



Parul University

FACULTY OF ENGINEERING AND
TECHNOLOGY BACHELOR OF TECHNOLOGY

**Operating
System**
(203105204)

IV SEMESTER
Computer Science &
Engineering Department

Laboratory Manual

CERTIFICATE

This is to certify that

Mr. **AAKASH SHARMA** with enrollment no. **200303105256** has
successfully completed his laboratory experiments in the Operating system
(**203105204**) from the department of **Computer Science And**
Engineering during the academic
year **2021-22**



Date of Submission:.....

Staff In charge:.....

Head Of Department:.....

TABLE OF CONTENT

Sr. No	Experiment Title	Page No		Date of Start	Date of Completion	Marks (out of 10)	Sign
		To	From				
1	Study of Basic commands of Linux.	7	1				
2	Study the basics of shell programming.	11	8				
3	Write a Shell script to print given numbers sum of all digits.	12	12				
4	Write a shell script to validate the entered date. (eg. Date format is: dd-mm-yyyy).	13	13				
5	a) Write a shell script to check entered string is palindrome or not. b) Write a Shell script to say Good morning/Afternoon/Evening as you log in to system.	15	14				
6	Write a C program to create a child process.	17	16				
7	Finding out biggest number from given three numbers supplied as command line arguments	18	18				
8	Printing the patterns using for loop.	19	19				
9	Shell script to determine whether given file exist or not.	20	20				
10	Write a program for process creation using C. (Use of gcc compiler).	23	21				
11	Implementation of FCFS & Round Robin Algorithm.	30	24				
12	Implementation of Banker's Algorithm.	33	31				

Experiment-1

AIM:- Study of Basic commands of Linux

1. pwd Command

The [pwd](#) command is used to display the location of the current working directory.

Syntax: pwd

output:

```
aakash@aakash-VirtualBox:~$ pwd
/home/aakash
```

2. mkdir Command

The [mkdir](#) command is used to create a new directory under any directory.

Syntax:

mkdir <directory name>

output:

```
aakash@aakash-VirtualBox:~$ mkdir demo
aakash@aakash-VirtualBox:~$
```

3. rmdir Command

The [rmdir](#) command is used to delete a directory.

Syntax:

rmdir <directory name>

output:

```
aakash@aakash-VirtualBox:~$ rmdir demo
aakash@aakash-VirtualBox:~$ ls
```

4. ls Command

The [ls](#) command is used to display a list of content of a directory.

Syntax:

ls

output:

```
aakash@aakash-VirtualBox:~$ ls
4          Documents      Music          swapg.sh
5          Downloads     oddoreven.sh  switch.sh
Aakash    factorial.sh          oneto100.sh  Templates
apple     fact.sh              Pictures      test1.sh
average.sh gangu.sh            printstar.sh test.sh
butterfly.sh greatestnobetnthreeno.sh pu           Videos
demo      halfpyramidafter180rotation.sh Public
Desktop   laptop              pyramid.sh
```

5. cd Command

The [cd](#) command is used to change the current directory.

Syntax:

cd <directory name>

output:

```
aakash@aakash-VirtualBox:~$ cd demo
aakash@aakash-VirtualBox:~/demo$
```

6. touch Command

The [touch](#) command is used to create empty files. We can create multiple empty files by executing it once.

Syntax:

touch <file name> touch

<file1> <file2>

output:

```
aakash@aakash-VirtualBox:~/demo$ touch test.txt
aakash@aakash-VirtualBox:~/demo$
```

7. cat Command

The [cat](#) command is a multi-purpose utility in the Linux system. It can be used to create a file, display content of the file, copy the content of one file to another file, and more.

Syntax:

cat [OPTION]... [FILE]..

output:

```
aakash@aakash-VirtualBox:~/demo$ cat >demo1.txt
Namaste London!!!
^C
aakash@aakash-VirtualBox:~/demo$ cat demo1.txt
Namaste London!!!
```

8. rm Command

The [rm](#) command is used to remove a file.

Syntax:

rm <file name>

output:

```
aakash@aakash-VirtualBox:~/demo$ rm demo3.txt
aakash@aakash-VirtualBox:~/demo$
```

9. cp Command

The [cp](#) command is used to copy a file or directory.

Syntax:

cp <existing file name> <new file name>

output:

```
aakash@aakash-VirtualBox:~/demo$ cat demo1.txt
Namaste London!!!
aakash@aakash-VirtualBox:~/demo$ cat >demo2.txt
^C
aakash@aakash-VirtualBox:~/demo$ cp demo1.txt demo2.txt
aakash@aakash-VirtualBox:~/demo$ cat demo2.txt
Namaste London!!!
```

10. mv Command

The [mv](#) command is used to move a file or a directory from one location to another location.

Syntax:

mv <file name> <directory path>

output:

```
aakash@aakash-VirtualBox:~/demo$ mv demo1.txt test
aakash@aakash-VirtualBox:~/demo$ cat test
Namaste London!!!
```

11. rename Command

The [rename](#) command is used to rename files. It is useful for renaming a large group of files.

Syntax:

rename 's/old-name/new-name/' files

output:

```
aakash@aakash-VirtualBox:~/demo$ rename 's/\.txt$/\.pdf/' *.txt
aakash@aakash-VirtualBox:~/demo$ ls
demo2.pdf  test  test.pdf
aakash@aakash-VirtualBox:~/demo$
```

12. head Command

The [head](#) command is used to display the content of a file. It displays the first 10 lines of a file.

Syntax:

head <file name>

output:

```
aakash@aakash-VirtualBox:~/demo$ cat >apple.txt
a
b
c
d
e
f
g
h
i
j
k
l^C
aakash@aakash-VirtualBox:~/demo$ head apple.txt
a
b
c
d
e
f
g
h
i
j
```

13. tail <file name>

The [tail](#) command is similar to the head command. The difference between both commands is that it displays the last ten lines of the file content. It is useful for reading the error message.

Syntax:

tail <file name>

output:

```
aakash@aakash-VirtualBox:~/demo$ tail apple.txt
b
c
d
e
f
g
h
i
j
k
```

14. tac Command

The [tac](#) command is the reverse of cat command, as its name specified. It displays the file content in reverse order (from the last line).

Syntax:

tac <file name>

output:

```
aakash@aakash-VirtualBox:~/demo$ tac apple.txt
k
j
i
h
g
f
e
d
c
b
a
```

15. date Command

The [date](#) command is used to display date, time, time zone, and more.

Syntax: date**output:**

```
aakash@aakash-VirtualBox:~/demo$ date
Tuesday 15 February 2022 09:37:21 PM IST
```

16. su Command

The [su](#) command provides administrative access to another user. In other words, it allows access of the Linux shell to another user.

Syntax:

su <user name>

output:

```
aakash@aakash-VirtualBox:~/demo$ su aakash
Password:
```

17. passwd Command

The [passwd](#) command is used to create and change the password for a user.

Syntax:

passwd <username>

output:

```
aakash@aakash-VirtualBox:~/demo$ passwd aakash
Changing password for aakash.
Current password:
New password:
Retype new password:
passwd: password updated successfully
```

18. grep Command

The [grep](#) is the most powerful and used filter in a Linux system. The 'grep' stands for "**global regular expression print.**" It is useful for searching the content from a file. Generally, it is used with the pipe.

Syntax:

command | grep <searchWord>

output:

```
aakash@aakash-VirtualBox:~/demo$ cat demo2.pdf | grep Namaste
Namaste London!!!
```

19. wc Command

The wc command is used to count the lines, words, and characters in a file.

Syntax:

wc <file name>

output:

```
aakash@aakash-VirtualBox:~/demo$ wc demo2.pdf
1  2 18 demo2.pdf
```

20. clear Command

Linux **clear** command is used to clear the terminal screen.

Syntax: clear

output:

```
aakash@aakash-VirtualBox:~/demo$ cat demo2.pdf | grep Namaste
Namaste London!!!
aakash@aakash-VirtualBox:~/demo$ wc demo2.pdf
1  2 18 demo2.pdf
aakash@aakash-VirtualBox:~/demo$ clear
```

Experiment-2

AIM:- Study the basics of shell programming.

1.Program Code ECHO:

```
echo "What is your name?"  
read PERSON echo "Hello,  
$PERSON"
```

Output:

```
aakash@aakash-VirtualBox:~/demo$ gedit practice1.sh  
aakash@aakash-VirtualBox:~/demo$ bash practice1.sh  
What is your name?  
Aakash shah  
hello,Aakash shah
```

2.Program Code SWAP:

```
echo "value of a->"  
read a echo "value of b->"  
read b temp=$a a=$b b=$temp
```

```
echo "a=$a b=$b"
```

Output:

```
aakash@aakash-VirtualBox:~$ bash swapg.sh
value of a->
10
value of b->
30
a=30 b=10
```

3.Program Code FACTORIAL:

```
echo "Enter the factorial of number you want->"
```

```
read n fact=1
```

```
while [ $n -ne 0 ]
```

```
do
```

```
fact=$((fact * $n))
```

```
n=$((n - 1))
```

```
done
```

```
echo "$fact"
```

Output:

```
aakash@aakash-VirtualBox:~$ bash factorial.sh
Enter the factorial of number you want->
5
120
```

4.Program Code WHILE LOOP:

```
i=1
```

```
while [ $i -lt 10 ]
```

```
do echo $i
```

```
i=$((i+1))
```

```
done
```

Output:

```
aakash@aakash-VirtualBox:~$ gedit while.sh
aakash@aakash-VirtualBox:~$ bash while.sh
1
2
3
4
5
6
7
8
9
10
```

5.Program code Average:

```
echo "write value of a:"
read a echo "write value of b:"
read b echo "write value of c:"
read c
sum=$(( $a + $b + $c ))
avg=$(( $sum / 3 ))
echo $avg
```



COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING & TECHNOLOGY
Operating System (203105204) B. Tech. 2nd YEAR
ENROLLMENT NO: 200303105256

Output:

```
aakash@aakash-VirtualBox:~$ bash average.sh
write value of a:
5
write value of b:
10
write value of c:
15
10
aakash@aakash-VirtualBox:~$
```

Experiment-3

AIM:- Write a Shell script to print given numbers sum of all digits.

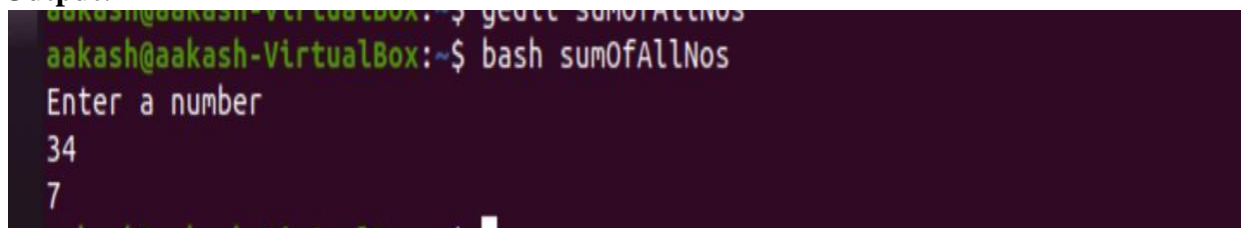
Program Code:

```
echo "Enter a number"
read num sum=0

while [ $num -gt 0 ] do
    mod=$((num % 10))
    sum=$((sum + mod))
    num=$((num / 10)) done

echo $sum
```

Output:



```
aakash@aakash-VirtualBox:~$ bash sumOfAllNos
Enter a number
34
7
```

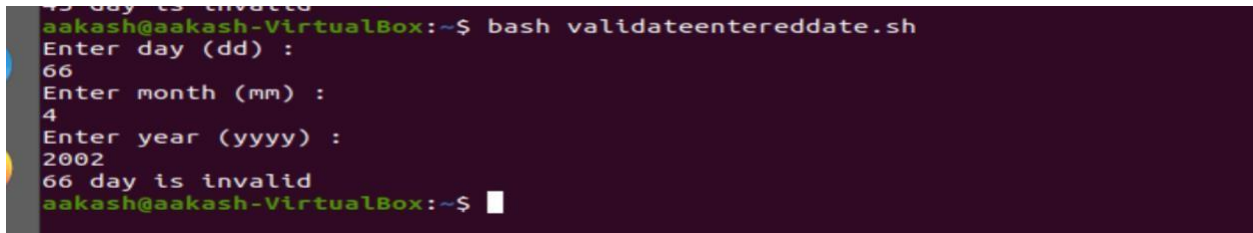

Experiment-4

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

Program Code:

```
echo "Enter day (dd) : "  
read d  
echo "Enter month (mm) : "  
read m  
echo "Enter year (yyyy) : "  
read y  
  
if [ $m -le 0 -o $m -gt 12 ] then  
    echo "$m is invalid month."  
    exit 1 fi  
  
if [ $d -le 0 -o $d -gt 30 ] then  
    echo "$d day is invalid"  
    exit 3 fi  
  
echo "$d/$m/$y is a vaild date"
```

Output:



```
66 day is invalid  
aakash@aakash-VirtualBox:~$ bash validateentereddate.sh  
Enter day (dd) :  
66  
Enter month (mm) :  
4  
Enter year (yyyy) :  
2002  
66 day is invalid  
aakash@aakash-VirtualBox:~$
```

Experiment-5A

A)AIM:- Write a shell script to check entered string is palindrome or not.

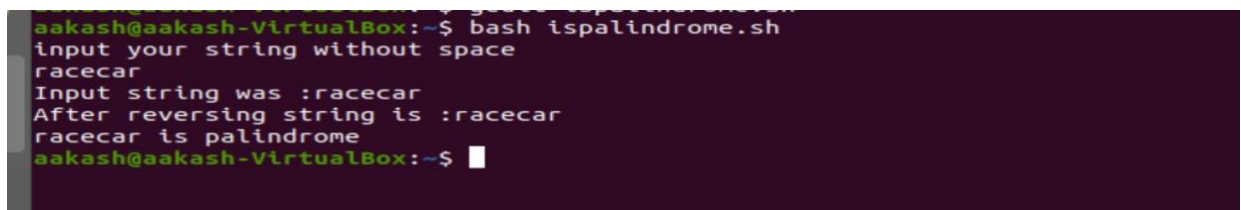
Program Code:

```
echo "input your string without space" read
vstr
for i in $(seq 0 ${#vstr})
do
    rvstr=${vstr:$i:1}${rvstr}
done

echo "Input string was :"$vstr echo
"After reversing string is :"$rvstr

if [ "$vstr" = "$rvstr" ]
then
    echo $vstr is palindrome
else
    echo $vstr is not plaindrome
fi
```

Output:



```
aakash@aakash-VirtualBox:~$ bash ispalindrome.sh
input your string without space
racecar
Input string was :racecar
After reversing string is :racecar
racecar is palindrome
aakash@aakash-VirtualBox:~$
```

Experiment-5B

B)AIM-: Write a Shell script to say Good morning/Afternoon/Evening as you log in to system.

Program Code:

```
h=$(date +"%H")  
  
if  
  
[ $h -gt 6 -a $h -le 12 ]  
  
then  
  
echo good morning  
  
elif  
  
[ $h -gt 12 -a $h -le 16 ]  
  
then  
  
echo good afternoon  
  
elif [ $h -gt 16 -a $h -le 20 ]  
  
then  
  
echo good evening  
  
else  
  
echo good night fi
```

Output:

```
aakash@aakash-VirtualBox:~$ date  
Wednesday 16 February 2022 10:18:19 AM IST  
aakash@aakash-VirtualBox:~$ bash greeting.sh  
good morning  
aakash@aakash-VirtualBox:~$
```

Practical-6

AIM: Write a C program to create a child process

Code:

```
#include<stdio.h>

#include<sys/wait.h>

#include<stdlib.h>

#include<sys/types.h>

#include<unistd.h>

int main(void)

{

int pid;

int status;

printf("hello World!!\n");

pid=fork();

if(pid == -1)

{

perror("bad fork!!");

exit(1);

}

if(pid == 0)

printf("I am the child process=%d\n",getpid());

else

{

printf("I am the parent process=%d\n",getppid());

}
```

}

Output:

```
ashok@kali: ~/Documents/oslab
File Actions Edit View Help

(ashok@kali)-[~/Documents/oslab]
$ gcc child.c -o child

(ashok@kali)-[~/Documents/oslab]
$ ./child
Hello World!
I am the child process.
I am the parent process.
```

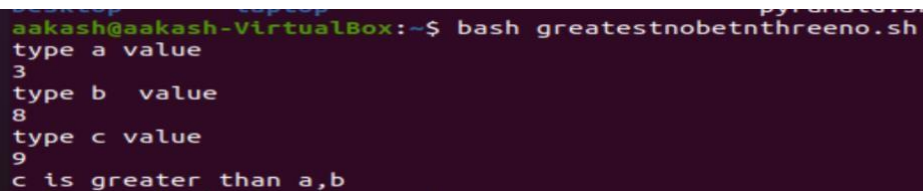
Practical-7

AIM:-Finding out biggest number from given three numbers supplied as command line arguments

Code:

```
echo "Enter 3 numbers: "  
read a read  
b read c  
if [ $a -gt $b ] && [ $a -gt $c ]  
then echo " " echo "Greatest  
num :$a" elif [ $b -gt $a ] && [  
$b -gt $c ]  
then echo  
" "  
echo "Greatest num :$b"  
else echo  
" "  
echo "Greatest num :$c" fi
```

Output:



```
aakash@aakash-VirtualBox:~$ bash greatestnobetnthreeno.sh  
type a value  
3  
type b value  
8  
type c value  
9  
c is greater than a,b
```

Practical-8

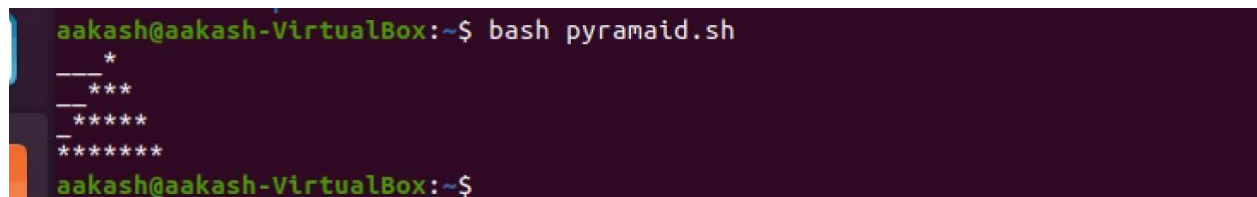
AIM:-Printing the patterns using for loop

Code:

```
n=4
for((i=1;i<=$n;i++)) do
for((j=1;j<=$n-$i;j++))
do echo -n "_" done
for((j=1;j<=2*$i-1;j++)) do
echo -n "*"

done echo
done
```

output:



```
aakash@aakash-VirtualBox:~$ bash pyramid.sh
  *
 ***
*****
aakash@aakash-VirtualBox:~$
```


Experiment-9

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

Program Code:

```
echo Enter file name ?  
read a  
if [ -f $a ]  
then  
    echo $a exists  
else  
    echo $a doesnt exists  
fi
```

Output:



```
aakash@aakash-VirtualBox:~$ gedit isfileexit.sh  
aakash@aakash-VirtualBox:~$ bash isfileexit.sh  
Enter file name ?  
aaku.txt  
aaku.txt doesnt exists  
aakash@aakash-VirtualBox:~$
```


Experiment-10

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

Program:

```
#include<sys/types.h >

#include<stdio.h>

#include<unistd.h>

int main()

{

int pid_t,pid,pid1,p,p1;

pid=fork();

if(pid== -1)

{

printf("enter in connection: ");

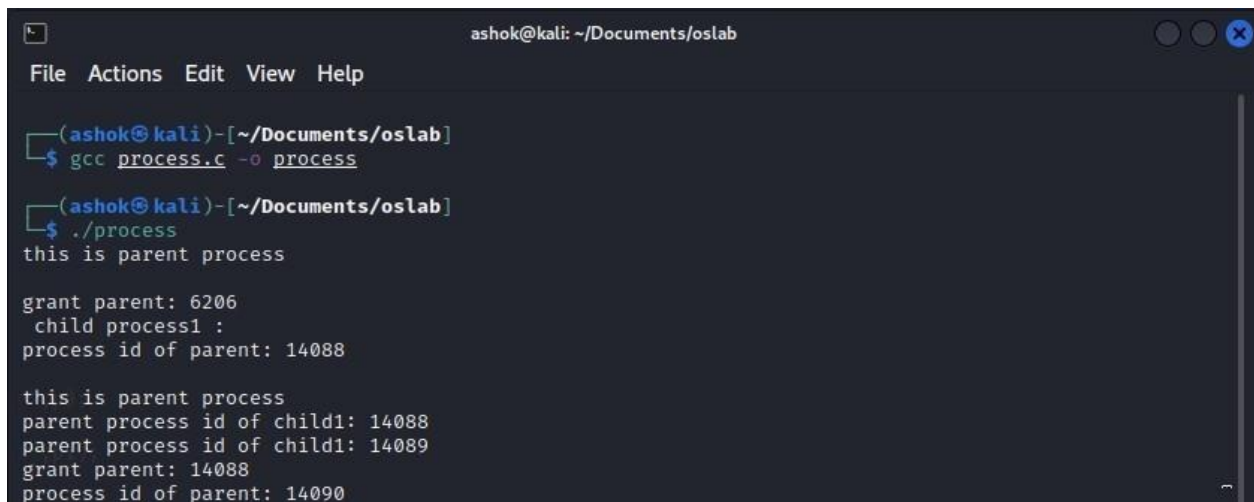
}

elseif(pid == 0)
```

```
{  
  
printf("\nchild process1 :\n\n");  
  
p=getppid();  
  
printf("parent process id of child1:%d\n",p);  
  
p1=getpid();  
  
printf("parent process id of child1: %d\n",p1);  
  
}  
  
else  
  
{  
  
pid1= fork();  
  
if(pid ==0)  
  
{  
  
printf("\nchild process 2: \n\n");  
  
p=getppid();  
  
printf("parent process id of child2: %d\n",p);  
  
p1=getpid();  
  
printf("parent process id of child2: %d\n",p1);  
  
}  
  
else  
  
{  
  
printf("this is parent process \n");  
  
p=getppid();  
  
printf("grant parent: %d \n",p);
```

```
p1=getpid();  
  
printf("process id of parent: %d \n",p1);  
  
}  
  
}  
  
return 0;  
  
}
```

Output:



```
ashok@kali: ~/Documents/oslab  
File Actions Edit View Help  
  
(ashok@kali)-[~/Documents/oslab]  
$ gcc process.c -o process  
  
(ashok@kali)-[~/Documents/oslab]  
$ ./process  
this is parent process  
  
grant parent: 6206  
child process1 :  
process id of parent: 14088  
  
this is parent process  
parent process id of child1: 14088  
parent process id of child1: 14089  
grant parent: 14088  
process id of parent: 14090
```

Experiment-11

AIM:-Implementation of FCFS & Round Robin Algorithm.

Code:

```
print("FIRST COME FIRST SERVE SCHEDULLING")

n= int(input("Enter number of processes : "))

d = dict()

for i in range(n):

    key = "P"+str(i+1)

    a = int(input("Enter arrival time of process"+str(i+1)+" :

    "))

    b = int(input("Enter burst time of process"+str(i+1)+" :

    "))

    l = []

    l.append(a)

    l.append(b)
```

```
d[key] = l

d = sorted(d.items(), key=lambda item:

item[1][0])

ET = []

for i in range(len(d)):

# first process

if(i==0):

    ET.append(d[i][1][1])

# get prevET + newBT

else:

    ET.append(ET[i-1] + d[i][1][1])

TAT = []

for i in range(len(d)):

    TAT.append(ET[i] - d[i][1][0])

WT = []

for i in range(len(d)):

    WT.append(TAT[i] - d[i][1][1])

avg_WT = 0

for i in WT:

    avg_WT +=i avg_WT = (avg_WT/n)

print("Process | Arrival | Burst | Exit | Turn Around |

Wait |")

for i in range(n):
```

```
print(" ",d[i][0]," | ",d[i][1][0]," | ",d[i][1][1]," |  
",ET[i]," | ",TAT[i]," | ",WT[i]," | ")  
  
print("Average Waiting Time: ",avg_WT)
```

Output:

```
aakash@aakash-VirtualBox:~/Desktop$ ls  
FCFS.py  
aakash@aakash-VirtualBox:~/Desktop$ python FCFS.py  
FIRST COME FIRST SERVE SCHEDULLING  
Enter number of processes : 4  
Enter arrival time of process1: 1  
Enter burst time of process1: 5  
Enter arrival time of process2: 0  
Enter burst time of process2: 4  
Enter arrival time of process3: 3  
Enter burst time of process3: 3  
Enter arrival time of process4: 2  
Enter burst time of process4: 5  
Process | Arrival | Burst | Exit | Turn Around | Wait |  
( ' , ' , ' P2 ' , ' | ' , 0 , ' | ' , 4 , ' | ' , 4 , ' | ' , 4 , ' | ' ,  
0 , ' , ' )  
( ' , ' , ' P1 ' , ' | ' , 1 , ' | ' , 5 , ' | ' , 9 , ' | ' , 8 , ' | ' ,  
3 , ' , ' )  
( ' , ' , ' P4 ' , ' | ' , 2 , ' | ' , 5 , ' | ' , 14 , ' | ' , 12 , ' |  
' , 7 , ' , ' )  
( ' , ' , ' P3 ' , ' | ' , 3 , ' | ' , 3 , ' | ' , 17 , ' | ' , 14 , ' |  
' , 11 , ' , ' )  
( ' Average Waiting Time : ' , 5 )  
aakash@aakash-VirtualBox:~/Desktop$
```

Program Code (ROUND ROBIN):

```
def findWaitingTime(processes,  
                    n, bt, wt, quantum):  
    rem_bt = [0] * n  
    # Copy the burst time into  
    rt[]  
    for i in range(n):  
        rem_bt[i] = bt[i] t = 0  
    # Current time  
    # Keep traversing processes in round
```

robin manner until all of them

are

not done.

while(1):

done = True

Traverse all processes one by

one repeatedly

for i in range(n):

If burst time of a process is greater

than 0 then only need to process further

if (rem_bt[i] > 0) :

done = False # There is a pending process

if (rem_bt[i] > quantum) :

Increase the value of t i.e. shows

how much time a process has been processed

t += quantum

Decrease the burst_time of current

process by quantum

rem_bt[i] -= quantum

If burst time is smaller than or equal

to quantum. Last cycle for this process

else:

Increase the value of t i.e. shows

```
# how much time a process has been processed  
t = t + rem_bt[i]
```

```
# Waiting time is current time minus  
# time used by this process  
wt[i] = t - bt[i]
```

```
# As the process gets fully executed  
# make its remaining burst time = 0  
rem_bt[i] = 0  
# If all processes are done  
if (done == True):  
    break
```

```
# Function to calculate turn around time  
def findTurnAroundTime(processes, n, bt,  
wt, tat):
```

```
    # Calculating turnaround  
    time    for i in range(n):  
        tat[i] = bt[i] + wt[i]
```

```
# Function to calculate average  
waiting # and turn-around times. def  
findavgTime(processes, n, bt,  
quantum):  
    wt = [0] * n  
    tat = [0] * n
```



```
# Function to find waiting
time
# of all processes
findWaitingTime(processes, n,
bt, wt, quantum)

# Function to find turn around
time
# for all processes
findTurnAroundTime(processes,n,b
t, wt, tat)

# Display processes along with all details
print("Processes   Burst Time   Waiting",
"Time   Turn-Around Time")   total_wt = 0
total_tat = 0   for i in range(n):
    total_wt = total_wt + wt[i]
    total_tat = total_tat + tat[i]
    print(" ", i + 1, "\t\t", bt[i],
"\t\t", wt[i], "\t\t", tat[i])

    print("\nAverage waiting time = %.5f"%(total_wt /n)
)   print("Average turn around time = %.5f"%(total_tat
/ n))
# Driver code
if __name__
=="__main__":
    # Process id's
    proc = [1, 2, 3]
    n = 3
```

```
# Burst time of all processes
burst_time = [10, 5, 8]

# Time quantum
quantum = 2;

findavgTime(proc, n, burst_time, quantum)
```

Output:

```
(aakash@kali)-[~]
└─$ python3 rr.py
Processes    Burst Time    Waiting Time    Turn-Around Time
1            10            13              23
2             5            10              15
3             8            13              21

Average waiting time = 12.00000
Average turn around time = 19.66667

(aakash@kali)-[~]
└─$
```

Experiment-12

AIM:- Implementation of Banker's Algorithm.

Code:

```
def main():

processes = int(input("number of processes : "))

resources = int(input("number of resources : "))

max_resources = [int(i) for i in input("maximum resources : ").split()]

    print("\n-- allocated resources for each process --")

currently_allocated = [[int(i) for i in input(f"process {j + 1} : ").split()]]

for j in range(processes)]

    print("\n-- maximum resources for each process --")

max_need = [[int(i)

for i in input(f"process {j + 1} : ").split()]]

for j in range(processes)]

    allocated = [0] * resources

for i in range(processes):

for j in range(resources):

allocated[j] += currently_allocated[i][j]

print(f"\ntotal allocated resources : {allocated}")

    available = [max_resources[i] - allocated[i]

for i in range(resources)]

print(f"total available resources : {available}\n")
```

```
running = [True] * processes

count = processes

while count != 0:

    safe = False

    for i in range(processes):

        if running[i]:

            executing = True

            for j in range(resources):

                if max_need[i][j] - currently_allocated[i][j] > available[j]:

                    executing = False

                    break

            if executing:

                print(f"process {i + 1} is executing")

            running[i] = False

            count -= 1

            safe = True

            for j in range(resources):

                available[j] += currently_allocated[i][j]

            break

    if not safe:

        print("the processes are in an unsafe state.")

        break
```

```
print(f"the process is in a safe state.\navailable resources : {available}\n")
```

```
if __name__ == '__main__': main()
```

Output:

```
(aakash@kali)~/Desktop
$ python3 bankers.py
number of processes : 5
number of resources : 4
maximum resources : 8 5 9 7

-- allocated resources for each process --
process 1 : 2 0 1 1
process 2 : 0 1 2 1
process 3 : 4 0 0 3
process 4 : 0 2 1 0
process 5 : 1 0 3 0

-- maximum resources for each process --
process 1 : 3 2 1 4
process 2 : 0 2 5 2
process 3 : 5 1 0 5
process 4 : 1 5 3 0
process 5 : 3 0 3 3

total allocated resources : [7, 3, 7, 5]
total available resources : [1, 2, 2, 2]

process 3 is executing
the process is in a safe state.
available resources : [5, 2, 2, 5]

process 1 is executing
the process is in a safe state.
available resources : [7, 2, 3, 6]

process 2 is executing
the process is in a safe state.
available resources : [7, 3, 5, 7]

process 4 is executing
the process is in a safe state.
available resources : [7, 5, 6, 7]

process 5 is executing
the process is in a safe state.
available resources : [8, 5, 9, 7]
```

```
(aakash@kali)~/Desktop
$
```

example outputs

```
number of processes : 5
number of resources : 4
maximum resources : 8 5 9 7

-- allocated resources for each process --
process 1 : 2 0 1 1
process 2 : 0 1 2 1
process 3 : 4 0 0 3
process 4 : 0 2 1 0
process 5 : 1 0 3 0

-- maximum resources for each process --
process 1 : 3 2 1 4
process 2 : 0 2 5 2
process 3 : 5 1 0 5
process 4 : 1 5 3 0
process 5 : 3 0 3 3

total allocated resources : [7, 3, 7, 5]
total available resources : [1, 2, 2, 2]

process 3 is executing
the process is in a safe state.
available resources : [5, 2, 2, 5]

process 1 is executing
the process is in a safe state.
available resources : [7, 2, 3, 6]

process 2 is executing
the process is in a safe state.
available resources : [7, 3, 5, 7]

process 4 is executing
the process is in a safe state.
available resources : [7, 5, 6, 7]

process 5 is executing
the process is in a safe state.
available resources : [8, 5, 9, 7]
```