OPERATING SYSTEMS LAB FILE

ETCS – 352

Faculty: Dr. Sandeep Tayal

Name: Sanyam Jain

Enroll No: 00814802719

Group: 6C123

Branch: Computer Science & Engineering

Maharaja Agrasen Institute of Technology, PSP Area, Sector – 22, Rohini, New Delhi – 110085

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Outcome Based Learning

Course Outcomes

(Revision)

Course Objectives:

The goal of this course is to provide an introduction to the internal operation of modern operating systems. The course will cover processes and threads, mutual exclusion, CPU scheduling, deadlock, memory management and operational commands of Linux Operating System.

Course Outcomes:

ETCS 352.1 To Demonstrate CPU Scheduling Algorithms of Process Management.
ETCS 352.2 To Apply Memory Management Algorithms for Solving Page Replacement Policy
ETCS 352.3 To Demonstrate Solutions for The Deadlock Problems
ETCS 352.4 To Use the Linux Shell Scripting to Perform Various Operations.

Department of Computer Science and Engineering
Rubrics for Lab Assessment

Rubrics		0	1	2	3
		Missing	**Inadequate**	**Needs Improvement**	**Adequate**
R1	Is able to identify the problem to be solved and define the objectives of the experiment.	No mention is made of the problem to be solved.	An attempt is made to identify the problem to be solved but it is described in a confusing manner, objectives are not relevant, objectives contain technical/ conceptual errors or objectives are not measurable.	The problem to be solved is described but there are minor omissions or vague details. Objectives are conceptually correct and measurable but may be incomplete in scope or have linguistic errors.	The problem to be solved is clearly stated. Objectives are complete, specific, concise, and measurable. They are written using correct technical terminology and are free from linguistic errors.
R2	Is able to design a reliable experiment that solves the problem.	The experiment does not solve the problem.	The experiment attempts to solve the problem but due to the nature of the design the data will not lead to a reliable solution.	The experiment attempts to solve the problem but due to the nature of the design there is a moderate chance the data will not lead to a reliable solution.	The experiment solves the problem and has a high likelihood of producing data that will lead to a reliable solution.
R3	Is able to communicate the details of an experimental procedure clearly and completely.	Diagrams are missing and/or experimental procedure is missing or extremely vague.	Diagrams are present but unclear and/or Experimental procedure is present but important details are missing.	Diagrams and/or Experimental procedure Is present but with minor omissions or vague details.	Diagrams and/or experimental procedure is clear and complete.
R4	Is able to record and represent data in a meaningful way	Data are either absent or incomprehensible.	Some important data are absent or incomprehensible.	All important data are present, but recorded in a way that requires some effort to comprehend.	All important data are present, organized and recorded clearly.
R5	Is able to make a judgment about the results of the experiment.	No discussion is presented about the results of the experiment.	A judgment is made about the results, but it is not reasonable or coherent.	An acceptable judgment is made about the result, but the reasoning is flawed or incomplete.	An acceptable judgment is made about the result, with clear reasoning. The effects of assumptions and experimental uncertainties are considered.

INDEX

Exp No.	Experiment Name	Date of Performance	Date of Checking	R1 (3)	R2 (3)	R3 (3)	R4 (3)	R5 (3)	Total Marks (15)	Signature
1	Write a program to implement CPU scheduling for first come first serve.	11/03/2022								
2	Write a program to implement CPU scheduling for shortest job first.	25/03/2022								
3	Write a program to perform priority scheduling.	01/04/2022								
4	Write a program to implement CPU scheduling for Round Robin.	08/04/2022								
5	Write a program for page replacement policy using a) LRU b) FIFO c) Optimal.	22/04/2022								
6	Write a program to implement first fit, best fit and worst fit algorithm for memory management.	29/04/2022								
7	Write a program to implement reader/ writer problem using semaphore.	06/05/2022								
8	Write a program to implement Banker's algorithm for deadlock avoidance.	06/05/2022								

Exp No.	Experiment Name	Date of Performance	Date of Checking	R1 (3)	R2 (3)	R3 (3)	R4 (3)	R5 (3)	Total Marks (15)	Signature
1	Installation of Linux operating system.	18/02/2022								
2	Introduction of Basics Files, directories, system Commands of Linux.	18/02/2022								
3	Write a script to find the greatest of three numbers (Numbers passed as command line parameters)	25/02/2022								
	Write a script to check whether the given no. is even/odd	25/02/2022								
	Write a script to check whether the given number is prime or not	25/02/2022								
	Write a script to check whether the given input is a number or a string	25/02/2022								
	Write a script to compute no. of characters and words in each line of given file	25/02/2022								
4	Write a script to calculate the average of n numbers	04/03/2022								
	Write a script to print the Fibonacci series up to n terms	04/03/2022								
	Write a script to calculate the factorial of a given number	04/03/2022								
	Write a script to calculate the sum of digits of the given number	04/03/2022								
	Write a script to check whether the given string is a palindrome	04/03/2022								

Experiment–1

Aim: Installation of Linux Operating System.

Theory

Linux is an open source and free operating system to install which allows anyone with programming knowledge to modify and create its own operating system as per their requirements. Over many years, it has become more user-friendly and supports a lot of features such as

- Reliable when used with servers
- No need of antivirus
- A Linux server can run nonstop with the boot for many years.

It has many distributions such as Ubuntu, Fedora, Redhat, Debian but all run on top of Linux server itself. Installation of every distribution is similar, thus we are explaining Ubuntu here. So let's get started using this wonderful operating system by any of the following methods.

Installing Linux Using Virtual Box

What Are Requirements?

- At least 2GB RAM
- At least 20GB of free space

Step 1: Download the Fedora ISO.

Step 2: Create an empty virtual machine and configure it. Start Virtual Box and click on New.

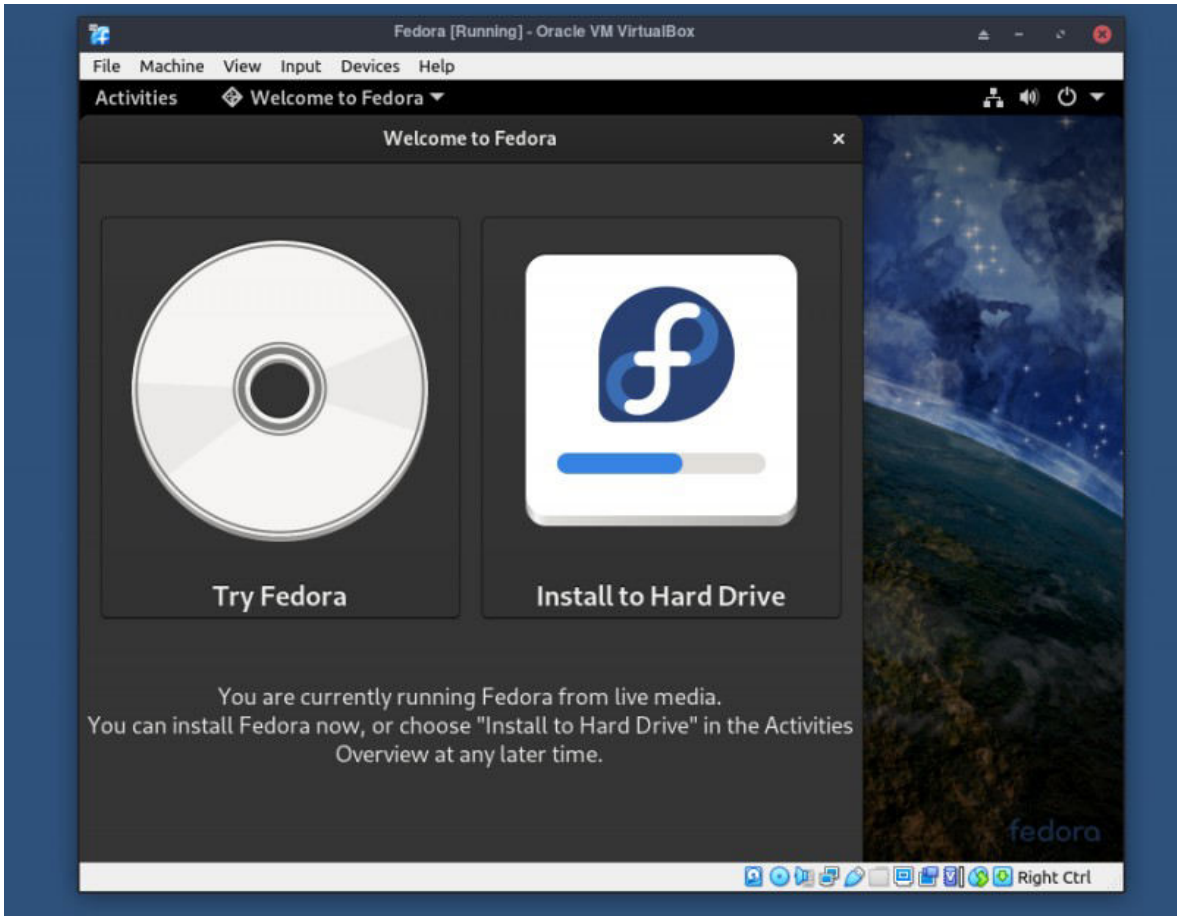
Step 3: Give name to virtual OS, type to be set to Linux and the version to Fedora (64-bit) and select the path. Click on next.

Step 4:Select RAM to allocate to Virtual machine, click on next. Choose Storage type (Dynamically allocated/Fixed size).

Step 5:Choose how much storage to allocate to fedora. Click on create.

Step 6: Click on Start and select fedora ISO file. The VM will boot from this ISO.

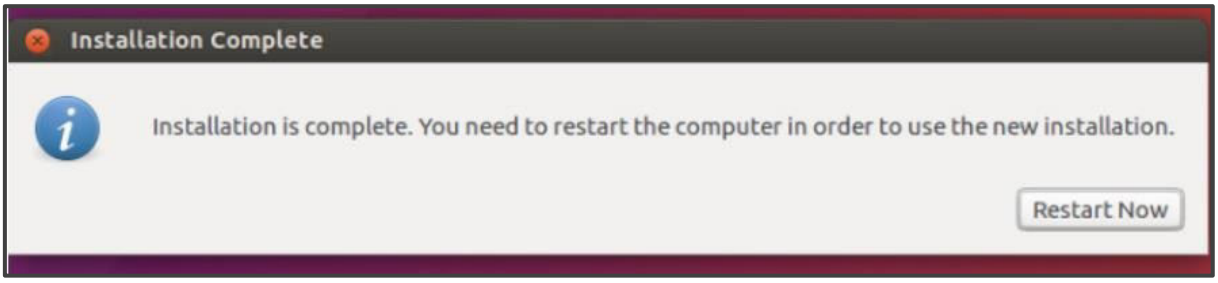
Step 7: Install Fedora in Virtual-Box. Click on Install to Hard Drive.



Step 8: Check the Keyboard Layout and Date & Time and then Select Installation Destination. Select the partition to install fedora and begin the installation.

Step 9: When installation will be finished Shut Down Fedora and remove the ISO File

Installation Finished, Start Virtual Machine, Fedora is Ready to use.



Experiment–2

Aim: Introduction to basic files, directories, system commands of Linux

Theory

1. ls –

In Linux, the *ls* command is used to list out files and directories. Some versions may support color-coding.

$ ls -l filename

2. cd /var/log –

Change the current directory. The forward slash is to be used in Linux.

$ cd /var/log

3. su / sudo command –

su command changes the shell to be used as a super user and until you use the exit command you can continue to be the super user

sudo – if you just need to run something as a super user, you can use the *sudo* command. Example – shutdown command the shutdown command safely turns off the computer system.

- *sudo shutdown 2* – shutdown and turns of the computer after 2 minutes
- *sudo shutdown –r 2* – shuts down and reboots in 2 minutes

$ sudo shutdown 2
$ sudo shutdown –r 2

4. pwd – Print Working Directory

One way to identify the directory you are working in is the *pwd* command

It displays the current working directory path and is useful when directory changes are often

$ pwd

5. mv – Move a file

To move a file or rename a file you would use the *mv* command. Here the file name gets changed from *first.txt* to *second.txt*

Type *ls* to view the change

$ mv first.txt second.txt

6. cp – Copy a file

cp source file destination file. In case you need a copy of the file *second.txt* in the same directory you have to use the *cp* command

$ cp second.txt third.txt

7. mkdir – to make a directory

mkdir [directory name] if you would like to create a directory in the name 'myproject' type

mkdir myproject

$ mkdir myproject

8. echo –

This command is used to display a text or a string to the standard output or a file.

$ echo –e "This is an article is for beginners. \nIt is on basic linux com mands

Will display the output as

This is an article is for beginners.

It is on basic linux commands

9. clear –

This command lets you clear the terminal screen.

$ clear

10. apt –get

apt -get is a powerful and free front-end package manager for Debian/Ubuntu systems. It is used to install new software packages, remove available software packages, upgrade existing software packages as well as upgrade the entire operating system. apt – stands for advanced packaging tool.

$ sudo apt-get update

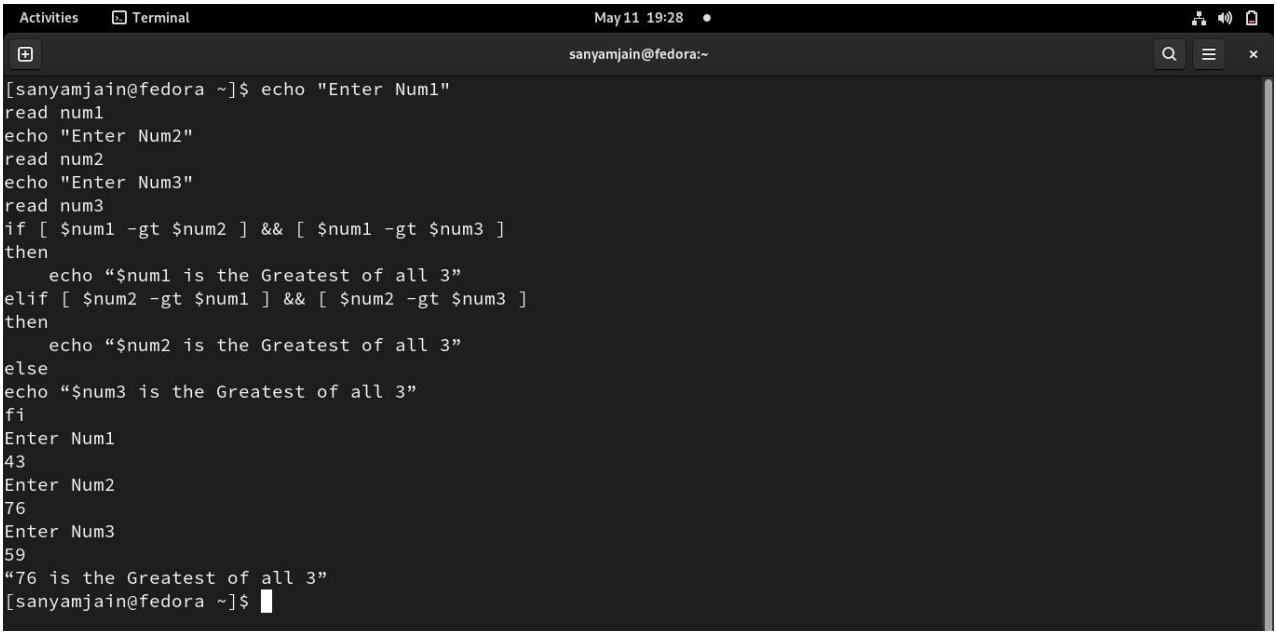
Experiment–3(i)

Aim: Write a script to find the greatest of three numbers.

Program

```
echo "Enter Num1"
read num1
echo "Enter Num2"
read num2
echo "Enter Num3"
read num3
if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]
then
   echo “$num1 is the Greatest of all 3”
elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]
then
   echo “$num2 is the Greatest of all 3”
else
echo “$num3 is the Greatest of all 3”
fi
```

Output



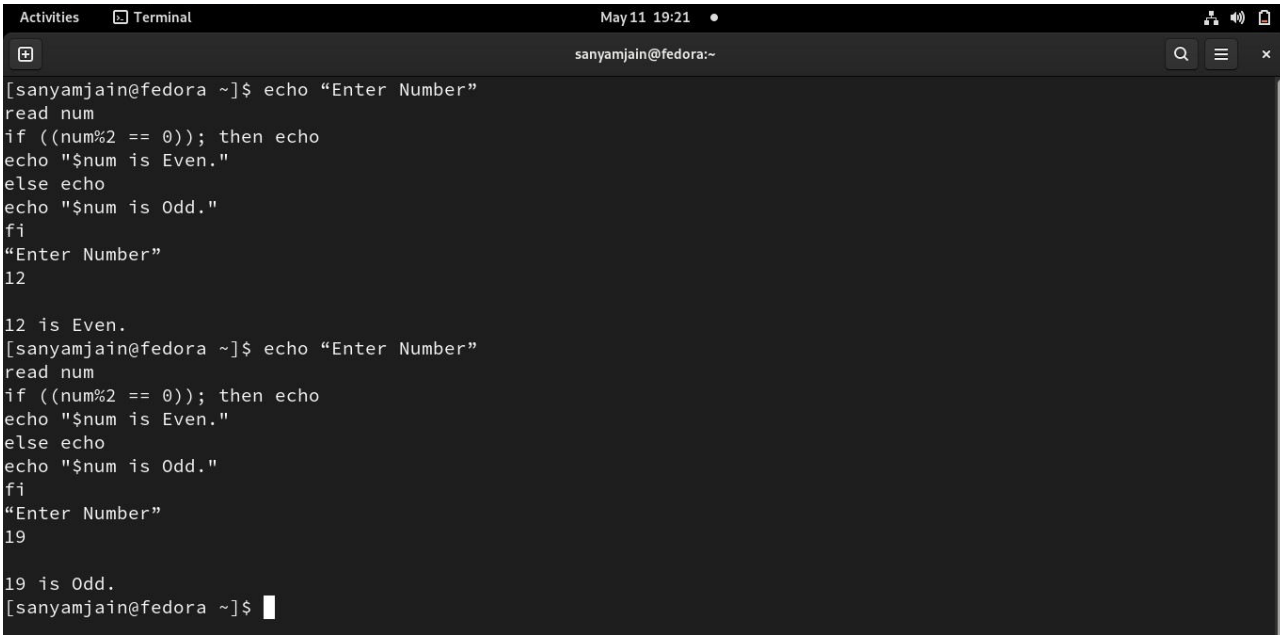
Experiment–3(ii)

Aim: Write a script to check whether the given number is even or odd.

Program

```
echo “Enter Number”
read num
if ((num%2 == 0)); then
    echo
    echo "$num is Even."
    else echo
    echo "$num is Odd."
fi
```

Output



Experiment–3(iii)

Aim: Write a script to check whether the given number is prime or not.

Program

```
echo -e "Enter Number : \c"
read n
for((i=2; i<=$n/2; i++))
do
  ans=$(( n%i ))
  if [ $ans -eq 0 ]
  then
    echo "$n is not a prime number."
    exit 0
  fi
done
echo "$n is a prime number."
```

Output

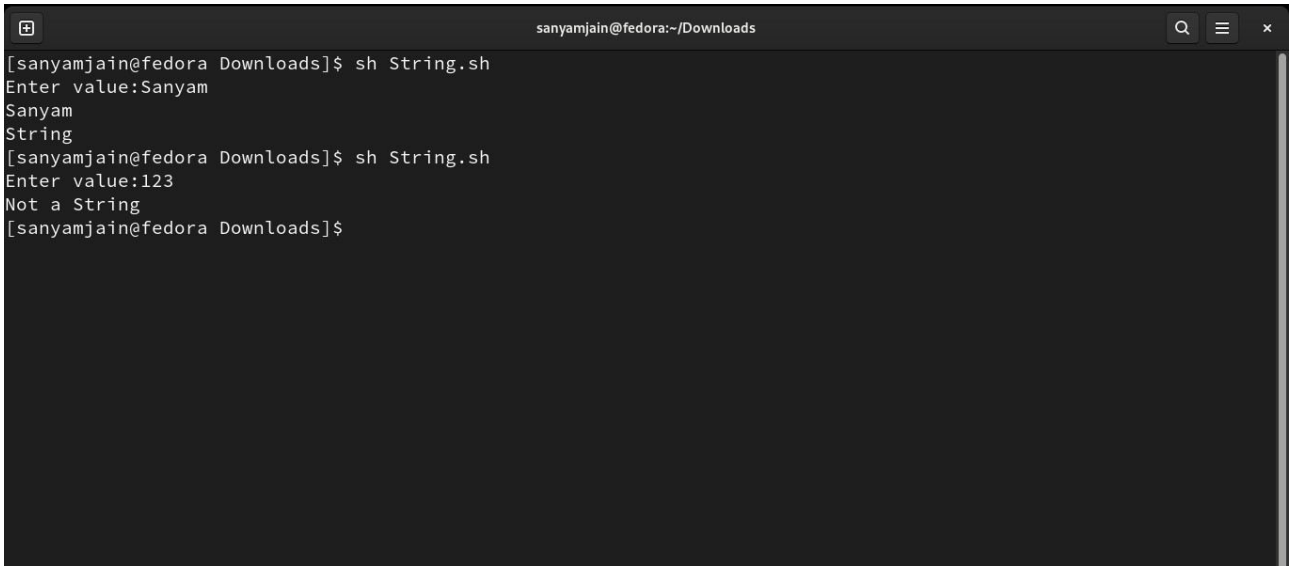
Experiment–3(iv)

Aim: Write a script to check whether the given input is a number or a string.

Program

```
read -p "Type a number or a string: " input if [[ $input =~ ^[+-]?[0-9]+$ ]]; then
    echo
    echo "'$input' is an integer."
elif [[ $input =~ ^[+-]?[0-9]+\.$ ]]; then echo
    echo "'$input' is a string."
elif [[ $input =~ ^[+-]?[0-9]+\.?[0-9]*$ ]]; then echo
    echo "'$input' is a float." else
    echo
    echo "'$input' is a string."
fi
```

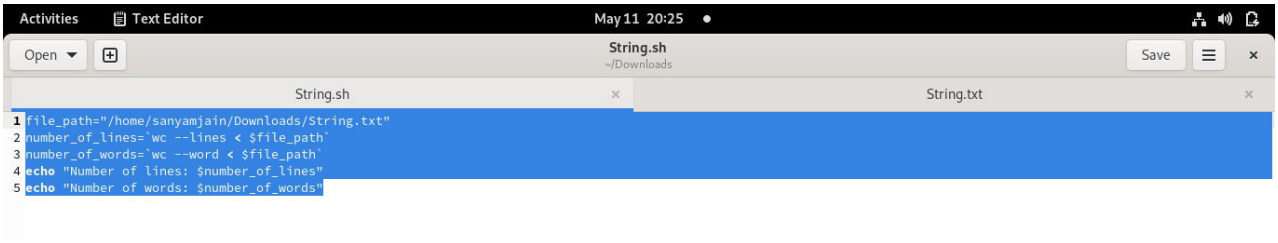
Output



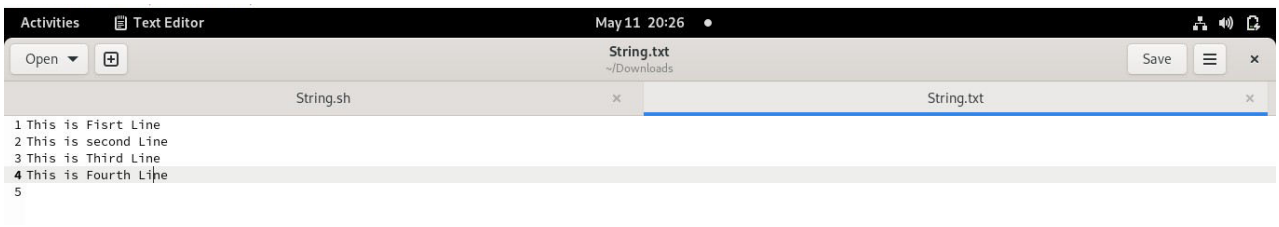
Experiment–3(v)

Aim: Write a script to compute number of characters and words in each line of a given file.

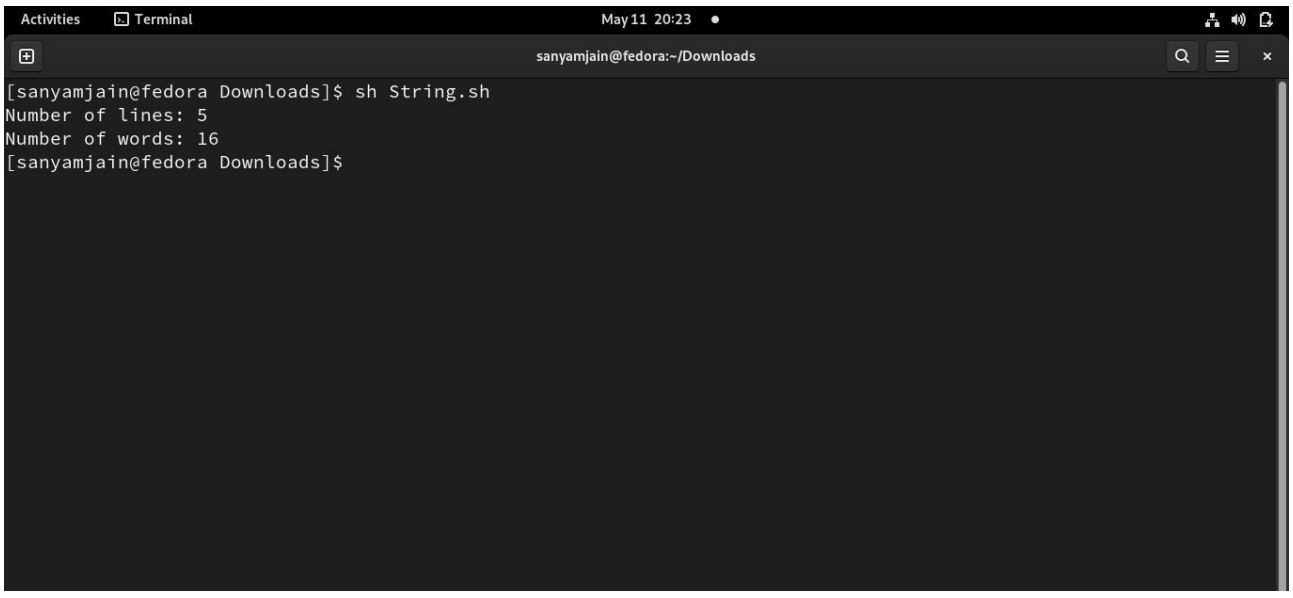
Program



Text File



Output



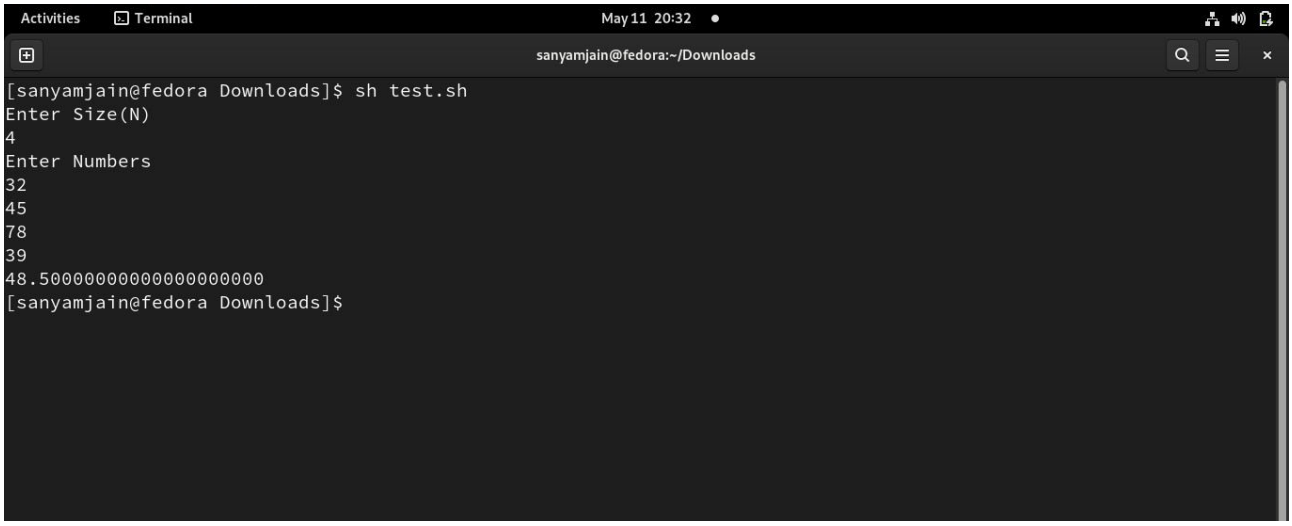
Experiment–4(i)

Aim: Write a script to calculate the average of n numbers.

Program

```
echo "Enter Size(N)"
read N
i=1
sum=0
echo "Enter Numbers"
while [ $i -le $N ]
do
read num
sum=$((sum + num))
i=$((i + 1))
done
avg=$(echo $sum / $N | bc
-l)
echo $avg
```

Output



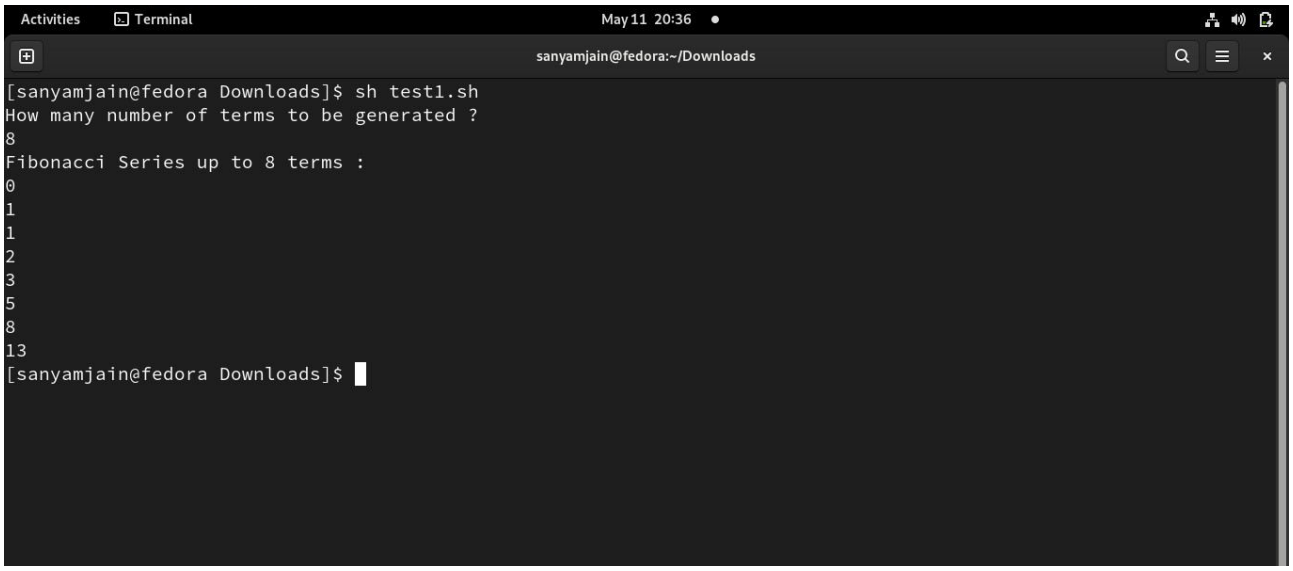
Experiment–4(ii)

Aim: Write a script to print the Fibonacci series upto n terms.

Program

```
echo "How many number of terms to be generated ?" read n
function fib{
   x=0 y=1 i=2 echo
   echo "Fibonacci Series up to $n terms :" echo "$x"
   echo "$y"
   while [ $i -lt $n ] do
        i=`expr $i + 1 `
        z=`expr $x + $y `
        echo "$z"
        x=$y
        y=$z
   done
}
r=`fib $n`
echo "$r"
```

Output



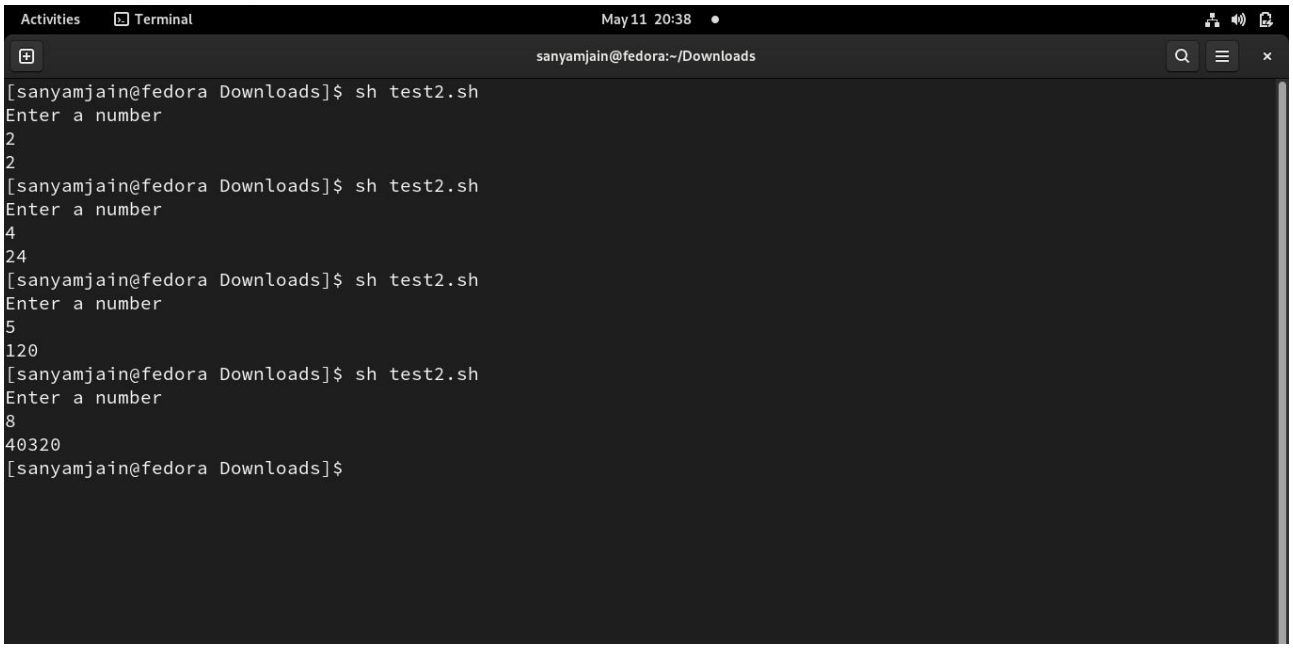
Experiment–4(iii)

Aim: Write a script to calculate the factorial of a given number.

Program

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

Output



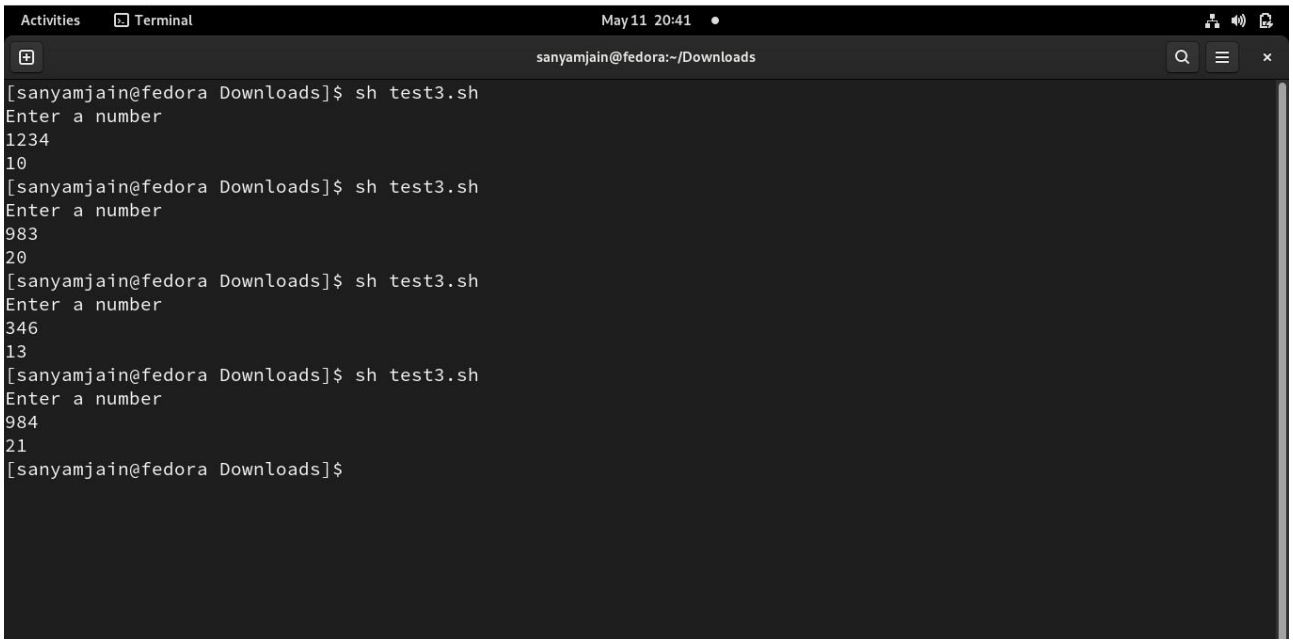
Experiment–4(iv)

Aim: Write a script to calculate the sum of digits of a given number.

Program

```
echo "Enter a number" read num
save=$num sum=0
while [ $num -gt 0 ] do
    mod=$((num % 10))
    sum=$((sum + mod))
    num=$((num / 10))       .
done echo
echo "Sum of digits($save): " $sum
```

Output



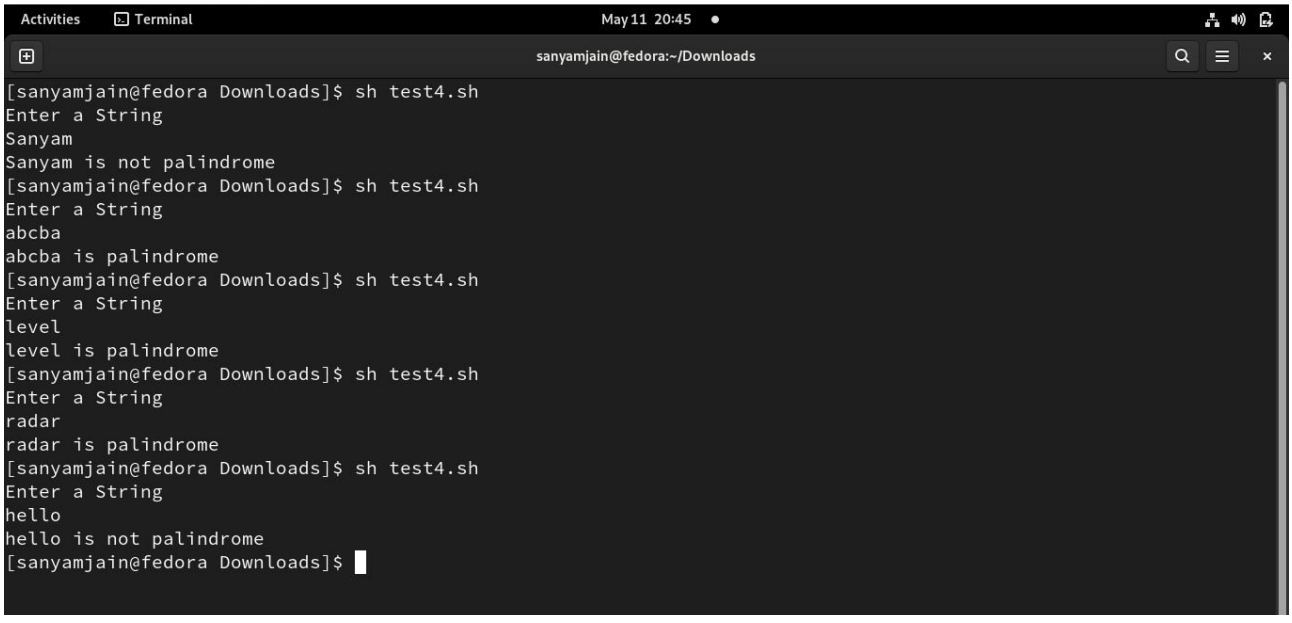
Experiment–4(v)

Aim: Write a script to check whether the string is a palindrome.

Program

```
echo "Enter a String" read   input reverse="" len=${#input}
for (( i=$len-1; i>=0; i-- )) do
        reverse="$reverse${input:$i:1}"

done

if [ $input == $reverse ] then
      echo

       echo "'$input' is a palindrome." else
      echo

      echo "$input is not palindrome."

fi
```

Output



ACCORDING TO SYLLABUS

EXPERIMENT-1

AIM: Write a program to implement CPU scheduling for first come first serve.

THEORY

First come first serve (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time. The job which comes first in the ready queue will get the CPU first. The lesser the arrival time of the job, the sooner will the job get the CPU. FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.

Advantages of FCFS

- Simple
- Easy
- First come, First serve

Disadvantages of FCFS

- The scheduling method is non-pre-emptive, the process will run to the completion.
- Due to the non-pre-emptive nature of the algorithm, the problem of starvation may occur.
- Although it is easy to implement, but it is poor in performance since the average waiting time is higher as compare to other scheduling algorithms.

CODE

```
processes=(1 2 3)
bt=(10 5 8)
declare -a wt
wt[0]=0
n=3
i=1
while [ $i -lt $n ] do
   wt[$i]=$(( wt[$i-1]+bt[$i-1] )) i=$((i + 1))
done
declare -a tat j=0
while [ $j -lt $n ] do
   tat[$j]=$(( bt[$j]+wt[$j] )) j=$((j+1))
done
total_wt=0
total_tat=0
echo "Processes Burst Time   Waiting Time   Turn Around Time"
```

```
k=0
while [ $k -lt $n ] do
   total_wt=$(( total_wt+wt[$k] )) total_tat=$(( total_tat+tat[$k] ))
   echo     " ${processes[$k]} "        " ${bt[$k]} "            " ${wt[$k]} "           " ${tat[$k]} k=$((k+1))"
done
total_wt=$(echo $total_wt / $n | bc -l) total_tat=$(echo $total_tat / $n | bc -l)
echo "Average Waiting Timw = " $total_wt echo "Average Turn Around Time = " $total_tat
```

VIVA QUESTIONS

Q1. What is CPU utilization?

Ans1. CPU utilization refers to a computer's usage of processing resources, or the amount of work handled by a CPU.

Q2. What are the different job scheduling techniques in operating systems?

Ans2. There are six popular process scheduling algorithms –

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Multiple-Level Queues Scheduling

Q3. Which scheduling algorithm allocates the CPU first to the process that requests the CPU first?

Ans3. First Come First Serve (FCFS) Scheduling

Q4. The FCFS algorithm is particularly troublesome for _____

Ans4. FCFS algorithm is particularly troublesome for **time-sharing systems**, where it is important that each user get a share of the CPU at regular intervals.

Q5. Which of the following statements are true?

i. **Shortest remaining time first scheduling may cause starvation**
ii. **Pre-emptive scheduling may cause starvation**
iii. **Round robin is better than FCFS in terms of response time**

Ans5. All three statements are true

```
Processes    Burst Time    Waiting Time    Turn Around Time
     1            10              0              10
     2            5              10              15
     3            8              15              23
Average Waiting Timw =  8.33333333333333333333
Average Turn Around Time =  16.00000000000000000000
```

EXPERIMENT-2

AIM: Write a program to implement CPU scheduling for shortest job first

THEORY

Shortest Job First (SJF) scheduling algorithm schedules the processes according to their burst time. In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next. However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.

Advantages of SJF

- Maximum throughput
- Minimum average waiting and turnaround time

Disadvantages of SJF

- May suffer with the problem of starvation
- It is not implementable because the exact Burst time for a process can't be known in advance.

CODE

```
processes=(1 2 3)
bt=(10 5 8)
 declare -a wt wt[0]=0
 n=3 a=0
 while [ $a -lt $n ] do
    pos=$a
    b=$((a+1))
    while [ $b -lt $n ] do
       if [ ${bt[$b]} -lt ${bt[$pos]} ] then
          pos=$b
       fi b=$((b+1))
    done temp=${bt[$a]} bt[$a]=${bt[$pos]} bt[$pos]=$temp
    temp=${processes[$a]} processes[$a]=${processes[$pos]} processes[$pos]=$temp
    a=$((a+1)) done
i=1
 while [ $i -lt $n ] do
    wt[$i]=$(( wt[$i-1]+bt[$i-1] )) i=$((i + 1))
done
 declare -a tat j=0
 while [ $j -lt $n ] do
    tat[$j]=$(( bt[$j]+wt[$j] )) j=$((j+1))
 done total_wt=0 total_tat=0
 echo "Processes Burst Time    Waiting Time   Turn Around Time" k=0
 while [ $k -lt $n ] do
    total_wt=$(( total_wt+wt[$k] )) total_tat=$(( total_tat+tat[$k] ))
    echo "    " ${processes[$k]} "         " ${bt[$k]} "             " ${wt[$k]} "            " ${tat[$k]} k=$((k+1))
```

```
done
 total_wt=$(echo $total_wt / $n | bc -l) total_tat=$(echo $total_tat / $n | bc -l)
 echo "Average Waiting Time = " $total_wt echo "Average Turn Around Time = " $total_tat
```

OUTPUT

```
Processes    Burst Time    Waiting Time    Turn Around Time
     2            5               0              5
     3            8               5              13
     1            10               13              23
Average Waiting Time =  6.000000000000000000
Average Turn Around Time =  13.666666666666666666
```

VIVA QUESTIONS

Q1. What is real difficulty with SJF in short term scheduling?

Ans1. The real difficulty with SJF in short term scheduling is knowing the length of the next CPU request.

Q2. Pre-emptive Shortest Job First scheduling is sometimes called _____

Ans2. Shortest Time Remaining First

Q3. Which scheduling algorithms give minimum average waiting time?

Ans3. Shortest Job First (SJF)

Q4. State true or false: Scheduling is allowing a job to use the processor?

Ans4. True

Q5. What do you mean by Average turnaround time and average waiting time?

Ans5. Turnaround time is the time interval from the time of submission of a process to the time of completion of that process. Average turnaround time can be defined as the average time taken by the processor to complete a process.

Waiting time is the time spent by a process waiting in the ready queue for getting the CPU. Average waiting time can be defined as the average time the CPU makes a process wait.

EXPERIMENT-3

AIM: Write a program to perform priority scheduling

THEORY

In **Priority scheduling**, there is a priority number assigned to each process. In some systems, the lower the number, the higher the priority. While, in the others, the higher the number, the higher will be the priority. The Process with the higher priority among the available processes is given the CPU. There are two types of priority scheduling algorithm exists. One is **Pre-emptive** priority scheduling while the other is **non-Pre-emptive** Priority scheduling.

CODE

```
processes=(1 2 3)
bt=(10 5 8)
priority=(2 1 3) declare -a wt wt[0]=0
n=3 a=0
while [ $a -lt $n ] do
   pos=$a b=$((a+1))
   while [ $b -lt $n ] do
      if [ ${priority[$b]} -lt ${priority[$pos]} ] then
         pos=$b
      fi b=$((b+1))
   done temp=${priority[$a]}
   priority[$a]=${priority[$pos]} priority[$pos]=$temp temp=${bt[$a]} bt[$a]=${bt[$pos]}
   bt[$pos]=$temp temp=${processes[$a]} processes[$a]=${processes[$pos]}
   processes[$pos]=$temp a=$((a+1))
done i=1
while [ $i -lt $n ] do
   wt[$i]=$(( wt[$i-1]+bt[$i-1] )) i=$((I + 1))
done
declare -a tat j=0
while [ $j -lt $n ] do
   tat[$j]=$(( bt[$j]+wt[$j] )) j=$((j+1))
done
total_wt=0 total_tat=0
echo "Processes Priority   Burst Time   Waiting Time   Turn Around Time" k=0
while [ $k -lt $n ] do
   total_wt=$(( total_wt+wt[$k] )) total_tat=$(( total_tat+tat[$k] ))
   echo "   " ${processes[$k]} " " ${priority[$k]} " " ${bt[$k]} " " ${wt[$k]} " " ${tat[$k]}
   k=$((k+1)) done
total_wt=$(echo $total_wt / $n | bc -l) total_tat=$(echo $total_tat / $n | bc -l)
echo "Average Waiting Time = " $total_wt echo "Average Turn Around Time = " $total_tat
```

OUTPUT

```
Processes    Priority    Burst Time    Waiting Time    Turn Around Time
     2           1             5              0               5
     1           2            10              5              15
     3           3             8             15              23
Average Waiting Time =  6.666666666666666666
Average Turn Around Time =  14.333333333333333333
```

VIVA QUESTIONS

Q1. In priority scheduling algorithm, when a process arrives at the ready queue, its priority is compared with the priority of which process?

Ans1. Currently running process

Q2. What is starvation?

Ans2. Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time.

Q3. What is aging?

Ans3. Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time. This helps to avoid starvation.

Q4. What are the factors for assigning priority to the process?

Ans4. Priority of processes depends some factors such as:

- Time limit
- Memory requirements of the process
- Ratio of average I/O to average CPU burst time

Q5. If two processes come on same time, on what criteria will process be chosen for processing?

Ans5. The process can be chosen on the basis of either Burst Time or Priority, whichever is required.

EXPERIMENT-4

AIM: Write a program to implement CPU scheduling for Round Robin

THEORY

Round Robin scheduling algorithm is one of the most popular scheduling algorithms which can actually be implemented in most of the operating systems. This is the **pre-emptive version** of first come first serve scheduling. The Algorithm focuses on Time Sharing. In this algorithm, every process gets executed in a **cyclic way**. A certain time slice is defined in the system which is called time **quantum**. Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will **terminate** else the process will go back to the **ready queue** and waits for the next turn to complete the execution.

CODE

```
processes=(1 2 3)
bt=(10 5 8)
quantum=2 declare -a rem_bt declare -a wt wt[0]=0
n=3 a=0
while [ $a -lt $n ] do
   rem_bt[$a]=${bt[$a]}
   a=$((a+1))
done t=0
while true do
   d=1
   i=0
   while [ $i -lt $n ] do
      if [ ${rem_bt[$i]} -gt 0 ] then
         d=0
         if [ ${rem_bt[$i]} -gt $quantum ] then
            t=$((t+quantum)) rem_bt[$i]=$((rem_bt[$i]-quantum))
         else
            t=$((t+rem_bt[$i])) wt[$i]=$((t-bt[$i])) rem_bt[$i]=0
         fi
      fi
      i=$((i+1)) done
      if [ $d -eq 1 ] then
      break
   fi done
declare -a tat j=0
while [ $j -lt $n ] do
   tat[$j]=$(( bt[$j]+wt[$j] )) j=$((j+1))
done
total_wt=0
total_tat=0
echo “Processes Burst Time   Waiting Time   Turn Around Time” k=0
while [ $k -lt $n ] do
   total_wt=$(( total_wt+wt[$k] )) total_tat=$(( total_tat+tat[$k] ))
```

```
    echo "   " ${processes[$k]} "        " ${bt[$k]} "            " ${wt[$k]} "           " ${tat[$k]} k=$((k+1))
done
total_wt=$(echo $total_wt / $n | bc -l) total_tat=$(echo $total_tat / $n | bc -l) echo "Time Quantum
= " $quantum
echo "Average Waiting Time = " $total_wt echo "Average Turn Around Time = " $total_tat
```

OUTPUT

```
Processes    Burst Time    Waiting Time    Turn Around Time
     1            10              13              23
     2            5               10              15
     3            8               13              21
Time Quantum =  2
Average Waiting Time =  12.000000000000000000
Average Turn Around Time =  19.666666666666666666
```

VIVA QUESTIONS

Q1. **What do you mean by Time Slice?**

Ans1. A short interval of time allotted to each user or program in a multitasking or timesharing system.

Q2. **Time quantum is defined in which scheduling algorithm?**

Ans2. Round Robin Scheduling algorithm

Q3. **What is the limitation of Round Robin Scheduling Algorithm?**

Ans3. Round Robin Scheduling Algorithm has various limitations:

- Setting the quantum too short, increases the overhead and lowers the CPU efficiency, but setting it too long may cause poor response to short processes.
- Average waiting time under the RR policy is often long.

Q4. **What are the factors that affect Round Robin Scheduling Algorithm?**

Ans4. Round Robin Scheduling Algorithm is affected only by the length of the quantum.

Q5. **Round robin scheduling falls under the category of which scheduling non-pre-emptive scheduling or Pre-emptive scheduling.**

Ans5. Pre-emptive scheduling

EXPERIMENT-5

AIM: Write a program for page replacement policy using

a) LRU b) FIFO c) Optimal.

THEORY

The page replacement algorithm decides which memory page is to be replaced. The process of replacement is sometimes called swap out or write to disk. Page replacement is done when the requested page is not found in the main memory (page fault). There are various page replacement algorithms. Each algorithm has a different method by which the pages can be replaced.

1. ***Optimal Page Replacement algorithm*** → this algorithm replaces the page which will not be referred for so long in future. Although it cannot be practically implementable but it can be used as a benchmark. Other algorithms are compared to this in terms of optimality.
2. ***Least recent used (LRU) page replacement algorithm*** → this algorithm replaces the page which has not been referred for a long time. This algorithm is just opposite to the optimal page replacement algorithm. In this, we look at the past instead of staring at future.
3. ***FIFO*** → in this algorithm, a queue is maintained. The page which is assigned the frame first will be replaced first. In other words, the page which resides at the rare end of the queue will be replaced on every page fault.

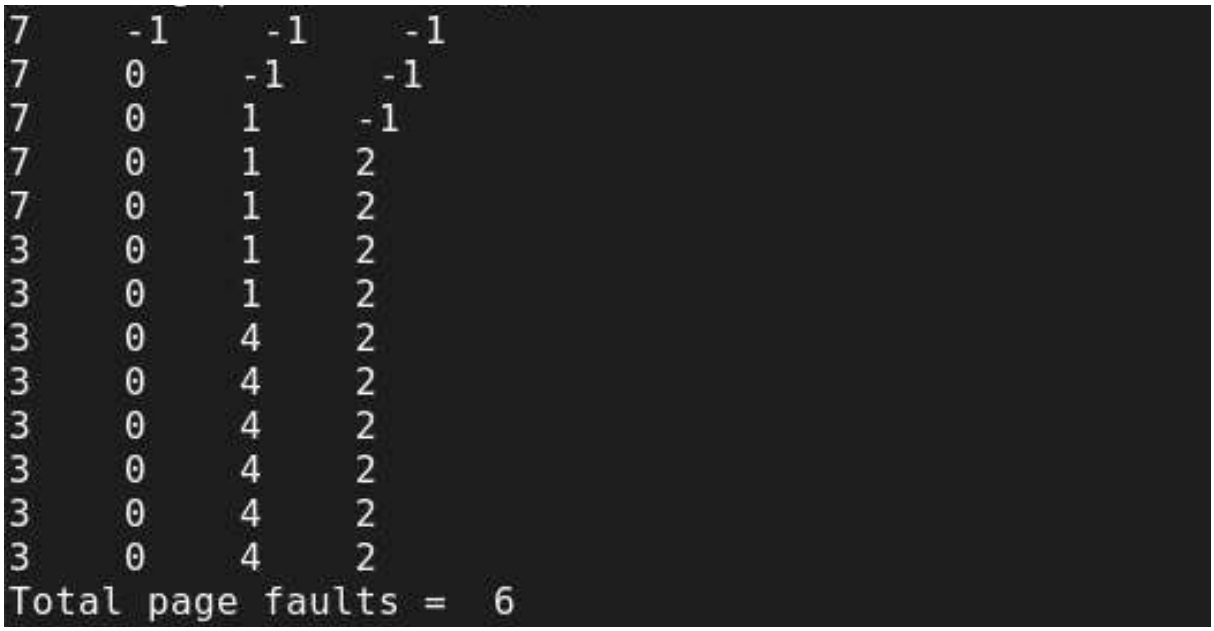
CODE AND OUTPUT

- **LRU**

```
pages=(7 0 1 2 0 3 0 4 2 3 0 3 2)
capacity=4
count=0
fault=0
n=13
declare -a frames declare -a time i=0
while [ $i -lt $capacity ] do
   frames[$i]=-1 i=$((i+1))
done j=0
while [ $j -lt $n ] do
   flag1=0 flag2=0 k=0
   while [ $k -lt $capacity ] do
   if [ ${frames[$k]} -eq ${pages[$j]} ] then
         count=$
         ((count+1))
         time[$j]=$count
         flag1=1
         flag2=1
```

```
        break
      fi
      k=$((k+1))
   done
   if [ $flag1 -eq 0 ] then
      a=0
      while [ $a -lt $capacity ] do
         if [ ${frames[$a]} -eq -1 ] then
            count=$((count+1)) fault=$((fault+1)) frames[$a]=${pages[$j]} time[$a]=$count
            flag2=1 break
         fi
         a=$((a+1)) done
   fi
   if [ $flag2 -eq 0 ] then
      pos=0 minimum=${time[0]} b=1
      while [ $b -lt $capacity ] do
         if [ ${time[$b]} -lt $minimum ] then
            minimum=${time[$b]} pos=$b
         fi b=$((b+1))
      done count=$((count+1)) fault=$((fault+1))
      frames[$pos]=${pages[$j]} time[$pos]=$count
   fi
   echo ${frames[0]} "  " ${frames[1]} "  " ${frames[2]} "  " ${frames[3]} j=$((j+1))
done
echo "Total page faults = " $fault
```

OUTPUT



a) FIFO

```
pages=(7 0 1 2 0 3 0 4 2 3 0 3 2)
capacity=4 count=0 fault=0 n=13
declare -a frames i=0
while [ $i -lt $capacity ] do
   frames[$i]=-1 i=$((i+1))
done j=0
while [ $j -lt $n ] do
   flag=0
   k=0
   while [ $k -lt $capacity ] do
      if [ ${frames[$k]} -eq ${pages[$j]} ] then
         flag=1 fault=$((fault-1))
      fi
      k=$((k+1)) done fault=$((fault+1))
      if [ $fault -le $capacity ] && [ $flag -eq 0 ] then
      frames[$j]=${pages[$j]}
      else if [ $flag -eq 0 ] then
         frames[$(( (fault-1)%capacity))]=${pages[$j]}
      fi
   fi
   echo ${frames[0]} "  " ${frames[1]} "  " ${frames[2]} "  " ${frames[3]} j=$((j+1))
done
echo "Total page faults = " $fault
```

OUTPUT

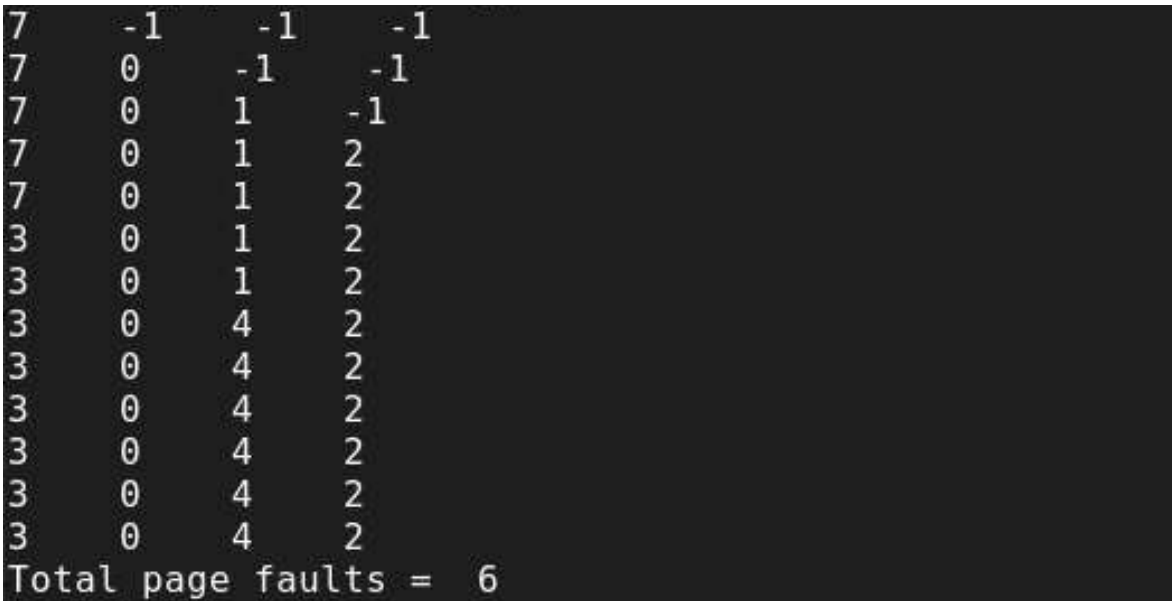
```
7    -1    -1    -1
7    0     -1    -1
7    0     1     -1
7    0     1     2
7    0     1     2
3    0     1     2
3    0     1     2
3    4     1     2
3    4     1     2
3    4     1     2
3    4     0     2
3    4     0     2
3    4     0     2
Total page faults =  7
```

- ***Optimal***

```
pages=(7 0 1 2 0 3 0 4 2 3 0 3 2)
capacity=4 fault=0 n=13
declare -a frames declare -a temp i=0
while [ $i -lt $capacity ] do
  frames[$i]=-1 i=$((i+1))
done j=0while [ $j -lt $n ] do
  flag1=0 flag2=0 k=0
  while [ $k -lt $capacity ] do
    if [ ${frames[$k]} -eq ${pages[$j]} ] then
      flag1=1 flag2=1 break
    fi k=$((k+1))
  done
  if [ $flag1 -eq 0 ] then
    a=0
    while [ $a -lt $capacity ] do
      if [ ${frames[$a]} -eq -1 ] then
        fault=$((fault+1)) frames[$a]=${pages[$j]} flag2=1
        break
      fi a=$((a+1))
    done
  fi
  if [ $flag2 -eq 0 ] then
    flag3=0 x=0
    while [ $x -lt $capacity ] do
      temp[$x]=-1 y=$((j+1))
      while [ $y -lt $n ] do
        if [ ${frames[$x]} -eq ${pages[$y]} ] then
          temp[$x]=$y break
        fi y=$((y+1))
      done x=$((x+1))
    done z=0
    while [ $z -lt $capacity ] do
      if [ ${temp[z]} -eq -1 ] then
        pos=$z flag3=1 break
      fi z=$((z+1))
    done
    if [ $flag3 -eq 0 ] then
      maximum=${temp[0]} pos=0
       w=0
      while [ $w -lt $capacity ] do
        if [ ${temp[$w]} -gt $maximum ] then
          maximum=${temp[$w]} pos=$w
        fi
        w=$((w+1)) done
    fi
    frames[$pos]=${pages[$j]} fault=$((fault+1))
  fi
```

```
   echo ${frames[0]} "  " ${frames[1]} "  " ${frames[2]} "  " ${frames[3]} j=$((j+1))
done
echo "Total page faults = " $fault
```

OUTPUT



VIVA QUESTIONS

Q1. **Define Demand Paging, Page fault interrupt, and Thrashing?**

Ans1. Demand Paging suggests keeping all pages of the frames in the secondary memory until they are required. In other words, it says that do not load any page in the main memory until it is required. Page Fault Interrupt is an interrupt signal used to detect page faults.

Thrashing occurs when a computer's virtual memory resources are overused, leading to a constant state of paging and page faults, inhibiting most application-level processing. It causes the performance of the computer to degrade or collapse. The situation can continue indefinitely until the user closes some running applications or the active processes free up additional virtual memory resources.

Q2. **Under what circumstances do page faults occur? Describe the actions taken by the operating system when a page fault occurs?**

Ans2. A page fault occurs when an access to a page that has not been brought into main memory takes place. The operating system verifies the memory access, aborting the program if it is invalid. If it is valid, a free frame is located and I/O is requested to read the needed page into the free frame. Upon completion of I/O, the process table and page table are updated and the instruction is restarted.

Q3. **What is the cause of thrashing? How does the system detect thrashing? Once it detects thrashing, what can the system do to eliminate this problem?**

Ans3. Thrashing is caused by under allocation of the minimum number of pages required by a process, forcing it to continuously page fault. The system can detect thrashing by evaluating the level of CPU utilization as compared to the level of multiprogramming. It can be eliminated by reducing the level of multiprogramming.

Q4. **What do you mean by Belady's anomaly?**

Ans4. Bélády's anomaly is the name given to the phenomenon where increasing the number of page frames results in an increase in the number of page faults for a given memory access pattern.

Q5. **Compare LRU, FIFO and Optimal page replacement algorithm.**

Ans5. In FIFO algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal. In Optimal page replacement algorithm, pages are replaced which would not be used for the longest duration of time in the future. In LRU algorithm, page will be replaced which is least recently used.

EXPERIMENT-6

AIM: Write a program to implement first fit, best fit and worst fit algorithm for memory management.

THEORY

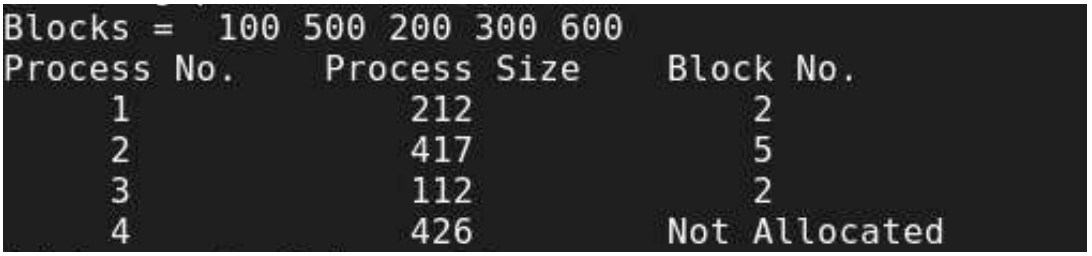
Memory is the important part of the computer that is used to store the data. Its management is critical to the computer system because the amount of main memory available in a computer system is very limited. At any time, many processes are competing for it. Moreover, to increase performance, several processes are executed simultaneously. For this, we must keep several processes in the main memory, so it is even more important to manage them effectively.

CODE

a) First fit

```
block=(100 500 200 300 600)
 echo “Blocks = “ ${block[@]} process=(212 417 112 426) m=5
n=4
 declare -a allocation i=0
 while [ $i -lt $n ] do
   allocation[$i]=-1 i=$((i+1))
       done j=0
 while [ $j -lt $n ] do
   k=0
   while [ $k -lt $m ] do
      if [ ${block[$k]} -ge ${process[$j]} ] then
        allocation[$j]=$k block[$k]=$((block[$k]-process[$j])) break
      fi k=$((k+1))
   done j=$((j+1))
done
 echo “Process No. Process Size   Block No.”
 a=0
 while [ $a -lt $n ] do
   if [ ${allocation[$a]} -ne -1 ] then
      echo “   “ $((a+1)) “          “ ${process[$a]} “          “ $((allocation[$a]+1)) else
      echo “   “ $((a+1)) “          “ ${process[$a]} “       Not Allocated”
   fi a=$((a+1))
done
```

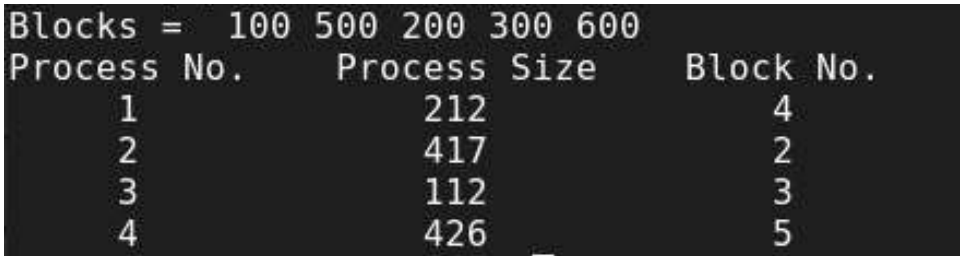
OUTPUT



b) Best fit

```
block=(100 500 200 300 600)
 echo “Blocks = “ ${block[@]} process=(212 417 112 426) m=5
n=4
 declare -a allocation i=0
 while [ $i -lt $n ] do
   allocation[$i]=-1 i=$((i+1))
 done j=0
 while [ $j -lt $n ] do
   bestindex=-1 k=0
   while [ $k -lt $m ] do
      if [ ${block[$k]} -ge ${process[$j]} ] then
      if [ $bestindex -eq -1 ] then
            bestindex=$k else
            if [ ${block[$bestindex]} -gt ${block[$k]} ] then
               bestindex=$k
            fi
         fi
      fi
      k=$((k+1)) done
   if [ $bestindex -ne -1 ] then
      allocation[$j]=$bestindex block[$bestindex]=$((block[$bestindex]-process[$j]))
   fi j=$((j+1))
done
 echo “Process No. Process Size   Block No.” a=0
 while [ $a -lt $n ] do
   if [ ${allocation[$a]} -ne -1 ] then
      echo “   “ $((a+1)) “          “ ${process[$a]} “          “ $((allocation[$a]+1)) else
      echo “   “ $((a+1)) “          “ ${process[$a]} “       Not Allocated”
   fi a=$((a+1))
done
```

OUTPUT



c) Worst fit

```
block=(100 500 200 300 600)
 echo “Blocks = “ ${block[@]} process=(212 417 112 426) m=5
n=4
 declare -a allocation i=0
 while [ $i -lt $n ] do
   allocation[$i]=-1 i=$((i+1))
 done j=0
 while [ $j -lt $n ] do
   worstindex=-1 k=0
   while [ $k -lt $m ] do
      if [ ${block[$k]} -ge ${process[$j]} ] then
        if [ $worstindex -eq -1 ] then
           worstindex=$k else
           if [ ${block[$worstindex]} -lt ${block[$k]} ] then
              worstindex=$k
           fi
         fi
      fi
      k=$((k+1)) done
   if [ $worstindex -ne -1 ] then
      allocation[$j]=$worstindex block[$worstindex]=$((block[$worstindex]-process[$j]))
   fi j=$((j+1))
done
 echo “Process No. Process Size   Block No.” a=0
 while [ $a -lt $n ] do
   if [ ${allocation[$a]} -ne -1 ] then
      echo “   “ $((a+1)) “          “ ${process[$a]} “          “ $((allocation[$a]+1))
      else
      echo “   “ $((a+1)) “          “ ${process[$a]} “       Not Allocated”
   fi a=$((a+1))
done
```

OUTPUT

```
Blocks =  100 500 200 300 600
Process No.    Process Size    Block No.
     1              212            5
     2              417            2
     3              112            5
     4              426        Not Allocated
```

VIVA-QUESTIONS

Q1. **Difference between Logical and Physical Address Space?**

Ans1. Logical Address Space is the set of all logical addresses generated by CPU for a program. Physical Address Space is the set of all physical address mapped to corresponding logical addresses.

Q2. **Describe first-fit, best-fit strategies and worst fit strategies for disk space allocation, with their merits and demerits.**

Ans2. First Fit: In the first fit approach is to allocate the first free partition or hole large enough which can accommodate the process. It finishes after finding the first suitable free partition.

Advantages of First-Fit Memory Allocation:
It is fast in processing.
Disadvantages of First-Fit Memory Allocation :
It wastes a lot of memory.

Best Fit: The best fit deals with allocating the smallest free partition which meets the requirement of the requesting process. This algorithm first searches the entire list of free partitions and considers the smallest hole that is adequate. It then tries to find a hole which is close to actual process size needed.

Advantages of Best-Fit Allocation :
Memory Efficient.
Disadvantages of Best-Fit Allocation :
It is a Slow Process.

Worst Fit: In worst fit approach is to locate largest available free portion so that the portion left will be big enough to be useful. It is the reverse of best fit.

Advantages of Worst-Fit Allocation :
Since this process chooses the largest hole/partition, therefore there will be large internal fragmentation. Now, this internal fragmentation will be quite big so that other small processes can also be placed in that leftover partition.

Disadvantages of Worst-Fit Allocation :
It is a slow process because it traverses all the partitions in the memory and then selects the largest partition among all the partitions, which is a time-consuming process.

Q3. **What is the impact of fixed partitioning on fragmentation?**

Ans3. In fixed partitioning, the partitions will be wasted and remain unused if the process size is lesser than the total size of the partition. In this way, memory gets wasted, and this is called Internal fragmentation.

Q4. **In fixed sized partition, the degree of multi programming is bounded by ____**

Ans4. The number of partitions

Q5. **The first fit, best fit and worst fit are strategies to select a _____**

Ans5. Free hole from a set of available holes.

EXPERIMENT-7

AIM: Write a program to implement reader/writer problem using semaphore.

THEORY

The readers-writers problem is a classical problem of process synchronization; it relates to a data set such as a file that is shared between more than one process at a time. Among these various processes, some are Readers - which can only read the data set they do not perform any updates, some are Writers - can both read and write in the data sets.

The readers-writers problem is used for managing synchronization among various reader and writer process so that there are no problems with the data sets, i.e., no inconsistency is generated.

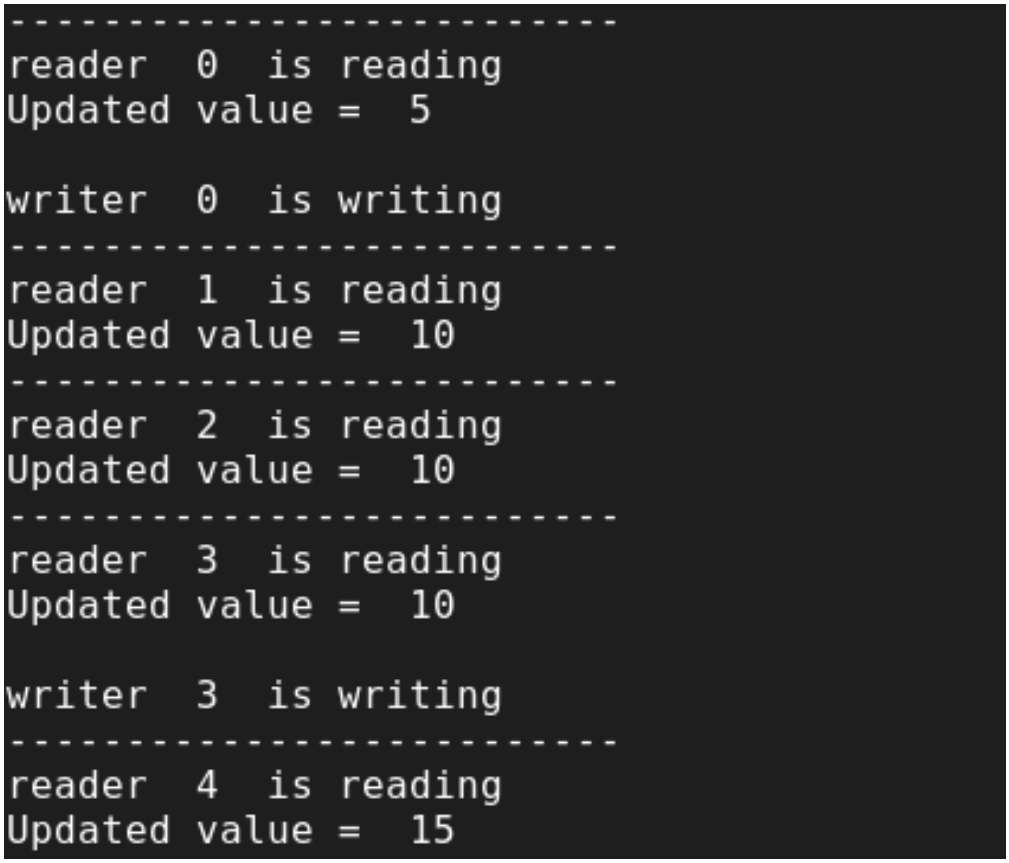
Semaphore is simply an integer variable that is shared between threads. This variable is used to solve the critical section problem and to achieve process synchronization in the multiprocessing environment.

CODE

```
function sem_wait(){
   while [ $1 -le 0 ] do
      $1=$(($1-1))
   done
}
function signal()
{
   result=$(($1+1))
}
 x=1 y=1 z=1
 wsem=1 rsem=1 readcount=0 writecount=0 var=5
function reader()
{
   echo "....." echo "reader " $1 " is reading" sem_wait $z
   sem_wait $rsem sem_wait $x
   readcount=$((readcount+1))
   if [ $readcount -eq 1 ]
   then
      sem_wait $wsem
   fi
   signal $x signal $rsem signal $z
   echo "Updated value = " $var sem_wait $x readcount=$((readcount-1)) if [ $readcount -eq 0 ]
   then
      signal $wsem
   fi
   signal $x
}
```

```
function writer(){
   echo
   echo "writer " $1 " is writing" sem_wait $y writecount=$((writecount+1)) if [ $writecount -eq 1 ]
   then
      sem_wait $rsem
   fi
   signal $y sem_wait $wsem var=$((var+5)) signal $wsem sem_wait $y
   writecount=$((writecount-1)) if [ $writecount -eq 0 ]
   then
      signal $rsem
   fi
   signal $y
}
reader 0
writer 0
reader 1
reader 2
reader 3
writer 3
reader 4
```

OUTPUT



VIVA-QUESTIONS

Q1. **Define the critical section problem and explain the necessary characteristics of a correct solution.**

Ans1. The critical section problem is used to design a protocol followed by a group of processes, so that when one process has entered its critical section, no other process is allowed to execute in its critical section. The critical section problem needs a solution to synchronize the different processes. The solution to the critical section problem must satisfy the following conditions −

- ***Mutual Exclusion***

Mutual exclusion implies that only one process can be inside the critical section at any time. If any other processes require the critical section, they must wait until it is free.

- ***Progress***

Progress means that if a process is not using the critical section, then it should not stop any other process from accessing it. In other words, any process can enter a critical section if it is free.

- ***Bounded Waiting***

Bounded waiting means that each process must have a limited waiting time. It should not wait endlessly to access the critical section.

Q2. **What do understand by Race Condition?**

Ans2. A race condition is an undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time, but because of the nature of the device or system, the operations must be done in the proper sequence to be done correctly.

Q3. **Define semaphore and its limitations.**

Ans3. Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization. There are two main types of semaphores i.e., counting semaphores and binary semaphores. Some of the disadvantages of semaphores are as follows −

- Semaphores are complicated so the wait and signal operations must be implemented in the correct order to prevent deadlocks.
- Semaphores are impractical for last scale use as their use leads to loss of modularity. This happens because the wait and signal operations prevent the creation of a structured layout for the system.
- Semaphores may lead to a priority inversion where low priority processes may access the critical section first and high priority processes later.

Q4. **Define monitor.**

Ans4. The Monitor is a module or package which encapsulates shared data structure, procedures, and the synchronization between the concurrent procedure invocations.

Q5. **What are classical problems of process synchronization?**

Ans5. The classical problems of synchronization are as follows:

- ***Bound-Buffer problem:*** In this problem, there is a buffer of n slots, and each buffer is capable of storing one unit of data. There are two processes that are operating on the buffer – Producer and Consumer. The producer tries to insert data and the consumer tries to remove data. If the processes are run simultaneously they will not yield the expected output.
- ***Sleeping barber problem:*** This problem is based on a hypothetical barbershop with one barber. When there are no customers the barber sleeps in his chair. If any customer enters he will wake up the barber and sit in the customer chair. If there are no chairs empty they wait in the waiting queue.
- ***Dining Philosophers problem:*** This problem states that there are K number of philosophers sitting around a circular table with one chopstick placed between each pair of philosophers. The philosopher will be able to eat if he can pick up two chopsticks that are adjacent to the philosopher.
- ***Readers and writers' problem:*** This problem occurs when many threads of execution try to access the same shared resources at a time. Some threads may read, and some may write. In this scenario, we may get faulty outputs.

EXPERIMENT-8

AIM: Write a program to implement Banker's algorithm for deadlock avoidance.

THEORY

It is a banker algorithm used to **avoid deadlock** and **allocate resources** safely to each process in the computer system. The '**S-State'** examines all possible tests or activities before deciding whether the allocation should be allowed to each process. It also helps the operating system to successfully share the resources between all the processes. The banker's algorithm is named because it checks whether a person should be sanctioned a loan amount or not to help the bank system safely simulate allocation resources.

CODE

```
alloc=( 0 1 0 2 0 0 3 0 2 2 1 1 0 0 2 )
max=( 7 5 3 3 2 2 9 0 2 2 2 2 4 3 3 )
 avail=(3 3 2) n=5
 m=3 declare -a f
 declare -a ans declare -a need ind=0
i=0
 while [ $i -lt $n ] do
   f[$i]=0 i=$((i+1))
 done a=0
 while [ $a -lt $n ] do
    b=0
   while [ $b -lt $m ] do
       need[ $(( a*m+b )) ]=$((max[ $(( a*m+b )) ]-alloc[ $(( a*m+b )) ])) b=$((b+1))
   done a=$((a+1))
 done p=0
 while [ $p -lt 5 ] do
    q=0
   while [ $q -lt $n ] do
   if [ ${f[$q]} -eq 0 ] then
         flag=0 r=0
         while [ $r -lt $m ] do
            if [ ${need[ $(( q*m+r )) ]} -gt ${avail[$r]} ] then
               flag=1 break
            fi r=$((r+1))
          done
         if [ $flag -eq 0 ] then
            ans[$ind]=$q ind=$((ind+1)) y=0
            while [ $y -lt $m ] do
               avail[$y]=$((avail[$y]+alloc[ $(( q*m+y )) ])) y=$((y+1))
            done f[$q]=1
```

```
        fi
      fi
      q=$((q+1)) done p=$((p+1))
done flag2=1 x=0
while [ $x -lt $n ] do
  if [ ${f[$x]} -eq 0 ] then
     flag2=0
     echo "The following systen is not safe" break
  fi x=$((x+1))
done
if [ $flag2 -eq 1 ] then
  echo "Following is the SAFE sequence"
  echo "P"${ans[0]} "--> P"${ans[1]} "--> P"${ans[2]} "--> P"${ans[3]} "--> P"${ans[4]}
fi
```

OUTPUT

VIVA-QUESTIONS

Q1. **What necessary conditions can lead to a deadlock situation in a system?**

Ans1. Conditions for Deadlock- Mutual Exclusion, Hold and Wait, No pre-emption, Circular wait. These 4 conditions must hold simultaneously for the occurrence of deadlock.

Q2. **What factors determine whether a detection-algorithm must be utilized in a deadlock avoidance system?**

Ans2. One is that it depends on how often a deadlock is likely to occur under the implementation of this algorithm. The other has to do with how many processes will be affected by deadlock when this algorithm is applied.

Q3. **What is a Safe State and its' use in deadlock avoidance?**

Ans3. A state is safe if the system can allocate resources to each process(up to its maximum requirement) in some order and still avoid a deadlock.

Q4. What are the Methods for Handling Deadlocks?

Ans4. There are mainly four methods for handling deadlock.

1. Deadlock ignorance
2. Deadlock prevention
3. Deadlock avoidance
4. Detection and recovery

Q5. What is resource allocation graph?

Ans5. The resource allocation graph is the pictorial representation of the state of a system. As its name suggests, the resource allocation graph is the complete information about all the processes which are holding some resources or waiting for some resources. It also contains the information about all the instances of all the resources whether they are available or being used by the processes.

Q6. What are the deadlock avoidance algorithms?

Ans6. A deadlock avoidance algorithm dynamically examines the resources allocation state to ensure that a circular wait condition case never exists. Where the resources allocation state is defined by the of available and allocated resources and the maximum demand of the process.

Q7. **Recovery from Deadlock?**

Ans7. When a Deadlock Detection Algorithm determines that a deadlock has occurred in the system, the system must recover from that deadlock. There are two approaches of breaking a Deadlock:

1. Process Termination
To eliminate the deadlock, we can simply kill one or more processes. For this, we use two methods:

(a). Abort all the Deadlocked Processes
Aborting all the processes will certainly break the deadlock, but with a great expense. The deadlocked processes may have computed for a long time and the result of those partial computations must be discarded and there is a probability to recalculate them later.

(b). Abort one process at a time until deadlock is eliminated
Abort one deadlocked process at a time, until deadlock cycle is eliminated from the system. Due to this method, there may be considerable overhead, because after aborting each process, we have to run deadlock detection algorithm to check whether any processes are still deadlocked.

2. Resource Pre-emption
To eliminate deadlocks using resource pre-emption, we pre-empt some resources from processes and give those resources to other processes. This method will raise three issues –

(a). Selecting a victim

We must determine which resources and which processes are to be pre-empted and also the order to minimize the cost.

(b). Rollback

We must determine what should be done with the process from which resources are pre-empted. One simple idea is total rollback. That means abort the process and restart it.

(c). Starvation

In a system, it may happen that same process is always picked as a victim. As a result, that process will never complete its designated task. This situation is called **Starvation** and must be avoided.

One solution is that a process must be picked as a victim only a finite number of times.

Q8. When does deadlock happen? How does Banker's algorithm avoid the deadlock condition?

Ans8. Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process. The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.