.
4. Create *access* array to store counter that indicates a measure of recent usage.
5. Create a function *arrmin* that returns position of minimum of the given array.
6. Initalize *frame* array upto length *nf* to -1.
7. Initialize position of the page replacement, say *j* to 0.
8. Initialize *freq* to 0 to track page frequency
9. Initialize no. of page faults, say *count* to 0.
10. For each page in reference string in the given order, examine:

a. Check whether page exist in the *frame* array. b. If page exist in memory then

i. Store incremented *freq* for that page position in *access* array. c. If page does not exist in memory then

i. Check for any empty frames.

ii. If there is an empty frame,

¾ Assign that frame to the page

¾ Store incremented *freq* for that page position in *access* array.

¾ Increment *count*.

iii. If there is no free frame then

¾ Determine page to be replaced using *arrmin* function.

¾ Store incremented *freq* for that page position in *access* array.

¾ Increment *count*.

iv. Display pages in *frame* array.

11. Print *count*.
12. Stop

Result

Thus page replacement was implemented using LRU algorithm.

Program

```
/* LRU page replacement - lrupr.c */

#include <stdio.h>

int arrmin(int[], int);

main()
{
int i,j,len,rs[50],frame[10],nf,k,avail,count=0;
int access[10], freq=0, dm;

printf("Length of Reference string : ");
scanf("%d", &len);
printf("Enter reference string :\n");
for(i=1; i<=len; i++)
scanf("%d", &rs[i]); printf("Enter
no. of frames : "); scanf("%d",
&nf);

for(i=0; i<nf; i++)
   frame[i] = -1;
j = 0;

printf("\nRef. str  Page frames");
for(i=1; i<=len; i++)
{
printf("\n%4d\t", rs[i]);
avail = 0;
for(k=0; k<nf; k++)
{
if(frame[k] == rs[i])
{
avail = 1;
access[k] = ++freq;
break;
}
}
if(avail == 0)
{
dm = 0;
for(k=0; k<nf; k++)
{
if(frame[k] == -1)
dm = 1;
break;
}
if(dm == 1)
{
frame[k] = rs[i]; access[k] =
++freq; count++;
}
else
{
j = arrmin(access, nf); frame[j] =
rs[i]; access[j] = ++freq; count++;
}
for(k=0; k<nf; k++)
printf("%4d", frame[k]);
}
}
printf("\n\nTotal no. of page faults : %d\n", count);
}

int arrmin(int a[], int n)
{
      int i, min = a[0];
   for(i=1; i<n; i++) if
            (min > a[i])
min = a[i];
      for(i=0; i<n; i++)
     if (min == a[i])
return i;
}
```

Output

```
$ gcc lrupr.c

$ ./a.out
Length of Reference string : 20
Enter reference string :
 1 2 3 4 2 1 5 6 2 1 2  3 7  6 3 2 1 2 3 6
 Enter no. of frames :  5

Ref. str  Page frames
    1    1     -1   -1   -1   -1
    2    1     2    -1   -1   -1
    3    1     2    3    -1   -1
    4    1     2    3    4    -1
    2
    1
    5    1     2    3    4    5
    6    1     2    6    4    5
    2
    1
    2
    3    1    2    6   3   5
    7    1    2    6   3   7
    6
    3
    2
    1
    2
    3
    6

Total no. of page faults : 8
```

PRACTICAL – 13

AIM: Implementation of Optical Page Replacement Algorithm

```
#include<stdio.h>
#include<conio.h>
  int fr[3];
     void main()
      {
             void display();
              int p[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];
               int max,found=0,lg[3],index,k,l,flag1=0,flag2=0,pf=0,frsize=3;
                clrscr();
                 for(i=0;i<3;i++)
                  {
                          fr[i]=-1;
                  }
                          for(j=0;j<12;j++)
             {
               flag1=0;
                 flag2=0;
                         for(i=0;i<3;i++)
                 {
                          if(fr[i]==p[j])
                           {
                            flag1=1;
                              flag2=1;
                                         break;
                           }
                 }
                  if(flag1==0)
                         {
                          for(i=0;i<3;i++)
                               {
                                       if(fr[i]==-1)
                                                    {
                                                      fr[i]=p[j];
                                                              flag2=1;
                                                                 break;
                                                    }
                                      }
                  }

                          if(flag2==0)
                            {
                               for(i=0;i<3;i++)
                                       lg[i]=0;
                                          for(i=0;i<frsize;i++)
                                                        {
```

```
                                        for(k=j+1;k<12;k++)
                                           {
                                                if(fr[i]==p[k])
                                                   {
                                                        lg[i]=k-j;
                                                          break;
                                                   }
                                  }
                        }
                         found=0;
                          for(i=0;i<frsize;i++)
                               {
                                 if(lg[i]==0)
                                       {
                                        index=i;
                                         found=1;
                                               break;
                                        }
                                }
                                 if(found==0)
                                       {
                                        max=lg[0];
                                          index=0;
                                               for(i=1;i<frsize;i++)
                                                {
                                                 if(max<lg[i])
                                                      {
                                                      max=lg[i];
                                                       index=i;
                                         }
                           }
                %}
           fr[index]=p[j];
           pf++;
        }
         display();
}
        printf("\n no of page faults:%d",pf);
           getch();
}
void display()
{
int i;
printf("\n");
for(i=0;i<3;i++)
printf("\t%d",fr[i]);
}
```

Output:

OUTPUT :

```
2 -1 -1
2 3 -1
2 3 -1
2 3 1
2 3 5
2 3 5
4 3 5
4 3 5
4 3 5
2 3 5
2 3 5
2 3 5

no of page faults : 3
```