

Parul University

FACULTY OF ENGINEERING AND TECHNOLOGY

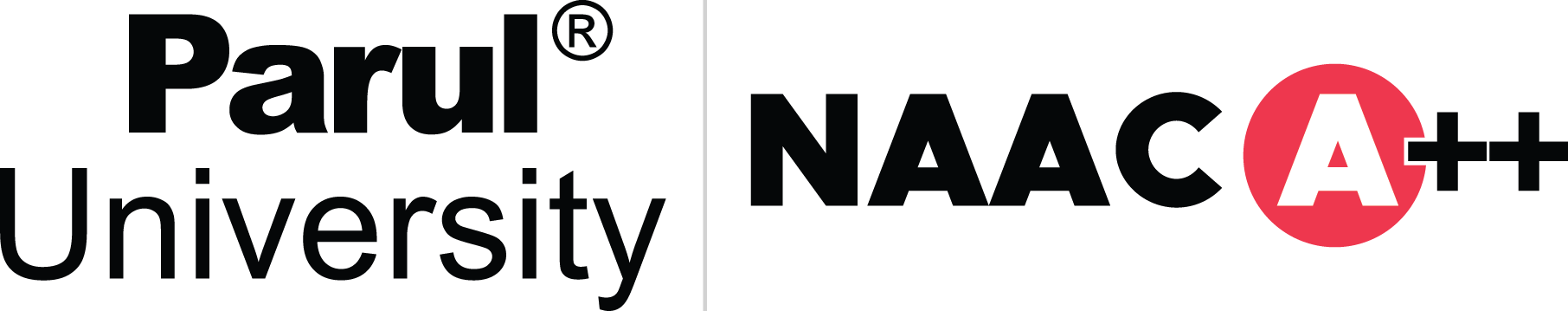
BACHELOR OF TECHNOLOGY

OPERATING SYSTEM LABORATORY

(303105252)

IV SEMESTER

Computer Science & Engineering Department



Laboratory Manual

Session 2023-24

CERTIFICATE

This is to Certify that

Mr./Ms. ㅤㅤㅤㅤㅤ VIJAY KUMAR ㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤ

With enrolment no. ㅤ 2203051050620 ㅤhas successfully completed his/her

Laboratory experiments in **Operating System laboratory (303105252)** From the

department of ㅤㅤㅤCOMPUTER SCIENCE AND ENGINEERING ㅤ ㅤㅤduring the academic year 2023 - 2024ㅤㅤ



Date of Submission : ………………………. Staff In Charge: ……………………….

Head of department: ……………………….

**TABLE OF CONTENT**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sr. No** | **Experiment Title** | **Page No** | | **Date of Start** | **Date of Completion** | **Sign** | **Marks (out of 10)** |
| **From** | **To** |
| **1.** | Study of Basic commands of Linux. | 6 | 12 | 30/11/23 | 30/11/23 |  |  |
| **2.** | Study the basics of shell programming. | 13 | 14 | 7/12/23 | 14/12/23 |  |  |
| **3.** | Write a Shell script to print given numbers sum of all digits. | 15 | 15 | 21/12/23 | 21/12/23 |  |  |
| **4.** | Write a shell script to validate the entered date. (eg. Date format is: dd-mm-yyyy). | 16 | 16 | 28/12/23 | 28/12/23 |  |  |
| **5.** | Write a shell script to check entered string is palindrome or not. | 17 | 17 | 04/01/24 | 11/01/24 |  |  |
| **6.** | Write a Shell script to say Good morning/Afternoon/Evening as you log in to system. | 18 | 18 | 18/01/24 | 25/01/24 |  |  |
| **7.** | Write a C program to create a child process | 19 | 19 | 01/02/24 | 01/02/24 |  |  |
| **8.** | Finding out biggest number from given three numbers supplied as command line arguments | 20 | 20 | 08/02/24 | 08/02/24 |  |  |
| **9.** | Printing the patterns using for loop | 21 | 21 | 15/02/24 | 22/02/24 |  |  |
| **10.** | Shell script to determine whether given file exist or not. | 22 | 22 | 29/02/24 | 07/03/24 |  |  |
| **11.** | Write a program for process creation using C. (Use of gcc compiler). | 23 | 23 | 14/03/24 | 14/03/24 |  |  |
| **12.** | Implementation of FCFS & Round Robin Algorithm. | 24 | 27 | 21/03/24 | 28/03/24 |  |  |
| **13.** | Implementation of Banker’s Algorithm. | 28 | 30 | 04/04/24 | 04/04/24 |  |  |

## **PRACTICAL – 1**

**AIM**: Study of Basic commands of Linux/UNIX.

Command shell:

A program that interprets commands is Command shell.

Shell Script: Allows a user to execute commands by typing them manually at a terminal, or automatically in programs called shell scripts. A shell is not an operating system. It is a way to interface with the operating system and run Commands.

BASH (Bourne Again Shell)

•Bash is a shell written as a free replacement to the standard Bourne Shell (/bin/sh) originally written by Steve Bourne for UNIX systems.

• It has all of the features of the original Bourne Shell, plus additions that make it easier to program with and use from the command line.

• Since it is Free Software, it has been adopted as the default shell on most Linux systems.

**BASIC LINUX COMMANDS:**

1. **pwd :**

Print Working Directory

**DESCRIPTION:**

pwd prints the full pathname of the current working directory.

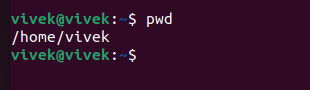
**SYNTAX:**

pwd

**EXAMPLE:**

$ pwd

**OUTPUT:**



1. **cd:**

Change Directory

**DESCRIPTION:**

It allows you to change your working directory. You use it to move around within the hierarchy of your file system.

**SYNTAX:**

cd directory\_name

**EXAMPLE:** To change into “work directory” in “KALI”

**OUTPUT:**



1. **cd ..**

**DESCRIPTION:**

Move up one directory.

**SYNTAX:**

cd ..

**EXAMPLE:**

If you are in work directory and want to go to documents then write

**OUTPUT:**

## 

## 

1. **ls :**

list all the files and directories

**DESCRIPTION:**

List all files and folders in the current directory in the column format.

**SYNTAX:**

ls [options]

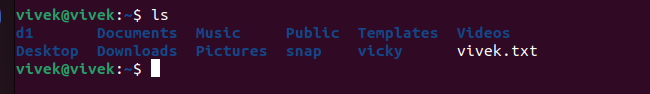
**EXAMPLE**:

Using various options

• Lists the total files in the directory and subdirectories, the names of the files in the current directory, their permissions, the number of subdirectories in directories listed, the size of the file, and the date of last modification. ls -l

• List all files including hidden files ls -a

**OUTPUT:**



1. **cat**

**DESCRIPTION:**

cat stands for "catenate".

It reads data from files, and outputs their contents.

It is the simplest way to display the contents of a file at the command line.

**SYNTAX:**

cat filename

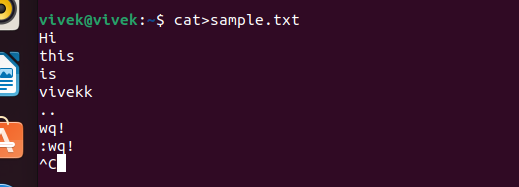
**EXAMPLES:**

• Print the contents of files mytext.txt and yourtext.txt cat mytext.txt yourtext.txt

• Print the cpu information using cat command cat /proc/cpuinfo

• Print the memory information using cat command cat /proc/meminfo

**OUTPUT:**



1. **head**

**DESCRIPTION:**

head, by default, prints the first 10 lines of each FILE to standard output. With more than one FILE, it precedes each set of output with a header identifying the file name. If no FILE is specified, or when FILE is specified as a dash ("-"), head reads from standard input.

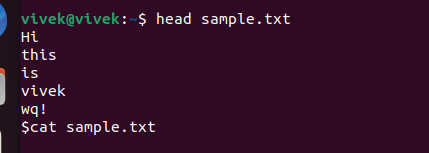
**SYNTAX:**

head [option]…[file/directory]

**EXAMPLE:**

Display the first ten lines of myfile.txt. head myfile.txt

**OUTPUT:**



1. **tail**

**DESCRIPTION:**

tail is a command which prints the last few number of lines (10 lines by default) of a certain file, then terminates.

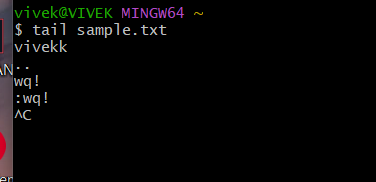
**VSYNTAX:**

tail [option]…[file/directory]

**EXAMPLE:**

Output the last 100 lines of the file myfile.txt. tail myfile.txt -n 100

**OUTPUT:**



1. **mv :**

Moving (and Renaming) Files

**DESCRIPTION**:

The mv command lets you move a file from one directory location to another. It also lets you rename a file (there is no separate rename command).

**SYNTAX: mv**

[option] source directory

**EXAMPLE:**

• Moves the file myfile.txt to the directory destination-directory. mv myfile.txt destination\_directory

**OUTPUT:**



1. **mkdir :**

Make Directory

**DESCRIPTION**:

If the specified directory does not already exist, mkdir creates it. More than one directory may be specified when calling mkdir.

**SYNTAX**:

mkdir [option] directory

**EXAMPLE:**

Create a directory named 210304124456. mkdir work

**OUTPUT:**



1. **cp :**

Copy Files

**DESCRIPTION:**

The cp command is used to make copy of files and directories.

**SYNTAX:**

cp [option] source directory

**EXAMPLE:**

Creates a copy of the file in the currently working directory named origfile. The copy will be named newfile, and will be located in the working directory. cp origfile newfile

**OUTPUT:**



1. **rmdir :**

Romove Directory

**DESCRIPTION:**

The rmdir command is used to remove a directory that contains other files or directories.

**SYNTAX**:

rm directory\_name

**EXAMPLE:**

Delete mydir directory along with all files and directories within that directory. Here, -r is for recursive and –f is for forcefully. rmdir -rf mydir

**OUTPUT:**



1. **echo**

**DESCRIPTION:**

Display text on the screen.

**SYNTAX:** echo yourtext

**EXAMPLE:** Print Hello World on the screen echo “Hello World”

**OUTPUT:**



1. **clear**

**DESCRIPTION:**

Used to clear the screen

**SYNTAX:** clear

**EXAMPLE:** Clear the entire screen

**OUTPUT:**

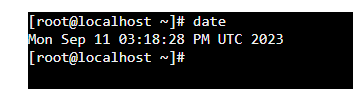


1. **date :**

**DESCRIPTION:**

display current date and time.

**OUTPUT:**

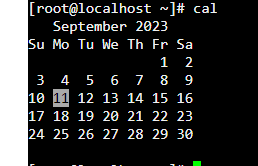
****

1. **cal:**

**DESCRIPTION:**

display current month calendar**.**

**OUTPUT:**

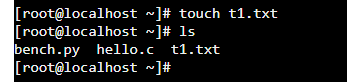
****

1. **touch:**

**DESCRIPTION:**

create a new file

**OUTPUT:**

****

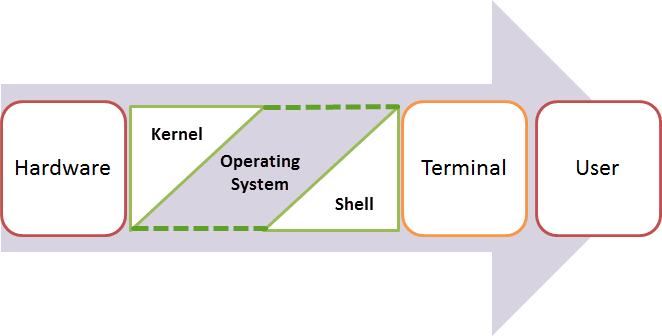
## **PRACTICAL – 2**

**AIM:** Study the basics of shell programming.

**What is a Shell?**

An Operating is made of many components, but its two prime components are -

* Kernel
* Shell



A Kernel is at the nucleus of a computer. It makes the communication between the hardware and software possible. While the Kernel is the innermost part of an operating system, a shell is the outermost one.

A shell in a Linux operating system takes input from you in the form of commands, processes it, and then gives an output. It is the interface through which a user works on the programs, commands, and scripts. A shell is accessed by a terminal which runs it.

When you run the terminal, the Shell issues **a command prompt (usually $),** where you can type your input, which is then executed when you hit the Enter key. The output or the result is thereafter displayed on the terminal.

The Shell wraps around the delicate interior of an Operating system protecting it from accidental damage. Hence the name **Shell**.

**What is Shell Scripting?**

Shell scripting is writing a series of command for the shell to execute. It can combine lengthy and repetitive sequences of commands into a single and simple script, which can be stored and executed anytime. This reduces the effort required by the end user.

Let us understand the steps in creating a Shell Script

1. **Create a file using** a **vi** editor(or any other editor). Name script file with **extension .sh**
2. **Start** the script with **#! /bin/sh**
3. Write some code.
4. Save the script file as filename.sh
5. For **executing** the script type **bash filename.sh**

"#!" is an operator called shebang which directs the script to the interpreter location. So, if we use"#!

/bin/sh" the script gets directed to the bourne-shell.

**PROGRAM TO UNDERSTAND BASIC OF SHELL PROGRAMING:**

#!/bin/sh

x=10

y=11

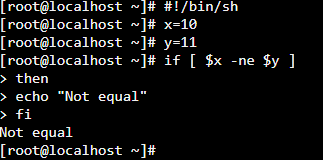
if [ $x-ne $y ]

then

echo "Not equal"

fi

**OUTPUT:**

****

## **PRACTICAL – 3**

**AIM:** Write a shell script to print given numbers sum of all digits

**CODE:**

#!/bin/bash

echo "Enter a Number:"

read n

temp=$n

sd=0

sum=0

while [ $n-gt 0 ]

do

sd=$(($n % 10))

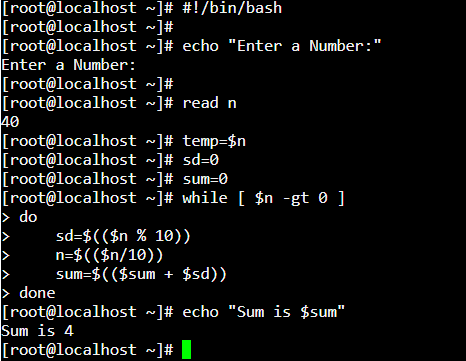
n=$(($n/10))

sum=$(($sum + $sd))

done

echo "Sum is $sum"

**OUTPUT:**



## **PRACTICAL – 4**

**AIM:** Write a shell script to validate the entered date. (eg. Date format is: dd-mm-yyyy).

**CODE:**

echo "Enter Valid Date"

read date

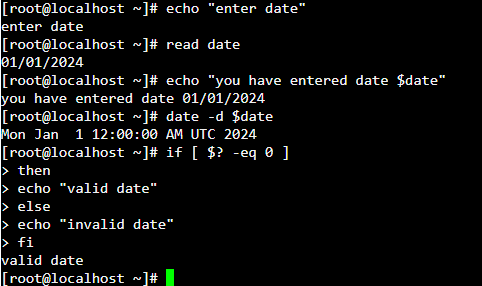
echo "You have entered $date" date -d $date if [ $? -eq 0]

then

echo "Enter Date is Valid" else

echo "Enter Date is Invalid" fi

**OUTPUT:**



## **PRACTICAL – 5**

**AIM:** Write a shell script to check entered string is palindrome or not**.**

**CODE:**

#!/bin/bash read name

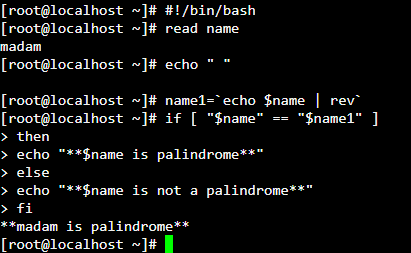
echo ""

name1=`echo $name | rev if [ "$name" == "$name1" ] then

echo "\*\*$name is palindrome\*\*" else

echo "\*\*$name is not a palindrome\*\*" fi

**OUTPUT:**



## **PRACTICAL – 6**

**AIM:** Write a Shell script to say Good morning/Afternoon/Evening as you log in to system

**CODE:**

#!/bin/bash

hour=`date +%H`

if [ $hour -lt 12 ] # if hour is less than 12

then

echo "GOOD MORNING WORLD"

elif [ $hour -le 16 ] # if hour is less than equal to 16

then

echo "GOOD AFTERNOON WORLD"

elif [ $hour -le 20 ] # if hour is less than equal to 20

then

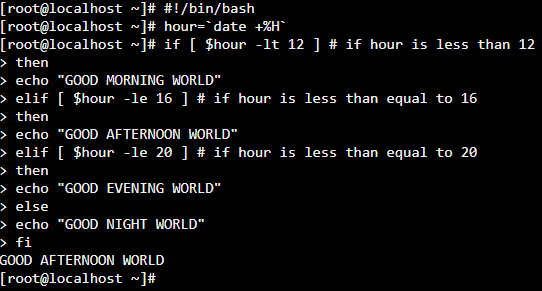
echo "GOOD EVENING WORLD"

else

echo "GOOD NIGHT WORLD"

fi

**OUTPUT:**



## **PRACTICAL – 7**

**AIM:** Write a shell script to check entered string is palindrome or not**.**

**CODE:**

#include <stdio.h>

#include <sys/wait.h> /\* contains prototype for wait \*/

int main(void)

{

int pid;

int status;

printf("Hello World!\n");

pid = fork( );

if(pid == -1) /\* check for error in fork \*/

{

perror("bad fork");

exit(1);

}

if (pid == 0)

printf("I am the child process.\n");

else

{

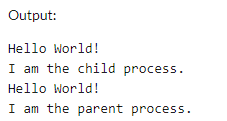
wait(&status); /\* parent waits for child to finish \*/

printf("I am the parent process.\n");

}

}

**OUTPUT:**



## **PRACTICAL – 8**

**AIM:** Find out the biggest number from given three numbers supplied as command line arguments.

**CODE:**

#!/bin/bash

echo "enter number 1"

read n1

echo "enter number 2"

read n2

echo "enter number 3"

read n3

if [ $n1 -gt $n2 ] && [ $n1 -gt $n3 ]

then

echo "Number 1 is biggest: $n1"

elif [ $n2 -gt $n1 ] && [ $n2 -gt $n3 ]

then

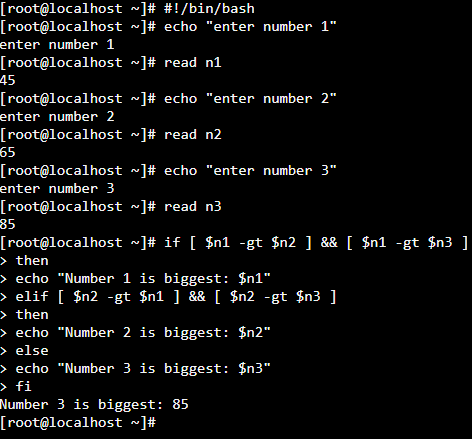
echo "Number 2 is biggest: $n2"

else

echo "Number 3 is biggest: $n3"

fi

**OUTPUT:**



## **PRACTICAL – 9**

**AIM:** Printing patterns using loops.

**CODE:**

#! /bin/sh

for ((i=1;i<=5;i++)); # Loop to create 5 lines of text

do

for((k=1;k<=(5-i);k++)); # Loop to padd prepending spaces

do

printf "%s" " ";

done;

for ((j=1;j<=i;j++)); # Loop to create asterix

do

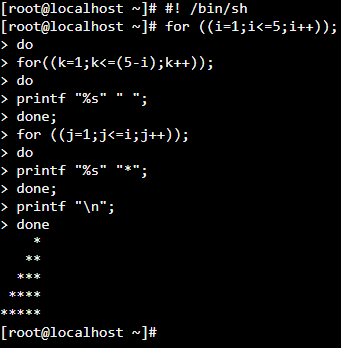
printf "%s" "\*";

done;

printf "\n"; # Print the carriage return

done

**OUTPUT:**



## **PRACTICAL – 10**

**AIM:** Shell script to determine whether given file exist or not.

**CODE:**

#!/bin/bash

if [ -f "File.txt" ];

then

# if file exist the it will be printed

echo "File is exist"

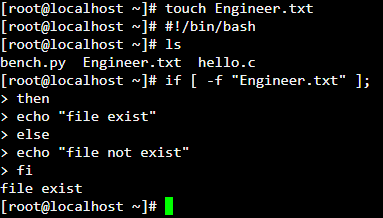
else

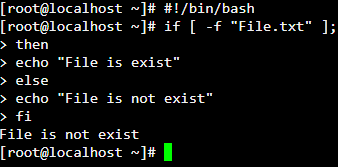
# is it is not exist then it will be printed

echo "File is not exist"

fi

**OUTPUT:**

****



## **PRACTICAL – 11**

**AIM:** Write a program for process creation using C. (Use of gcc compiler)..

**CODE:**

#include <stdio.h>

#include <sys/wait.h> /\* contains prototype for wait \*/

int main(void)

{

int pid;

int status;

printf("PROCESS CREATION\n");

pid = fork( );

if(pid == -1) /\* check for error in fork \*/

{

perror("bad fork");

exit(1);

}

if (pid == 0)

printf("I am the child process.\n");

else

{

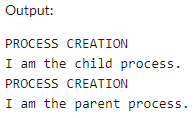
wait(&status); /\* parent waits for child to finish \*/

printf("I am the parent process.\n");

}

}

**OUTPUT:**



## **PRACTICAL – 12**

**AIM:** Implementation of FCFS & Round Robin Algorithm

**CODE:**

**Practical 12.1:** FCFS

#include<stdio.h>

int main(){

int bt[10]={0},at[10]={0},tat[10]={0},wt[10]={0},ct[10]={0};

int n,sum=0;

float totalTAT=0,totalWT=0;

printf("Enter number of processes");

scanf("%d",&n);

printf("Enter arrival time and burst time for each process\n\n");

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

{

printf("Arrival time of process[%d] ",i+1);

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

printf("Burst time of process[%d] ",i+1);

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

printf("\n");

}

//calculate completion time of processes

for(int j=0;j<n;j++)

{

sum+=bt[j];

ct[j]+=sum;

}

//calculate turnaround time and waiting times

for(int k=0;k<n;k++)

{

tat[k]=ct[k]-at[k];

totalTAT+=tat[k];

}

for(int k=0;k<n;k++)

{

wt[k]=tat[k]-bt[k];

totalWT+=wt[k];

}

printf("Solution: \n\n");

printf("P#\t AT\t BT\t CT\t TAT\t WT\t\n\n");

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

{

printf("P%d\t %d\t %d\t %d\t %d\t %d\n",i+1,at[i],bt[i],ct[i],tat[i],wt[i]);

}

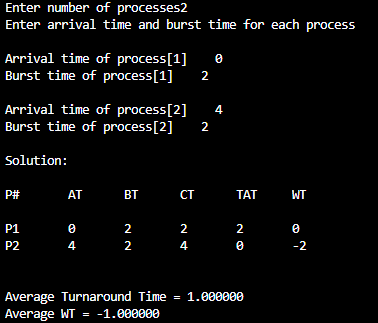
printf("\n\nAverage Turnaround Time = %f\n",totalTAT/n);

printf("Average WT = %f\n\n",totalWT/n);

return 0;

}

**OUTPUT:**



**Practical 12.2:** Round Robin Algorithm

#include<stdio.h>

int main()

{

int count,j,n,time,remain,flag=0,time\_quantum;

int wait\_time=0,turnaround\_time=0,at[10],bt[10],rt[10];

printf("Enter Total Process:\t ");

scanf("%d",&n);

remain=n;

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

{

printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);

scanf("%d",&at[count]);

scanf("%d",&bt[count]);

rt[count]=bt[count];

}

printf("Enter Time Quantum:\t");

scanf("%d",&time\_quantum);

printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");

for(time=0,count=0;remain!=0;)

{

if(rt[count]<=time\_quantum && rt[count]>0)

{

time+=rt[count];

rt[count]=0;

flag=1;

}

else if(rt[count]>0)

{

rt[count]-=time\_quantum;

time+=time\_quantum;

}

if(rt[count]==0 && flag==1)

{

remain--;

printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);

wait\_time+=time-at[count]-bt[count];

turnaround\_time+=time-at[count];

flag=0;

}

if(count==n-1)

count=0;

else if(at[count+1]<=time)

count++;

else

count=0;

}

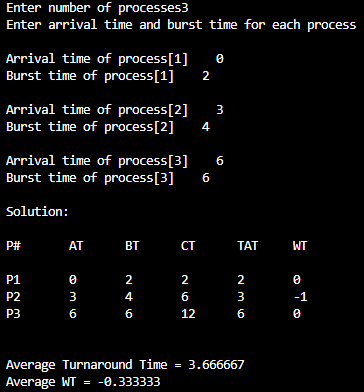
printf("\nAverage Waiting Time= %f\n",wait\_time\*1.0/n);

printf("Avg Turnaround Time = %f",turnaround\_time\*1.0/n);

return 0;

}

**OUTPUT:**



## **PRACTICAL – 13**

**AIM:** Implementation of Banker’s Algorithm.

**CODE:**

/\*<--------------PREPROCESSING STATEMENTS--------------->\*/

#include <stdio.h>

#include <stdlib.h>

/\*<--------------MAIN FUNCTION--------------->\*/

int main()

{

int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];

int p, r, i, j, process, count;

count = 0;

printf("Enter the no of processes : ");

scanf("%d", &p);

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

completed[i] = 0;

printf("\n\nEnter the no of resources : ");

scanf("%d", &r);

printf("\n\nEnter the Max Matrix for each process : ");

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

{

printf("\nFor process %d : ", i + 1);

for(j = 0; j < r; j++)

scanf("%d", &Max[i][j]);

}

printf("\n\nEnter the allocation for each process : ");

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

{

printf("\nFor process %d : ",i + 1);

for(j = 0; j < r; j++)

scanf("%d", &alloc[i][j]);

}

printf("\n\nEnter the Available Resources : ");

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

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

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

for(j = 0; j < r; j++)

need[i][j] = Max[i][j] - alloc[i][j];

do

{

printf("\n Max matrix:\tAllocation matrix:\n");

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

{

for( j = 0; j < r; j++)

printf("%d ", Max[i][j]);

printf("\t\t");

for( j = 0; j < r; j++)

printf("%d ", alloc[i][j]);

printf("\n");

}

process = -1;

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

{

if(completed[i] == 0)//if not completed

{

process = i ;

for(j = 0; j < r; j++)

{

if(avail[j] < need[i][j])

{

process = -1;

break;

}

}

}

if(process != -1)

break;

}

if(process != -1)

{

printf("\nProcess %d runs to completion!", process + 1);

safeSequence[count] = process + 1;

count++;

for(j = 0; j < r; j++)

{

avail[j] += alloc[process][j];

alloc[process][j] = 0;

Max[process][j] = 0;

completed[process] = 1;

}

}

}while(count != p && process != -1);

if(count == p)

{

printf("\nThe system is in a safe state!!\n");

printf("Safe Sequence : < ");

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

printf("%d ", safeSequence[i]);

printf(">\n");

}

else

printf("\nThe system is in an unsafe state!!");

}

**OUTPUT:**

