PRINCIPLES OF OPERATING SYSTEM

18CS4SP05L

PRACTICAL RECORD

SUBMITTED BY

Academic Year: February – June 2022

Department of Computer Science and Engineering

School of Engineering and Technology

JAIN (Deemed-to-be University)

Jain Global Campus, Jakkasandra post, Kanakapura Taluk,

Ramanagara District – 562112

Jain Global Campus, Jakkasandra Post, Kanakapura Taluk,
Ramanagara District - 562112

Laboratory Certificate

This is to certify that Mr. /Ms. .....

..... has satisfactorily completed the course of experiments in practical PRINCIPLES OF OPERATING SYSTEMS LABORATORY prescribed by the Jain University 4TH Semester Course in the Laboratory of this college in year 20 - 20 .

Date:

Name of the Candidate: .....

Reg. No .....

Date of Practical Examination .....

REMARKS

VALUED
Examiner 1.....
Examiner 2 .....

Signature of the Teacher
In charge of the Batch

Head of Department

EXPERIMENT NO - 1	BASIC COMMANDS OF UNIX AND LINUX
DATE:	

Objective:

To learn the basic UNIX commands.

- To be able to work with VI editor.
- To learn UNIX shell programming at an introductory level.

Descriptions:

UNIX command prompt: A command prompt, also referred to simply as a prompt, is a short text message at the start of the command line on a command line interface. The command line is the line on which commands are typed in a console or terminal window. A command is an instruction to tell a computer to do something, e.g., to execute a program. The functions of a command prompt are:

i) to inform the user that the system is ready for the next command, data element or other input.
ii) to help the user plan and execute subsequent operations.

The dollar sign prompt (or a prompt ending with a dollar sign) means that UNIX is now ready to interpret and execute your commands as typed in from your keyboard.

The VI editor:

The default editor that comes with the UNIX operating system is called vi (visual editor). vi is a screen editor where a portion of the file is displayed on the terminal screen, and the cursor can be moved around the screen to indicate where you want to make changes. You can select which part of the file you want to have displayed. Screen editors are also called display editors, or visual editors. vi is one of the more popular screen editors that run on the UNIX system.

UNIX shell:

The shell acts as an interface between the user and the kernel. When a user logs in, the login program checks the username and password and then starts another program called the shell. The shell is a command line interpreter (CLI). It interprets the commands

that user types in and arranges for them to be carried out. The commands are themselves programs: when they terminate, the shell gives the user another prompt.

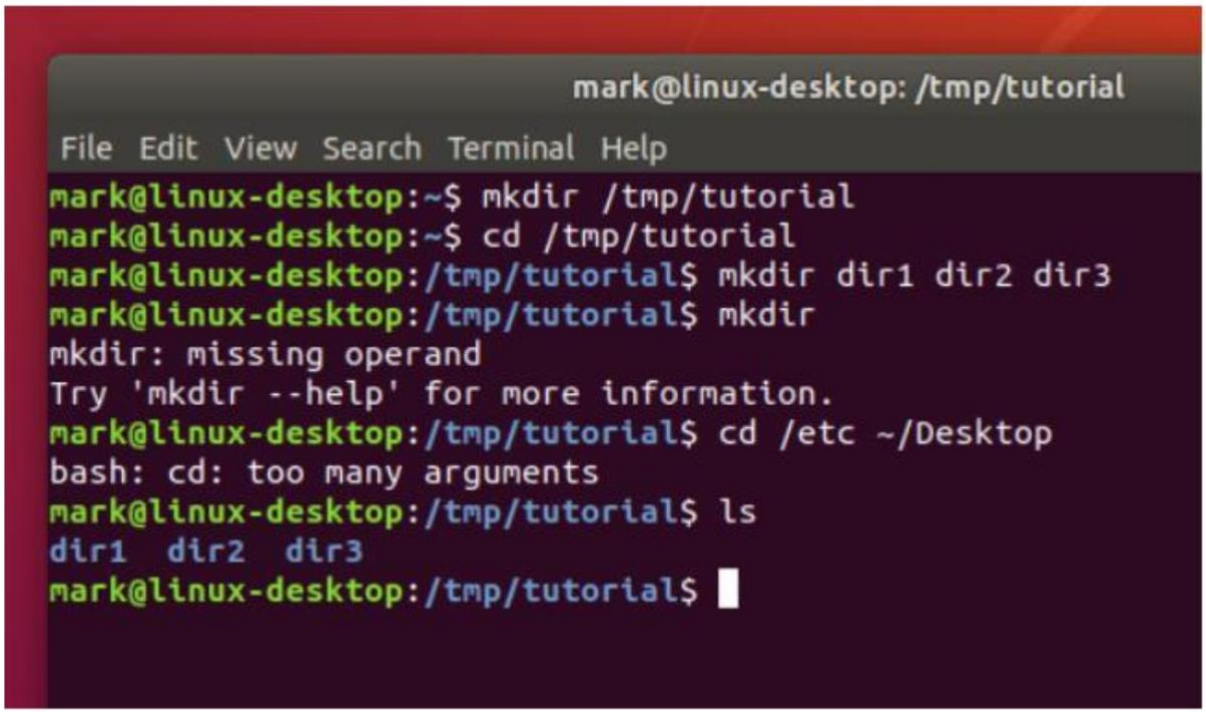
List of Programs:

Program 1: Execute 15 basic commands of UNIX.

The following table lists some of the basic UNIX commands. To execute the commands, open the command prompt and type those as they are and press 'Enter' button.

SL NO	COMMAND	EXAMPLE	DESCRIPTION
1	ls	ls ls -alF	Lists files in the current directoryList in long format
2	cd	cd tempdir cd – cd absolute path more	Change directory to tempdir Move back one directory
3	mkdir	mkdirgraphics	Make a directory called graphics
4	rmdir	rmdirgraphics	Remove directory (must be empty)
5	cp	cp file1web- docscp file1 file1.bak	Copy file into directoryMake backup of file1
6	rm	rm file1.bak rm *	Remove ordelete file1Remove all files
7	mv	mv old.html new.htmlmv file "newfile path" mv dir1 dir2	Move or rename files Moves the files to the new locationRenames dir1to dir2
8	more	more /var/log/auth.log cat /var/log/auth.log | more	Look at file, one page at a time
9	lpr	pr index.html	Send file to printer
10	man	man ls	Online manual (help) about command
11	grep<str><files>	grep "ABC" *	Find which files contain a certain word (e.g."ABC")
12	who	who	Lists who is logged on yourmachine
13	cat	cat filename cat > filename cat file1file2> file3	Displays the contents of the given file.Creates newfile. Joins two files (file1, file2) and stores the output in a new file (file3)

SAMPLE OUTPUT:



EXPERIMENT NO – 2	**BASIC FUNCTIONALLY AND MODES OF VI EDITOR**
DATE:	

Introduction: -

vi, command, and input modes: One of the most important aspects to remember about vi is that most of the commands fall into one of three modes:

1. **vi mode**: in this mode, most keys on the keyboard are defined to be a specific command. As the key or key sequence is issued, that command is executed. This is the mode vi starts in. At any time, pressing the key returns the user to vi mode.

2. **command mode:** to reach that mode, one must first be in vi mode, then issue a colon (":"). That same colon will appear at the bottom left corner of the screen. Then the command may be issued following the colon. One exception to this rule is the search command; a forward slash is issued instead of the colon.

3**. input mode:** this is where most users expect an editor to start. This "mode" actually refers to commands issued from vi mode but that allows the user to start inputting data into the file.

Objective: -

To learn Basic functionality of VI editor.

To learn Basic modes of VI editor.

Key Terminologies: -

Invoking vi: -

Type in the command prompt: vi filename

which will put filename into a buffer, and display the file on the screen. If the file is larger than the screen can display, the screen will act as a window into the file. At the beginning of a session, the screen will display the first part of the file. If filename does not exist, vi will create it. Upon entry to vi, the bottom of the screen will print the name of the file being edited, the number of lines in the file, and the size of the file (in characters).

Write two paragraphs (whatever you want) in the file (for testing the various commands).

Exiting vi: -

Usually, the new or modified file is saved when you leave vi. However, it is also possible to quit vi without saving the file.

Note: The cursor moves to the bottom of the screen whenever a colon (:) is typed. This type of command is completed by hitting the <Return>(or<Enter>)key

: x quit vi, writing out the modified file to file named in the original invocation

: wq quit vi, writing out the modified file to file named in the original invocation

: q! quit vi even though latest changes have not been saved for this vi call.

Moving the Cursor:

Unlike many of the PC and Macintosh editors, the mouse does not move the cursor within the vi editor screen (older versions). You must use the key commands listed below.

On some UNIX platforms, the arrow keys may be used as well; however, since vi was designed with the Qwerty keyboard (containing no arrow keys) in mind, the arrow keys sometimes produce strange effects in viand should be avoided.

If you go back and forth between a PC environment and a UNIX environment, you may find that this dissimilarity in methods for cursor movement is the most frustrating difference between the two. In the table below, the symbol ^ before a letter means that the < Ctrl & gt; key should be held down while the letter key is pressed.

Note: Since following are the commands they will not work in the INSERT mode. Just open the file by writing.

vi filename

on the command prompt and execute the commands without pressing 'I'. Before that, make sure that something is written in the file (refer invoking vi).

SAMPLE CODE:

j or <Enter>

[or down-arrow]

move cursor down one line

k [or up-arrow] move cursor up one line

h or [or left-arrow]

l or [or right-arrow]

move cursor left one character

move cursor right one character

0 (zero) move cursor to start of the current line (the one with the cursor)

$ move cursor to the end of the current line

w move cursor to the beginning of next word

b moves the cursor back to the beginning of preceding word

:0<Enter>or 1G move the cursor to the first line in the file

: $< Enter>or G move the cursor to the last line in the file

Screen Manipulation:

The following commands allow the vi-editor screen (or window) to move up or down several lines and to be refreshed.

Note: Since following are the commands, they will not work in the INSERT mode. Just open the file by writing vi filename on the command prompt and execute the commands without pressing 'I'. Before that make sure, that something is written in the file (refer invoking vi).

^f move forward one screen

^b move backward one screen

Adding and Deleting Text:

Unlike PC editors, you cannot replace or delete text by highlighting it with the mouse. Instead, use the commands in the following tables.

Perhaps the most important command is the one that allows you to back up and undo your last action. Unfortunately, this command acts like a toggle, undoing and redoing your most recent action. You cannot go back more than one step.

u undo whatever you just did; a simple toggle.

The main purpose of an editor is to create, add, or modify text for a file .

Inserting or Adding Text:

The following commands allow you to insert and add text. Each of these commands puts the vi editor into insert mode; thus, the key must be pressed to terminate the entry of text and to put the vi editor back into command mode.

Note 1: Since following are the commands, they will not work in the INSERT mode. Just open the file by writing

vi filename

on the command prompt and execute the commands without pressing 'i'. Before that make sure that something is written in the file (refer invoking vi).

Note 2: Each of these commands puts the vi editor into insert mode; thus, the key must be pressed to terminate the entry of text and to put the vi editor back into command mode.

I insert text before the cursor, until <Esc>hit

I insert text at beginning of current line, until <Esc>hit

A append text after the cursor, until <Esc>hit

A append text to the end of current line, until <Esc>hit

Deleting Text:

The following commands allow you to delete text.

X delete single character under the cursor

Nx delete N characters, starting with a character under the cursor

Dw delete the single word beginning with a character under the cursor

dNw delete N words beginning with a character under cursor; e.g., d5w deletes 5 words
D delete the remainder of the line, starting with the current cursor position

Dd delete entire current line

Ndd or dNd delete N lines, beginning with the current line; e.g., 5dd deletes 5 lines

Cutting and Pasting Text:

The following commands allow you to copy and paste text.

Yy copy (yank, cut) the current line into the buffer

Nyy or yNy copy (yank, cut) the next N lines, including the current line, into the buffer

P put (paste) the line(s) in the buffer into the text after the current line

Searching Text:

A common occurrence in text editing is to replace one word or phrase by another. To locate instances of particular sets of characters (or strings), use the following commands.

/String search forward for the occurrence of a string in the text

? string search backward for the occurrence of a string in the text

n move to next occurrence of the search string

N move to next occurrence of the search string in opposite direction

The rest of the experiments in the list involve shell programming.

PROGRAM 2: Follow the following steps in each case to execute the programs:

1. To write the programs, create new files for each program by writing in the command prompt: vi filename

2. Write the program as plain text (in Insert mode)

3. Save the file and exit

4. Run the file by giving following command in the command prompt: sh filename

EXPERIMENT NO – 3	TO ACCEPT OR REPORTS IF THE USER IS LOGGED IN
DATE:	

Objective: -

program that accepts username and reports if the user is logged in.

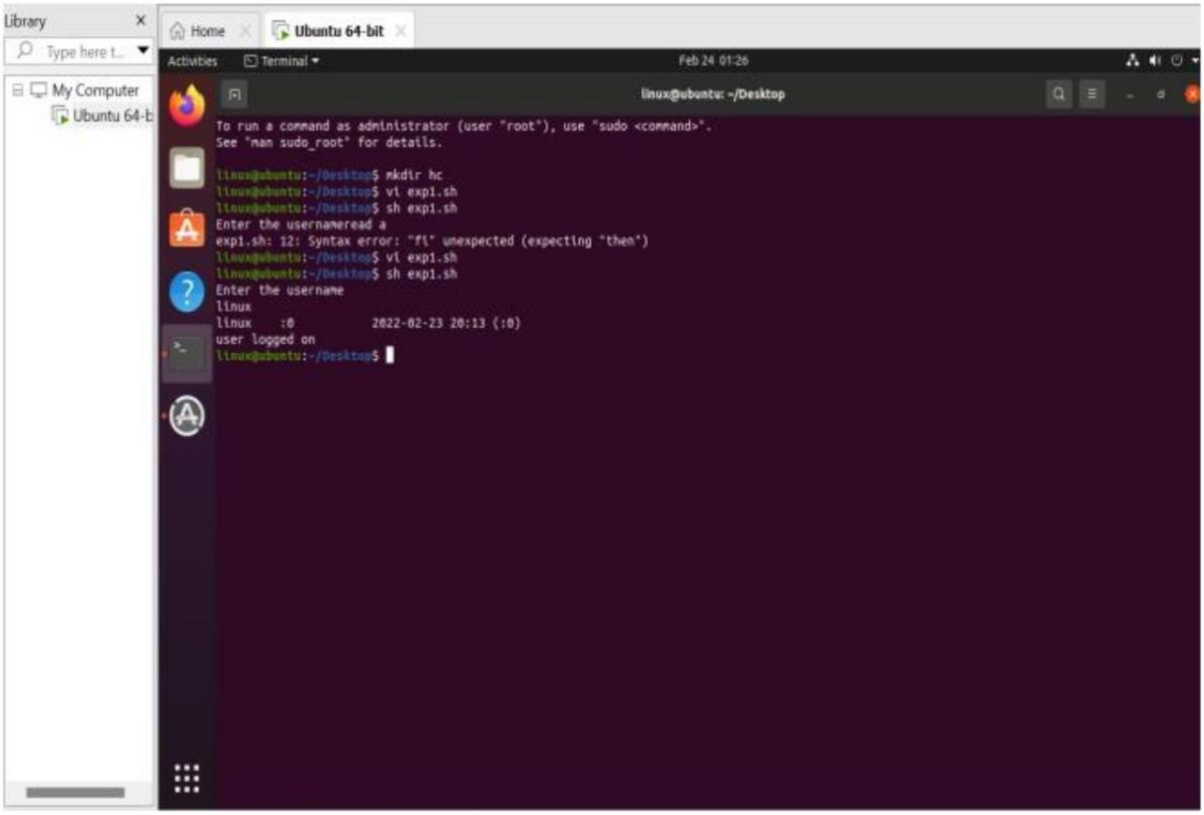
Descriptions: -

This program tells us about whether we have logged in or not logged . If we have logged with correct windows it checks and will say that you have logged in.

Sample Code: -

```
echo “Enter the username” read a

who>userlist

if grep $a userlist then

echo “user logged on” else

echo “user not logged on”

fi
```

Sample Output: -


Library
Type here t...
My Computer
Ubuntu 64-b
Home
Ubuntu 64-bit
Activities
Terminal
Feb 24 01:26
linux@ubuntu: ~/Desktop
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
linux@ubuntu:~/Desktop$ mkdir hc
linux@ubuntu:~/Desktop$ vi exp1.sh
linux@ubuntu:~/Desktop$ sh exp1.sh
Enter the usernameread a
exp1.sh: 12: Syntax error: "fi" unexpected (expecting "then")
linux@ubuntu:~/Desktop$ vi exp1.sh
linux@ubuntu:~/Desktop$ sh exp1.sh
Enter the username
linux
linux :0 2022-02-23 20:13 (:0)
user logged on
linux@ubuntu:~/Desktop$

EXPERIMENT NO – 4	PROGRAMS DISPLAYS FOLLOWING MENU AND EXECITES THE OPTION SELECTED BY THE USER
DATE:	

Objective: -

Program that displays the following menu and executes the option selected by the user:

Descriptions: -

i) Ls
ii) pwd
iii) who
iv) ls -l
v) ps -fe

1) ls;; # lists directory content

2) pwd;; # prints name of the current directory

3) who;; # shows who is logged on

4) ls -l;; # shows directory content listing format

5) ps – fe;; # The -e option generates a list of information about every process currently running. The -f option generates a listing that contains fewer items of information for each process than the -l option.

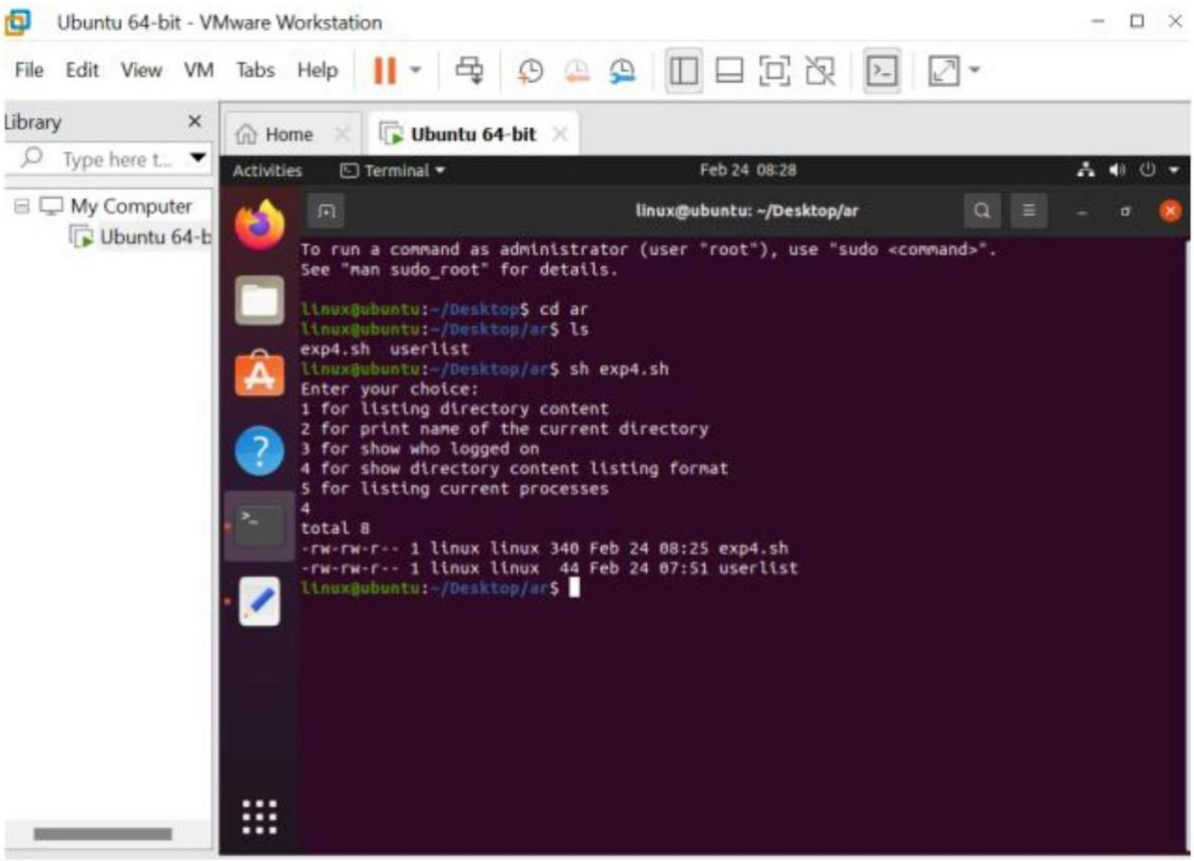
Sample Code: -

```
echo “Enter yourchoice:”
echo “1 for listing directory content”
echo “2 for print name of the current directory”
echo “3 for show who is logged on”
echo “4 for show directory content listing format”
echo “5 for listing current processes”
read ch
case $ch in *)
```

echo “Invalid choice. Try again.”

Esac

Sample Output: -

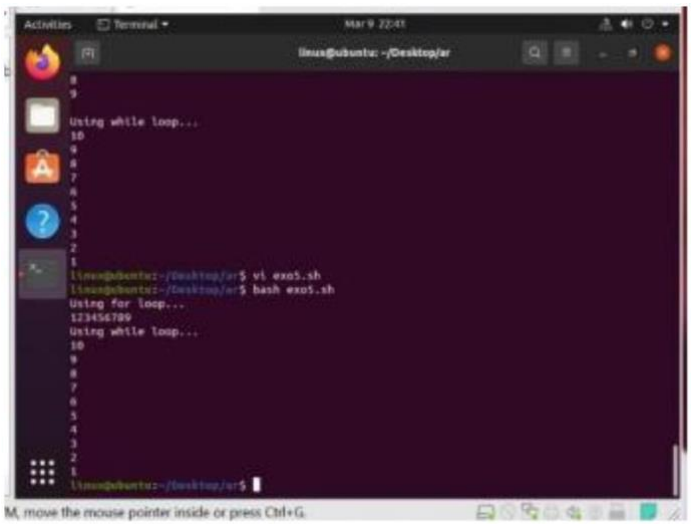


EXPERIMENT NO – 5	PROGRAM TO PRINT NUMBERS FROM 1 TO 10
DATE:	

CODE: -

```
echo “Using for loop... “
for (( i=10; i>0; i-- ))
do echo -n “$i “ done
echo “”
echo “Using while loop...”
j=10
while [ $j -ge 1 ]
do
echo -n “$j “
j=$(( j - 1 )) # decrease number by 1 done
echo ‘’’
done
```

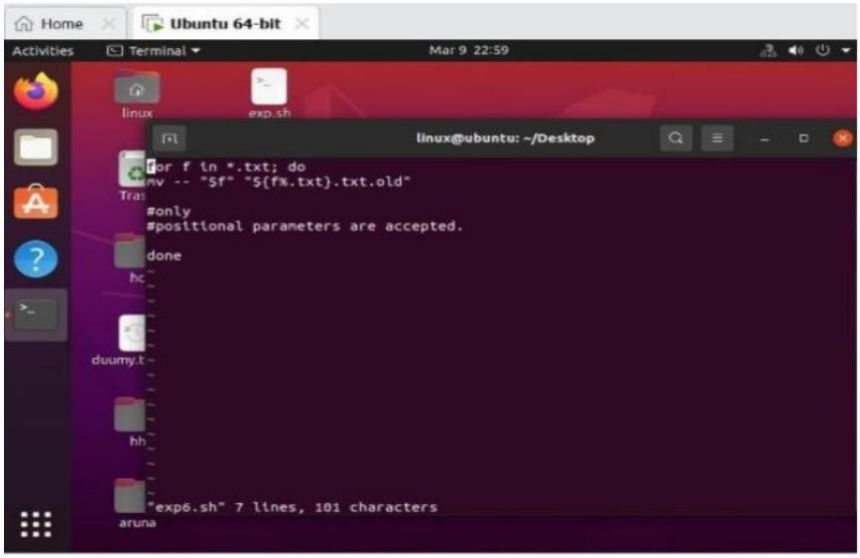
SAMPLE OUTPUT:



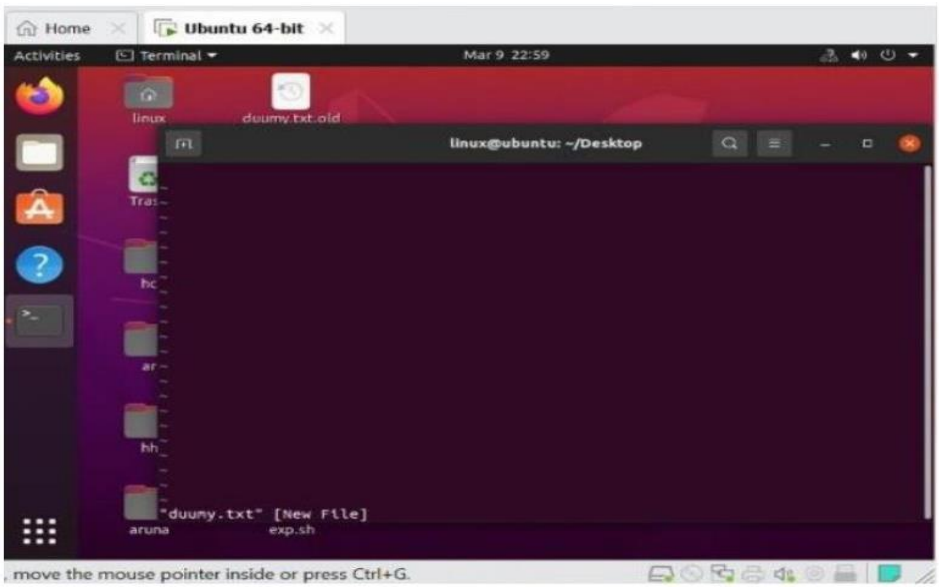
EXPERIMENT NO – 6	PROGRAM TO PRINT THAT REPLACE ALL “*.txt” FILE NAMES WITH “*.txt.old” IN THE CURRENT WORKING DIRECTORY
DATE:	

CODE: -

```
for f in *.txt ; do
mv -- "$f" "${f%.txt}.txt.old"#-- is used to signify the end of command options, afterwhich
only
positional parameters are accepted.
Done
```



SAMPLE OUTPUT:



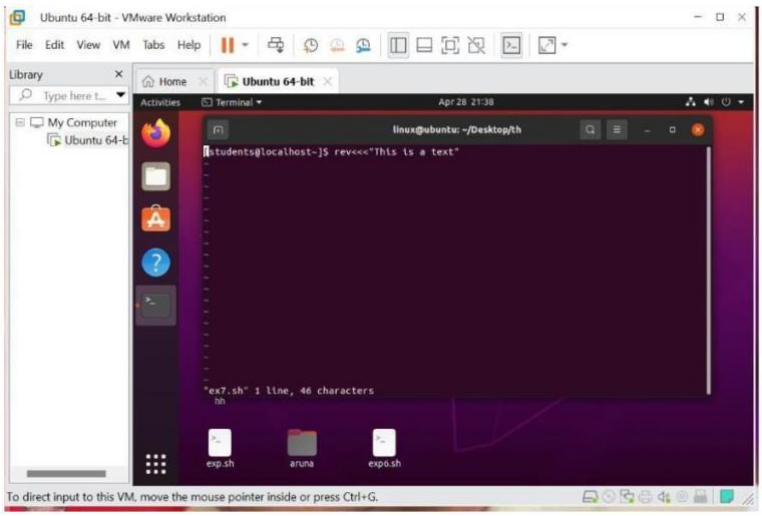
EXPERIMENT NO – 7	
DATE:	**PROGRAM THAT ECHOSE ITSELF TO STDOUT, BUT BACKWARD**

CODE: -

[students@localhost ~]$rev<<<"This is a test"

(or)

[students@localhost ~]$echo welcome | rev



SAMPLE OUTPUT:

EXPERIMENT NO – 8	PROGRAM THAT TAKES FILE NAME AS INPUTAND CHECKS IF FILE MODE EXECUTABLE, IF NOT. MAKE IT EXECUTABLE
DATE:	

CODE: -

echo “Enter yourfile name”

read a

if [! -e $a] then

echo “file not exist”

elif [-x $a]

then

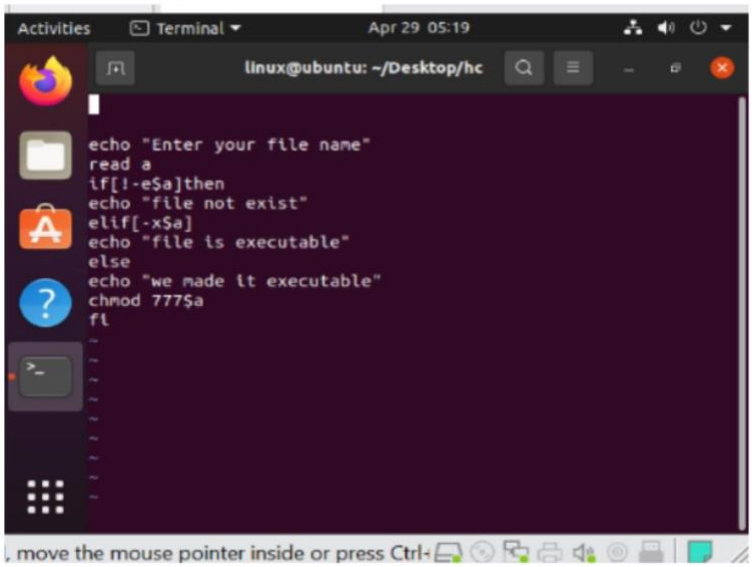
echo “file is executable”

else

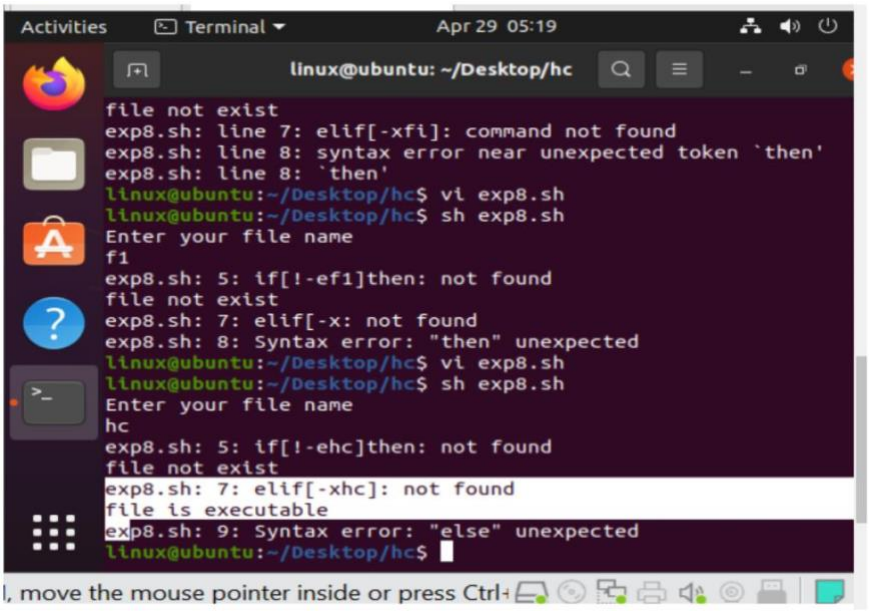
echo “we made it executable”

chmod 777 $a

fi



SAMPLE OUTPUT:



EXPERIMENT - 9	WAP TO TAKE A STRING AS COMMAND LINE ARGUMENT AND REVERSE IT
DATE:	

read -p " Enter Here : " text echo "You have entered : " $text echo -n "Reverse of String : " arr=($text)

arrlength=${#arr[@]} arrlength=`expr $arrlength - 1` while [$arrlength -ge 0]

do

echo -n ${arr[arrlength]} echo -n " "

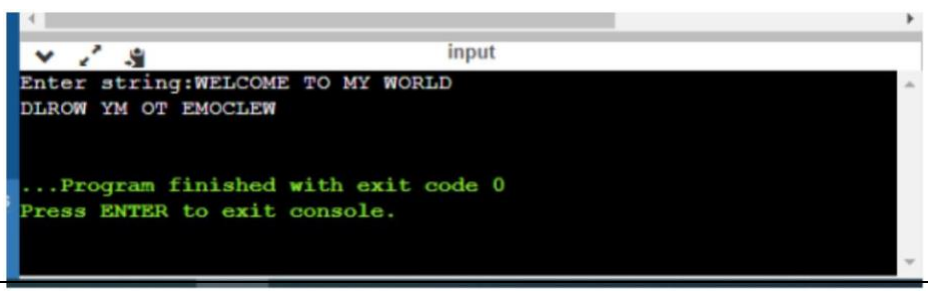
arrlength=`expr $arrlength - 1` done

echo

SAMPLE CODE:

```
#                              Online Bash Shell.
#                  Code, Compile, Run and Debug Bash script onl
# Write your code in this editor and press "Run" button to exe


#!/bin/bash
read -p "Enter string:" string
len=${#string}
for (( i=$len-1; i>=0; i-- ))
do
# "${string:$i:1}"extract single single character from string.
reverse="$reverse${string:$i:1}"
done
echo "$reverse"

```

SAMPLE OUTPUT:



Ex No: 10 Date:	CREATION OF A DATA FILE AND ITS OPERATIONS

Objective:

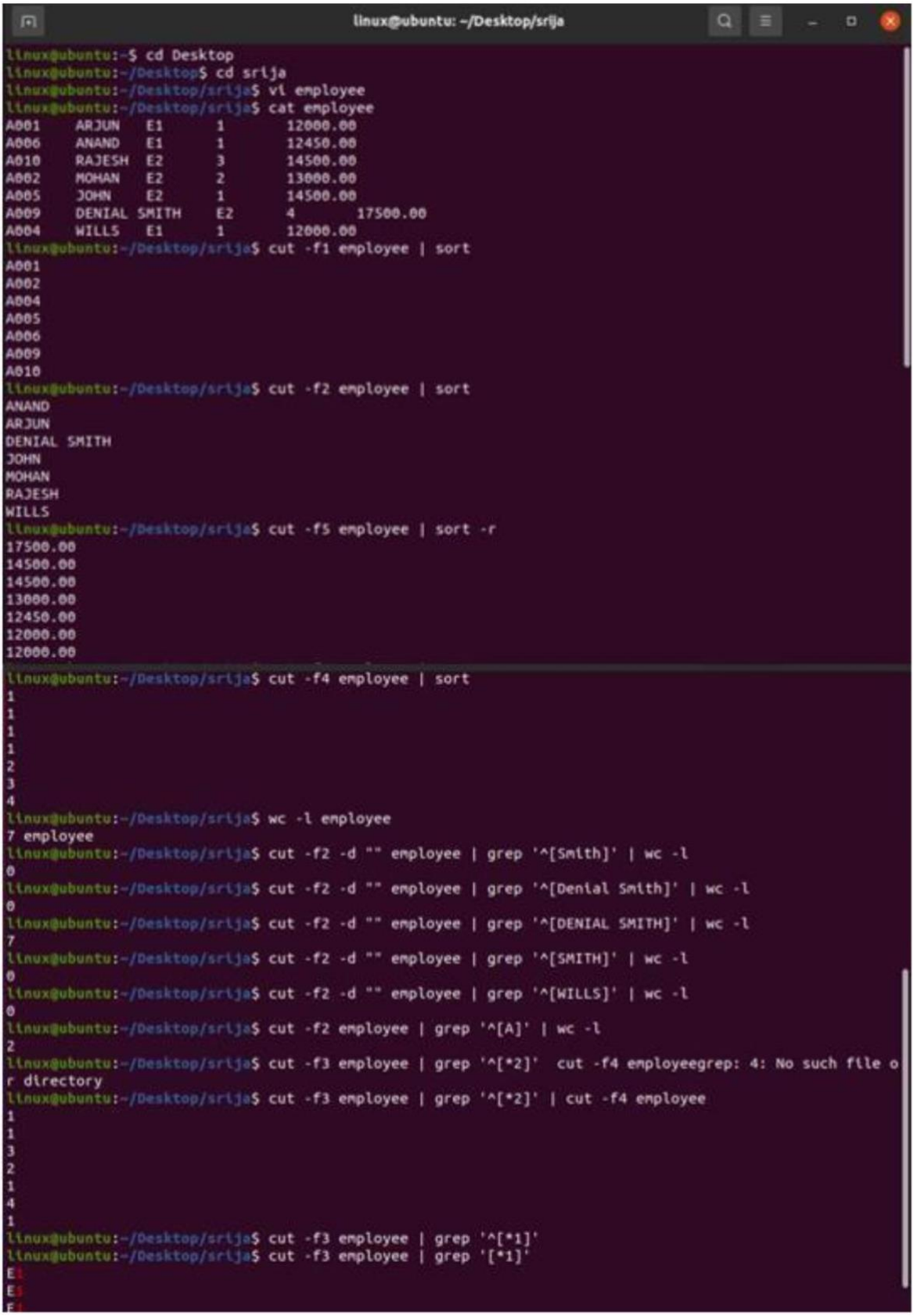
Creating a data file called employee and performing operations while sorting the file.

Code:

```
linux@ubuntu:~$ cd Desktop
linux@ubuntu:~/Desktop$ cd Akhil

linux@ubuntu:~/Desktop/Akhjil$ vi employee
linux@ubuntu:~/Desktop/Akhil$ cat
employeeA001          ARJUN E1     1
        12000.00
A006    ANAND         E1       1        12450.00
A010    RAJESH        E2       3        14500.00
A002    MOHAN         E2       2        13000.00
A005    JOHN   E2     1        14500.00
A009    DENIAL SMITHE2         4        17500.00
A004    WILLS  E1     1        12000.00
linux@ubuntu:~/Desktop/Akhil$  cut  -f1  employee  |
sort employee
A001
A002
A004
A005
A006
A009
A010
linux@ubuntu:~/Desktop/Akhil$ cut -f2 employee |
sortANAND
ARJUN
DENIAL SMITH
JOHN
MOHAN
RAJESH
WILLS
linux@ubuntu:~/Desktop/Akhil$ cut -f5 employee | sort
17500.00
14500.00
14500.00
13000.00
12450.00
12000.00
12000.00
```

```
linux@ubuntu:~/Desktop/Akhil$ cut -f4 employee |
sort1
1
1
1
2
3
4
linux@ubuntu:~/Desktop/Akhil$ wc -l
employee7 employee
linux@ubuntu:~/Desktop/Akhil$ cut -f2 -d "" employee | grep '^[Smith]' | wc
-l0
linux@ubuntu:~/Desktop/Akhil$ cut -f2 employee | grep '^[A]' | wc
-l2
linux@ubuntu:~/Desktop/Akhil$ cut -f3 employee | grep '^[*2]' | cut -f4
employee1
1
3
2
1
4
1
linux@ubuntu:~/Desktop/Akhil$ cut -f3 employee | grep
'^[*1]'linux@ubuntu:~/Desktop/Akhil$ cut -f3 employee |
grep '[*1]' E1
E1
E1
linux@ubuntu:~/Desktop/Akhil$ cd
..linux@ubuntu:~/Desktop $ cd ..
linux@ubuntu:~ $ clear
linux@ubuntu:~ $ exit
```

Sample Output:



Git-hub:

https://github.com/Akhil/Linux/blob/main/Data%20file%20creation

-

EXPERIMENT NO – 12	SIMULATION OFF FCFS, SJF AND RR SCHEDULER
DATE:	

FCFS CPU:

Given n processes with theirburst times, the task is to find average waiting time and average turnaround time using

FCFS scheduling algorithm.

First in, first out (FIFO), also known as first come, first served (FCFS), is the simplest scheduling algorithm. FIFO

simply queues processes in the orderthat they arrive in the ready queue.

In this, the process that comes first will be executed first and next process starts only afterthe previous gets fully

executed.

Completion Time: Time at which process completes its execution.

Turn Around Time: Time Difference between completion time and arrival time. Turn Around Time = Completion

Time – Arrival Time

Waiting Time(W.T): Time Difference between turn around time and burst time.

Waiting Time = Turn Around Time – Burst Time

Implementation:

1- Input the processes along with their burst time (bt).

2- Find waiting time (wt) for all processes.

3- As first process that comes need not to wait so

waiting time for process 1 will be 0 i.e. wt[0] = 0.

4- Find waiting time for all other processes i.e. for

process i ->

wt[i] = bt[i-1] + wt[i-1] .

5- Find turnaround time = waiting_time + burst_time

for all processes.

6- Find average waiting time = total_waiting_time / no_of_processes.

7- Similarly, find average turnaround time = total_turn_around_time / no_of_processes.

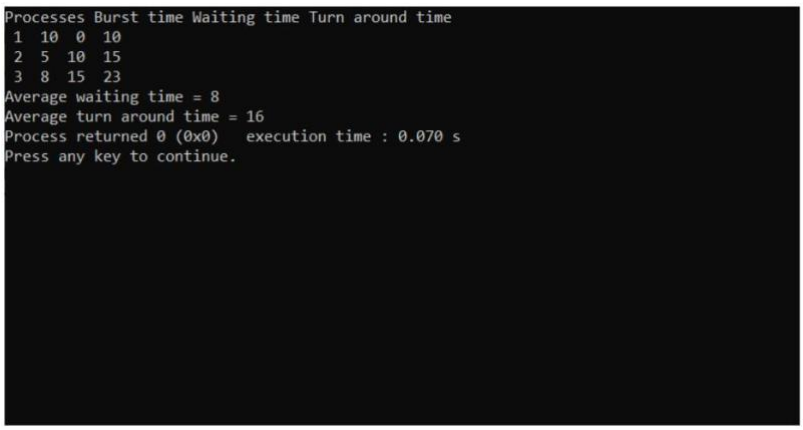
SAMPLE CODE:

```
// C program for implementation of FCFS
//scheduling
#include<stdio.h>
// Function to find the waiting time for all
// processes
void findWaitingTime(int processes[], int n,
int bt[], int wt[])
{
// waiting time for first process is 0
wt[0] = 0;
// calculating waiting time
for (int i = 1; i < n ; i++ )
wt[i] = bt[i-1] + wt[i-1] ;
}
// Function to calculate turn around time
void findTurnAroundTime( int processes[], int n,
int bt[], int wt[], int tat[])
{
// calculating turnaround time by adding
// bt[i] + wt[i]
```

```
for (int i = 0; i < n ; i++)
tat[i] = bt[i] + wt[i];
}
//Function to calculate average time
void findavgTime( int processes[], int n, int bt[])
{
int wt[n], tat[n], total_wt = 0, total_tat = 0;
//Function to find waiting time of all processes
findWaitingTime(processes, n, bt, wt);
//Function to find turn around time for all processes
findTurnAroundTime(processes, n, bt, wt, tat);
//Display processes along with all details
printf("Processes Burst time Waiting time Turn around time\n");
// Calculate total waiting time and total turn
// around time
for (int i=0; i < n; i++)
{
total_wt = total_wt + wt[i];
total_tat = total_tat + tat[i];
printf(" %d",(i+1));
printf("  %d", bt[i]);
printf("  %d",wt[i]);

printf("  %d\n",tat[i] );
}
int s=(float)total_wt / (float)n;
int t=(float)total_tat / (float)n;
printf("Average waiting time = %d",s);
printf("\n");
```

```
printf("Average turn around time = %d ",t);
}
// Driver code
int main()
{
//process id&#39;s
int processes[] = { 1, 2, 3};
int n = sizeof processes / sizeof processes[0];
//Burst time of all processes
int burst_time[] = {10, 5, 8};
findavgTime(processes, n, burst_time);
return 0;
}
```

SAMPLE OUTPUT:



EXPERIMENT NO – 13 DATE:	**USE FORK SYSTEM CALL**

```
#define _POSIX_SOURCE
#include <stdio.h>
#include <inttypes.h>
#include <unistd.h>
//#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <stdbool.h>
/* Size of the data blocks copied in bytes */
#define BLOCK_SIZE 1024
/* Declare function to copy from one file handler to an other */
int16_t copy(int src, int dest, int pid);
int main(int argc, char const *argv[]) {
/* Check number of arguments and print manual */
if(argc < 3) {
printf("ERROR: Too few arguments. Usage: ./PipeCopy src dest\n");
return 3;
}
/* Create ordinary pipe */
int fd[2];
```

```
if(pipe(fd)) {
fprintf (stderr, "ERROR: Pipe creation failed.\n");
return 1;
}
/* Fork process */
pid_t child_1 = fork();
pid_t child_2 = 0;
/* Code for parent after forking child 1 */
if(child_1 > 0) {
/* Fork process again */
child_2 = fork();
}
/* Code for parent after forking child 2 */
if(child_2 > 0) {
/* Close both ends of the pipe for parent */
close(fd[0]);
close(fd[1]);
/* Wait for both childs to finish */
bool error = false;
for(uint8_t i=0; i<2; i++) {
/* Wait for one child to terminate */
int status;
pid_t done = wait(&status);
/* Query the plain exit status */
status = WEXITSTATUS(status);
/* Check which child is done */
uint8_t child_number = 0;
if(child_1 == done) {
child_number = 1;
```

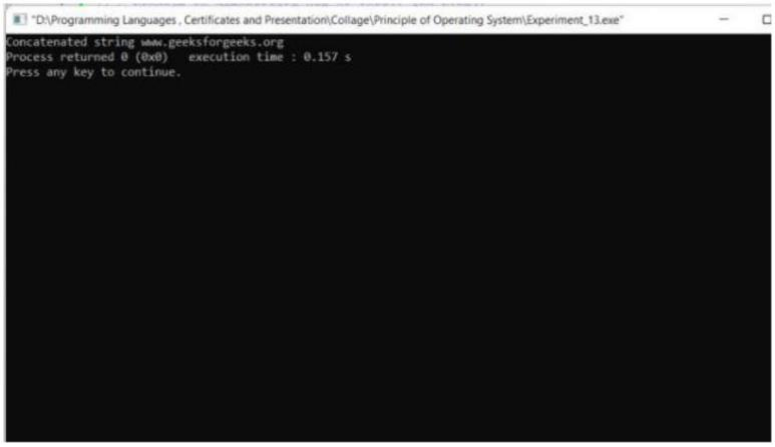
```
} else {
child_number = 2;
}
/* If the status is ok, print successull text */
if(status == 0) {

printf("SUCCESS: Child %d finished normally.\n", child_number);
}
/* If not, print error message and cancel other child */
else {
/* Print error */
printf("ERROR: Child %d finished abnormally with status %d\n", child_number,status);

/* Set error flag */
error = true;
}
}
/* Print message that all childs are terminated */
if(!error) {
printf("SUCCESS: All children are terminated normally.\n");
return 0;
} else {
printf("ERROR: One or more child finished abnormally. Operation failed.\n");
return 2;
}
}
/* Code for child 1 (read file) */
if(child_1 == 0) {
/* Close read end of pipe */
```

```
close(fd[0]);
/* Open file pointer for source and handle error */
int src = open(argv[1], O_RDONLY);
if(src < 0) {
printf("ERROR: Unable to open source file \"%s\" (%s)\n", argv[1], strerror(errno));
close(fd[1]);
return 1;
}
/* Copy from file to pipe */
if(copy(src, fd[1], 1) < 0) {
printf("ERROR: error while copying: %s\n", strerror(errno));
close(src);
close(fd[1]);
return 2;
}
/* Close file and pipe */
close(src);
close(fd[1]);
return 0;
}
/* Code for child 2 (write file) */
if(child_2 == 0) {
/* Close write end of pipe */
close(fd[1]);
/* Open file pointer for destination and handle error */
int dest = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR |
S_IRGRP | S_IWGRP | S_IROTH);
if(dest < 0) {
printf("ERROR: Unable to open destination file \"%s\" (%s)\n", argv[2], strerror(errno));
close(fd[0]);
```

```
return 1;
}
/* Copy from pipe to file */
if(copy(fd[0], dest, 2) < 0) {
printf("ERROR: error while copying: %s\n", strerror(errno));
close(dest);
close(fd[0]);
return 2;
}
/* Close file and pipe */
close(dest);
close(fd[0]);
return 0;
}
/* Code for error handling */
if(child_1 < 0 || child_2 < 0) {
printf("ERROR: Unable to for process!\n");
/* If child 1 is already forked, we just let it finish normally */
return 1;
}
return 0;
}
/* Copies the content from the src file handler to the dest file handler. */
int16_t copy(int src, int dest, int pid) {
/* Create buffer and counter */
uint8_t buffer[BLOCK_SIZE];
int16_t read_count = 0;
/* Copy blocks */
while((read_count = read(src, buffer, BLOCK_SIZE)) > 0) {
```

```
printf("[%d] %d bytes copied...\n", pid, read_count);
write(dest, buffer, read_count);
}
return read_count;
}
```

SAMPLE OUTPUT:



EXPERIMENT – 14	Inter process communications
DATE:	

Burger Buddies Problem: Design, implement and test a solution for the IPC problem specified below. Suppose we have the following scenario:

```
#define COOK_COUNT 3
#define CASHIER_COUNT 2
#define CUSTOMER_COUNT 4
#define RACK_HOLDER_SIZE 4
#define WAITING_TIME 5
#include &lt;stdio.h&gt;
#include &lt;stdlib.h&gt;

#include &lt;pthread.h&gt;
#include &lt;semaphore.h&gt;
#include &lt;inttypes.h&gt;
#include &lt;stdbool.h&gt;
#include &lt;unistd.h&gt;
#include &lt;time.h&gt;
/* Flag which will be set to true when the threads should terminate themselfs */
bool interrupt = false;
/* define struct representing a cashier */
typedef struct {
uint8_t id;
sem_t *order;
sem_t *burger;
} cashier_t;
/* define struct to pass args to the run functions */
typedef struct {
uint8_t id;
sem_t *init_done;
} simple_arg_t;
/* Declare funcitons for the cook, cashiers and customers to run */
void *cook_run();
void *cashier_run();
void *customer_run();
/* Declare function to check current system state */
void assure_state();
/* Define all needed semaphores */
sem_t rack;
sem_t cook;
sem_t cashier;
sem_t cashier_awake;
sem_t customer;
sem_t customer_private_mutex;
```

```
/* Define memory space to store a cashier */
cashier_t cashier_exchange;
/* Define the counter for available burgers in the rack */
uint8_t burger_count = 0;
int main(int argc, char **argv) {
/* Init random number generator */
srand(time(NULL));

/* Init all semaphores */
sem_init(&amp;amp;rack, 0, 1);
sem_init(&amp;amp;cashier, 0, 1);
sem_init(&amp;amp;cashier_awake, 0, 0);
sem_init(&amp;amp;cook, 0, RACK_HOLDER_SIZE);
sem_init(&amp;amp;customer, 0, 0);
sem_init(&amp;amp;customer_private_mutex, 0, 1);
/* Create semaphore to synchronise thread init and args */
simple_arg_t args;
sem_t init_done;
sem_init(&amp;amp;init_done, 0, 0);
args.init_done = &amp;amp;init_done;
/* Start all cook threads */
pthread_t cooks[COOK_COUNT];
for(uint8_t i=0; i&amp;lt;COOK_COUNT; i++) {
/* Set id for cook */
args.id = i;
/* Start cook thread and pass args, handle possible erros */
if(pthread_create(cooks+i, NULL, cook_run, (void*) &amp;amp;args)) {
printf(&amp;quot;[MAIN]\t\t ERROR: Unable to create cook thread.\n&amp;quot;);
exit(1);
}
/* Wait until the cook is initialised and ready to run */
sem_wait(&amp;amp;init_done);
}
/* Start all cashier threads */
pthread_t cashiers[CASHIER_COUNT];
for(uint8_t i=0; i&amp;lt;CASHIER_COUNT; i++) {
/* Set id for cashier */
args.id = i;
/* Start cashier thread and pass args, handle possible erros */
if(pthread_create(cashiers+i, NULL, cashier_run, (void*) &amp;amp;args)) {
printf(&amp;quot;[MAIN]\t\t ERROR: Unable to create cashier thread.\n&amp;quot;);
exit(2);
}
/* Wait until the cashier is initialised and ready to run */
sem_wait(&amp;amp;init_done);
}
/* Start all customer threads */
```



```
pthread_t customers[CUSTOMER_COUNT];
for(uint8_t i=0; i&amp;lt;CUSTOMER_COUNT; i++) {
/* Set id for customer */
args.id = i;
/* Start customer thread and pass args, handle possible erros */

if(pthread_create(customers+i, NULL, customer_run, (void*) &amp;amp;args)) {
```

```
printf(&quot;[MAIN]\t\t ERROR: Unable to create customer thread.\n&quot;);
exit(3);
}
/* Wait until the customer is initialised and ready to run */
sem_wait(&amp;init_done);
}
/* destroy init semaphore */
sem_destroy(&amp;init_done);
/* wait for all customer threads to finish */
for(uint8_t i=0; i&lt;CUSTOMER_COUNT; i++) {
/* Join customer and handle possible errors */
if(pthread_join(customers[i], NULL)) {
printf(&quot;[MAIN]\t\t ERROR: Unable to join cutomers[%d]\n&quot;, i);
exit(4);
}
}
/*
* CLEANUP
*/
printf(&quot;[MAIN]\t\t SUCCESS: All customers terminated\n&quot;);
printf(&quot;\n----------\n\n[MAIN]\t\t SUCCESS: Starting Cleanup\n&quot;);
/* Set interrupt flag */
interrupt = true;
/* Wake all cooks up, they will see the interrupt flag and will exit */
for(uint8_t i=0; i&lt;COOK_COUNT; i++) {
sem_post(&amp;cook);
}
/* Wake all cashiers up, they will see the interrupt flag and will exit */
for(uint8_t i=0; i&lt;CASHIER_COUNT; i++) {
sem_post(&amp;customer);
}
/* All threads were woken up */

printf(&quot;[MAIN]\t\t SUCCESS: Told all threads to terminate themselves\n&quot;);
/* wait for all cook threads to finish */
for(uint8_t i=0; i&lt;COOK_COUNT; i++) {
/* Join customer and handle possible errors */
if(pthread_join(cooks[i], NULL)) {
printf(&quot;[MAIN]\t\t ERROR: Unable to join cooks[%d]\n&quot;, i);
exit(5);
}
}
/* wait for all cashier threads to finish */
for(uint8_t i=0; i&lt;CASHIER_COUNT; i++) {
/* Join customer and handle possible errors */
if(pthread_join(cashiers[i], NULL)) {
printf(&quot;[MAIN]\t\t ERROR: Unable to join cashiers[%d]\n&quot;, i);
exit(6);
}
}
/* Print status and exit */
assure_state();
printf(&quot;[MAIN]\t\t SUCCESS: All threads terminated, state consistent.\n&quot;);

}
void *cook_run(void *args) {
```

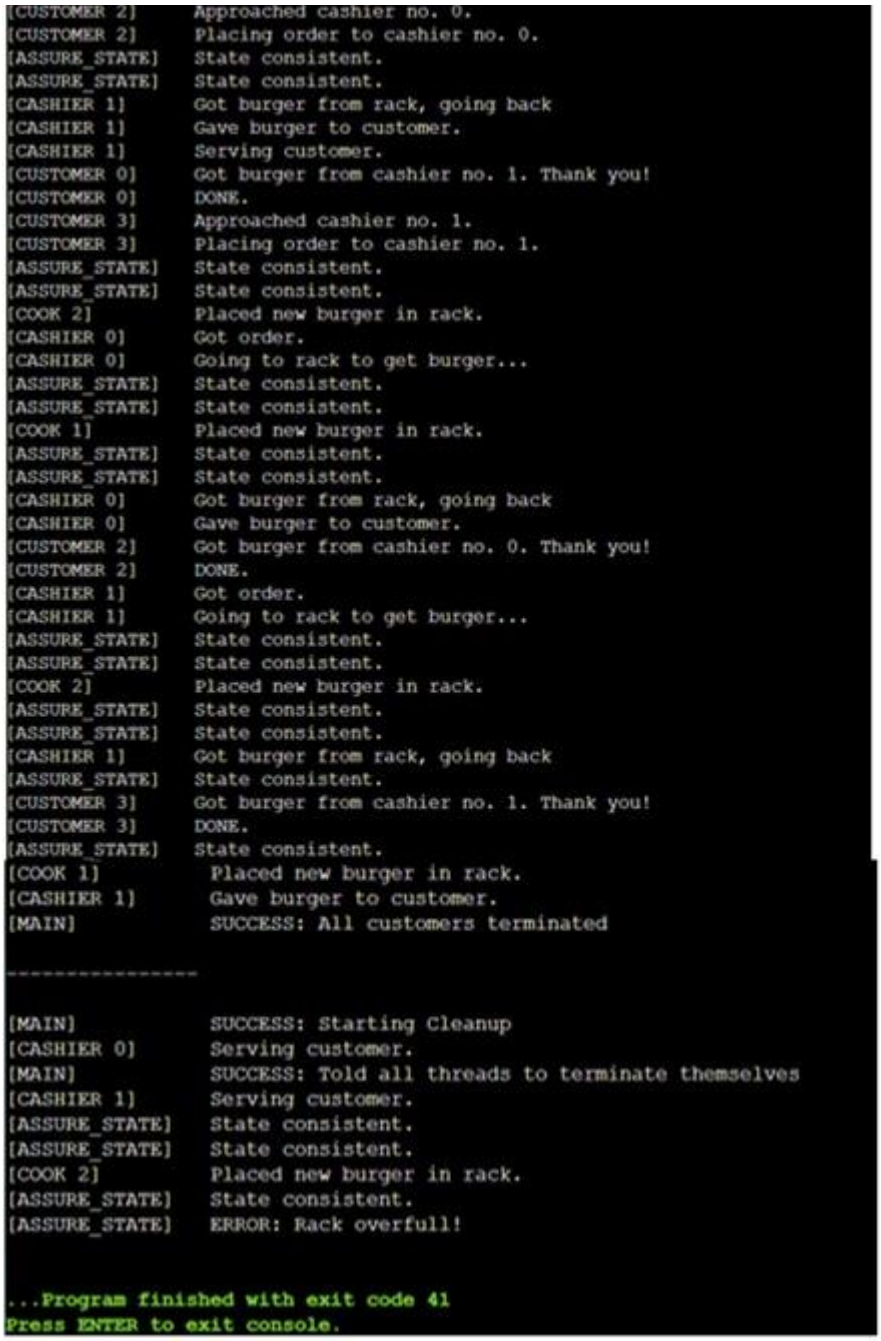
```
/* Get args from void pointer */
simple_arg_t *args_ptr = (simple_arg_t*) args;
/* Get id */
uint8_t cook_id = args_ptr-&amp;gt;id;
/* Print status and signal the init_done semaphore */
printf(&amp;quot;[COOK %d]\t CREATED.\n&amp;quot;, cook_id);
sem_post(args_ptr-&amp;gt;init_done);
/* Infinite loop */
while(1) {
/* Aquire cook semaphore */
sem_wait(&amp;amp;cook);
/* Check if we should terminate */
if(interrupt) {
break;
}
/* Cook */
sleep(rand() % WAITING_TIME);


/* Lock rack and produce burger */
sem_wait(&amp;amp;rack);
assure_state();
burger_count++;
assure_state();
sem_post(&amp;amp;rack);
printf(&amp;quot;[COOK %d]\t Placed new burger in rack.\n&amp;quot;, cook_id);
/* Signal a waiting cashier */
sem_post(&amp;amp;cashier);
}
printf(&amp;quot;[COOK %d]\t DONE.\n&amp;quot;, cook_id);
return NULL;
}
void *cashier_run(void *args) {
/* Get args from void pointer */
simple_arg_t *args_ptr = (simple_arg_t*) args;
/* Get id */
uint8_t cashier_id = args_ptr-&amp;gt;id;
/* Create private order and burger semaphore */
sem_t order;
sem_t burger;
sem_init(&amp;amp;order, 0, 0);
sem_init(&amp;amp;burger, 0, 0);
/* Print status and signal the init_done semaphore */
printf(&amp;quot;[CASHIER %d]\t CREATED.\n&amp;quot;, cashier_id);
sem_post(args_ptr-&amp;gt;init_done);
/* Infinite loop */
while(1) {
/* Wait for customer */
sem_wait(&amp;amp;customer);
/* Check if we should terminate */
if(interrupt) {
break;

}
/* print status */
printf(&amp;quot;[CASHIER %d]\t Serving customer.\n&amp;quot;, cashier_id);
```

```
/* Save my id to the exchange */
cashier_exchange.order = &amp;order;
cashier_exchange.burger = &amp;burger;
cashier_exchange.id = cashier_id;
/* Tell customer that I am awake and information is placed */
sem_post(&amp;cashier_awake);
/* Wait for the order and print sttaus */
sem_wait(&amp;order);
printf(&quot;[CASHIER %d]\t Got order.\n&quot;, cashier_id);
/* Print sttaus that now the burger will be get from the rack */
printf(&quot;[CASHIER %d]\t Going to rack to get burger...\n&quot;, cashier_id);
/* Go to rack */
sleep(rand() % WAITING_TIME);
/* Aquire cashier semaphore */
sem_wait(&amp;cashier);
/* Lock rack and get burger */
sem_wait(&amp;rack);
assure_state();
burger_count--;
assure_state();
sem_post(&amp;rack);
/* Signal a waiting cook a new burger can be produced */
sem_post(&amp;cook);
/* Got successfull a burger. Print status */
printf(&quot;[CASHIER %d]\t Got burger from rack, going back\n&quot;, cashier_id);
/* Go back to customer */
sleep(rand() % WAITING_TIME);
/* Give burger to customer and print status */
sem_post(&amp;burger);
printf(&quot;[CASHIER %d]\t Gave burger to customer.\n&quot;, cashier_id);
}
/* free semaphores and print message */
sem_destroy(&amp;order);
sem_destroy(&amp;burger);
printf(&quot;[CASHIER %d]\t DONE.\n&quot;, cashier_id);
return NULL;

}
void *customer_run(void *args) {
/* Get args from void pointer */
simple_arg_t *args_ptr = (simple_arg_t*) args;
/* Get id */
uint8_t customer_id = args_ptr-&gt;id;
/* Print status and signal the init_done semaphore */
printf(&quot;[CUSTOMER %d]\t CREATED.\n&quot;, customer_id);
sem_post(args_ptr-&gt;init_done);
/* Wait random time to mix customers (wait atleast 1s to assure that all
customers are already created) */


sleep(rand() % WAITING_TIME + 1);
/* Synchronize all customers to get a cashier. This will queue up the
customers and guarantees every customer to get a cashier without race
conditions */
```

```
sem_wait(&amp;amp;customer_private_mutex);
/* Signal cashier that a customer is in the room and wait for a cashier to
wake up */
sem_post(&amp;amp;customer);
sem_wait(&amp;amp;cashier_awake);
/* The cashier has placed his information in the cashier_exchange variable.
-&amp;gt; Get it! */
sem_t *order = cashier_exchange.order;
sem_t *burger = cashier_exchange.burger;
uint8_t cashier_id = cashier_exchange.id;
/* Leave syncronized area, now the next customer may aquire a cashier */
sem_post(&amp;amp;customer_private_mutex);
/* Print a status about the approached cashier */
printf(&amp;quot;[CUSTOMER %d]\t Approached cashier no. %d.\n&amp;quot;,
customer_id, cashier_id);
/* Print status that now the order will be placed */
printf(&amp;quot;[CUSTOMER %d]\t Placing order to cashier no. %d.\n&amp;quot;,
customer_id, cashier_id);
/* Place order */
sleep(rand() % WAITING_TIME);
/* Tell cashier about the order */
sem_post(order);
/* Wait for cashier to hand over the burger */
sem_wait(burger);
/* Process done, burger received. Print status and exit */
printf(&amp;quot;[CUSTOMER %d]\t Got burger from cashier no. %d. Thank you!\n&amp;quot;,
customer_id, cashier_id);
printf(&amp;quot;[CUSTOMER %d]\t DONE.\n&amp;quot;, customer_id);
return NULL;
}
void assure_state() {
/* Assure that more than zero burgers are available */
if(burger_count &amp;lt; 0) {
printf(&amp;quot;[ASSURE_STATE]\t ERROR: Negative burger count!\n&amp;quot;);
exit(40);
}
/* Assure that not more burgers than rack spaces are available */
if(burger_count &amp;gt; RACK_HOLDER_SIZE) {
printf(&amp;quot;[ASSURE_STATE]\t ERROR: Rack overfull!\n&amp;quot;);
exit(41);
}
printf(&amp;quot;[ASSURE_STATE]\t State consistent.\n&amp;quot;);
```

Output:

```
[COOK 0]          CREATED.
[COOK 0]          DONE.
[COOK 1]          CREATED.
[COOK 2]          CREATED.
[CASHIER 0]       CREATED.
[CASHIER 1]       CREATED.
[CUSTOMER 0]      CREATED.
[CUSTOMER 1]      CREATED.
[CUSTOMER 2]      CREATED.
[CUSTOMER 3]      CREATED.
[CASHIER 0]       Serving customer.
[CUSTOMER 1]      Approached cashier no. 0.
[CUSTOMER 1]      Placing order to cashier no. 0.
[ASSURE_STATE]    State consistent.
[ASSURE_STATE]    State consistent.
[COOK 2]          Placed new burger in rack.
[ASSURE_STATE]    State consistent.
[ASSURE_STATE]    State consistent.
[COOK 2]          Placed new burger in rack.
[CASHIER 1]       Serving customer.
[CUSTOMER 0]      Approached cashier no. 1.
[CUSTOMER 0]      Placing order to cashier no. 1.
[ASSURE_STATE]    State consistent.
[ASSURE_STATE]    State consistent.
[COOK 1]          Placed new burger in rack.
[CASHIER 1]       Got order.
[CASHIER 1]       Going to rack to get burger...
[CASHIER 0]       Got order.
[CASHIER 0]       Going to rack to get burger...
[ASSURE_STATE]    State consistent.
[ASSURE_STATE]    State consistent.
[CASHIER 0]       Got burger from rack, going back
[CASHIER 0]       Gave burger to customer.
[CASHIER 0]       Serving customer.
[CUSTOMER 1]      Got burger from cashier no. 0. Thank you!
[CUSTOMER 1]      DONE.
[CUSTOMER 2]      Approached cashier no. 0.
```


[CUSTOMER 2] Approached cashier no. 0.
[CUSTOMER 2] Placing order to cashier no. 0.
[ASSURE_STATE] State consistent.
[ASSURE_STATE] State consistent.
[CASHIER 1] Got burger from rack, going back
[CASHIER 1] Gave burger to customer.
[CASHIER 1] Serving customer.
[CUSTOMER 0] Got burger from cashier no. 1. Thank you!
[CUSTOMER 0] DONE.
[CUSTOMER 3] Approached cashier no. 1.
[CUSTOMER 3] Placing order to cashier no. 1.
[ASSURE_STATE] State consistent.
[ASSURE_STATE] State consistent.
[COOK 2] Placed new burger in rack.
[CASHIER 0] Got order.
[CASHIER 0] Going to rack to get burger...
[ASSURE_STATE] State consistent.
[ASSURE_STATE] State consistent.
[COOK 1] Placed new burger in rack.
[ASSURE_STATE] State consistent.
[ASSURE_STATE] State consistent.
[CASHIER 0] Got burger from rack, going back
[CASHIER 0] Gave burger to customer.
[CUSTOMER 2] Got burger from cashier no. 0. Thank you!
[CUSTOMER 2] DONE.
[CASHIER 1] Got order.
[CASHIER 1] Going to rack to get burger...
[ASSURE_STATE] State consistent.
[ASSURE_STATE] State consistent.
[COOK 2] Placed new burger in rack.
[ASSURE_STATE] State consistent.
[ASSURE_STATE] State consistent.
[CASHIER 1] Got burger from rack, going back
[ASSURE_STATE] State consistent.
[CUSTOMER 3] Got burger from cashier no. 1. Thank you!
[CUSTOMER 3] DONE.
[ASSURE_STATE] State consistent.
[COOK 1] Placed new burger in rack.
[CASHIER 1] Gave burger to customer.
[MAIN] SUCCESS: All customers terminated

[MAIN] SUCCESS: Starting Cleanup
[CASHIER 0] Serving customer.
[MAIN] SUCCESS: Told all threads to terminate themselves
[CASHIER 1] Serving customer.
[ASSURE_STATE] State consistent.
[ASSURE_STATE] State consistent.
[COOK 2] Placed new burger in rack.
[ASSURE_STATE] State consistent.
[ASSURE_STATE] ERROR: Rack overfull!
...Program finished with exit code 41
Press ENTER to exit console.

Experiment-15	Simulate Bankers Algorithm for Deadlock Avoidance
Date-	

```
#include&lt;stdio.h&gt;
 void main()
{
int n,r,i,j,k,p,u=0,s=0,m;
int block[10],run[10],active[10],newreq[10];
int max[10][10],resalloc[10][10],resreq[10][10];
int totalloc[10],totext[10],simalloc[10];
//clrscr();
printf(&quot;Enter the no of processes:&quot;);
scanf(&quot;%d&quot;,&amp;n);
printf(&quot;Enter the no ofresource classes:&quot;);
scanf(&quot;%d&quot;,&amp;r);
printf(&quot;Enter the total existed resource in each class:&quot;);
for(k=1; k&lt;=r; k++)
scanf(&quot;%d&quot;,&amp;totext[k]);
printf(&quot;Enter the allocated resources:&quot;);
for(i=1; i&lt;=n; i++)
for(k=1; k&lt;=r; k++)
scanf(&quot;%d&quot;,&amp;resalloc);
printf(&quot;Enter the process making the new request:&quot;);
scanf(&quot;%d&quot;,&amp;p);
printf(&quot;Enter the requested resource:&quot;);
for(k=1; k&lt;=r; k++)
scanf(&quot;%d&quot;,&amp;newreq[k]);
printf(&quot;Enter the process which are n blocked or running:&quot;);
for(i=1; i&lt;=n; i++)
{
if(i!=p)
{
printf(&quot;process %d:\n&quot;,i+1);
scanf(&quot;%d%d&quot;,&amp;block[i],&amp;run[i]);
}
}
block[p]=0;
run[p]=0;
for(k=1; k&lt;=r; k++)
{
j=0;
for(i=1; i&lt;=n; i++)
{
totalloc[k]=j+resalloc[i][k];
j=totalloc[k];
}
}
for(i=1; i&lt;=n; i++)

{
if(block[i]==1||run[i]==1)
active[i]=1;
else
active[i]=0;
```

```
}
for(k=1; k&lt;=r; k++)
{
resalloc[p][k]+=newreq[k];
totalloc[k]+=newreq[k];
}
for(k=1; k&lt;=r; k++)
{
if(totext[k]-totalloc[k]&lt;0)
{
u=1;
break;
}
}
if(u==0)
{
for(k=1; k&lt;=r; k++)
simalloc[k]=totalloc[k];
for(s=1; s&lt;=n; s++)
for(i=1; i&lt;=n; i++)
{
if(active[i]==1)
{
j=0;
for(k=1; k&lt;=r; k++)
{
if((totext[k]-simalloc[k])&lt;(max[i][k]-resalloc[i][k]))
{
j=1;
break;
}
}
}
if(j==0)
{
active[i]=0;
for(k=1; k&lt;=r; k++)
simalloc[k]=resalloc[i][k];
}
}
m=0;
for(k=1; k&lt;=r; k++)
resreq[p][k]=newreq[k];
printf(&quot;Deadlock willn&#39;t occur&quot;);
}
else
{
for(k=1; k&lt;=r; k++)
{

resalloc[p][k]=newreq[k];

totalloc[k]=newreq[k];
}
printf(&quot;Deadlock will occur&quot;);
}
```

}

Sample Output:

Enter the no of processes:4
Enter the no ofresource classes:3
Enter the total existed resource in each class:3 2 2
Enter the allocated resources:1 0 0 5 1 1 2 1 1 0 0 2
Enter the process making the new request:2
Enter the requested resource:1 1 2
Enter the process which are n blocked or running:process 2:
1 2
process 4:
1 0
process 5: