LABORATORY MANUAL

on

OPERATING SYSTEM LAB

of

4th semester of Information Technology

GOVERNMENT POLYTECHNIC, BHUBANESWAR
Government of Odisha
ସରକାରୀ ବହୁବୃତ୍ତି ଅନୁଷ୍ଠାନ, ଭୁବନେଶ୍ୱର

VISION AND MISSION OF THE INSTITUTE

VISION

The Vision of the Government Polytechnic, Bhubaneswar is to be a premier Diploma Institution in the Eastern India by transforming the learners into competent technocrats by imparting technical education in multiple disciplines.

MISSION

The mission of the Government Polytechnic, Bhubaneswar is to:

- Focus on quality teaching and learning by providing better academic environment for both faculties and students.
- Develop professionally and technically skilled students who work for betterment of the society
- Nurture positive attitude and social values amongst the students and staff.

VISION AND MISSION OF THE DEPARTMENT

DEPARTMENT OF INFORMATION TECHNOLOGY

VISION

The Vision of the Department of Information Technology is to develop knowledge and skill of students by quality education to make responsible and competent technocrats who will work for the needs of industries and betterment of the society.

MISSION

The Mission of the Department of Information Technology is to:

M-1. Provide an excellent learning environment for the students to be technically efficient in solving problems.

M-2. Prepare the students with strong fundamental concepts, analytical capability and programming skills.

M-3. Create a nurturing environment for lifelong learning.

M-4. Generate social awareness and responsibility within the students to serve the mankind and protect the environment.

Program Educational Objectives (PEOs)

Program Educational Objective of Diploma in Information Technology:

The Program Educational Objectives (PEOs) of the Department of Information Technology are:

PEO-1: ***Strong Fundamentals & Technically skilled:*** *To equip the students with sound knowledge in mathematics, science and basic engineering fundamentals and develop skill to solve engineering problems.*

PEO-2: ***Higher study and Professional responsibilities:*** *To enable students to pursue higher studies for professional growth and/or work as an entrepreneur for successful professional career.*

PEO-3: ***Teamwork and lifelong learning:*** *To inculcate soft skill, professional and ethical attitude, teamwork, leadership skills and multidisciplinary approach within the students which will enable them to correlate societal issue with engineering solution and pursue lifelong learning*

Program Outcomes and Program Specific Outcomes (PSOs)

Program Outcomes for an Engineering diploma graduate:

i) **Basic and Discipline specific knowledge:** Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the engineering problems

ii) **Problem analysis:** Identify and analyze well-defined engineering problems using codified standard methods.

iii) **Design/ development of solutions:** Design solutions for well-defined technical problems and assist with the design of systems components or processes to meet specified needs.

iv) **Engineering Tools, Experimentation and Testing:** Apply modern engineering tools and appropriate technique to conduct standard tests and measurements.

v) **Engineering practices for society, sustainability and environment:** Apply appropriate technology in context of society, sustainability, environment and ethical practices.

vi) **Project Management:** Use engineering management principles individually, as a team member or a leader to manage projects and effectively communicate about well-defined engineering activities.

vii) **Life-long learning:** Ability to analyze individual needs and engage in updating in the context of technological changes.

viii) Program Specific Outcome of Diploma in Information Technology:

ix) PSO-1: Ability to design and develop methodologies in computer programming and apply those skills in the field of Information Technology.

x) **PSO-2:** Develop potentiality to apply the skills learnt for betterment of the society by updating it on the basis of the advanced technological changes and effectively communicate about well-defined engineering activities.

Pr.1-OPERATING SYSTEM LAB

Total Periods	**60**	**Maximum Marks**	**50 Marks**
Lab. Periods:	**4 Periods /week**	**Term Works**	**25 Marks**
Examination	**3hours**	**End Semester Examination**	**25Marks**

A. LIST OF PRACTICALS:-

1. Write a Shell script to print the command line arguments in reverse order.
2. Write a Shell script to check whether the given number is palindrome or not.
3. Write a Shell script to sort the given array elements in ascending order using bubble sort.
4. Write a Shell script to perform sequential search on a given array elements.
5. Write a Shell script to perform binary search on a given array elements.
6. Write a Shell script to accept any two file names and check their file permissions.
7. Write a Shell script to read a path name, create each element in that path e.g: a/b/c i.e., 'a' is directory in the current working directory, under 'a' create 'b', under 'b' create 'c'.
8. Write a Shell script to illustrate the case-statement.
9. Write a Shell script to accept the file name as arguments and create another shell script, which recreates these files with its original contents.
10. Write a Shell script to demonstrate Terminal locking.
11. Write a Shell script to accept the valid login name, if the login name is valid then print its home directory else an appropriate message.
12. Write a Shell script to read a file name and change the existing file permissions.
13. Write a Shell script to print current month calendar and to replace the current day number by '*' or '**' respectively.
14. Write a Shell Script to display a menu consisting of options to display disk space, the current users logged in, total memory usage, etc. (using functions.)
15. Write a C-program to fork a child process and execute the given Linux commands.
16. Write a C-program to fork a child process, print owner process ID and its parent process ID.
17. Write a C-program to prompt the user for the name of the environment variable, check its validity and print an appropriate message.
18. Write a C-program to READ details of N students such as student name, reg number, semester and age. Find the eldest of them and display his details.

Books Recommended:-

Sl.No	Name of Authors	Title of the Book	Name of the publisher
1	Sumitabha Das, 4th Edition,	"UNIX – Concepts and Applications",	Tata McGraw Hill, 2006.
3	Yashvant Kanetkar	Unix Sell Programming 1st edition	BPB Publication

COURSE OUTCOMES:

After completion of this course the student will be able to

1. understand the fundamental concept of UNIX/ LINUX and use basic UNIX/ LINUX commands
2. create the codes in Shell programming.
3. Develop the codes using UNIX/LINUX system calls.

COs, POs and PSOs Mapping

Cos, Pos and PSOs mapping	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PSO1	PSO2
CO1	2	-	-	1	1	-	2	1	1
CO2	2	-	-	1	-	-	2	1	1
CO3	2	-	-	1	1	-	1	1	1
Total	6	-	-	3	2	-	5	3	3
Average	2	-	-	1	1	-	1.67	1	1

Ex.No:1.a	BASICS OF UNIX COMMANDS
	INTRODUCTION TO UNIX

AIM:

To study about the basics of UNIX

UNIX:

It is a multi-user operating system. Developed at AT & T Bell Industries, USA in 1969.

Ken Thomson along with Dennis Ritchie developed it from MULTICS (Multiplexed Information and Computing Service) OS.

By1980, UNIX had been completely rewritten using C language.

LINUX:

It is similar to UNIX, which is created by Linus Torualds. All UNIX commands works in Linux. Linux is a open source software. The main feature of Linux is coexisting with other OS such as windows and UNIX.

STRUCTURE OF A LINUXSYSTEM:

It consists of three parts.

a)UNIX kernel
b) Shells
c) Tools and Applications

UNIX KERNEL:

Kernel is the core of the UNIX OS. It controls all tasks, schedule all Processes and carries out all the functions of OS.

Decides when one programs tops and another starts.

SHELL:

Shell is the command interpreter in the UNIX OS. It accepts command from the user and analyses and interprets them

Ex.No:1.b	BASICS OF UNIX COMMANDS
	BASIC UNIX COMMANDS

AIM:

To study of Basic UNIX Commands and various UNIX editors such as vi, ed, ex and EMACS.

CONTENT:

Note: Syn->Syntax

a) **date**

–used to check the date and time

Syn:$date

Format	Purpose	Example	Result
+%m	To display only month	$date+%m	06
+%h	To display month name	$date+%h	June
+%d	To display day of month	$date+%d	O1
+%y	To display last two digits of years	$date+%y	09
+%H	To display hours	$date+%H	10
+%M	To display minutes	$date+%M	45
+%S	To display seconds	$date+%S	55

b) **cal**

–used to display the calendar

Syn:$cal 2 2009

c)**echo**

–used to print the message on the screen.

Syn:$echo “text”

d)**ls**

–used to list the files. Your files are kept in a directory.

Syn:$lsls–s

All files (include files with prefix)

ls–l Lodetai (provide file statistics)

ls–t Order by creation time

ls– u Sort by access time (or show when last accessed together with –l)

ls–s Order by size

ls–r Reverse order

ls–f Mark directories with /,executable with* , symbolic links with @, local sockets with =, named pipes(FIFOs)with

ls–s Show file size

ls– h“ Human Readable”, show file size in Kilo Bytes & Mega Bytes (h can be used together with –l or)

ls[a-m]*List all the files whose name begin with alphabets From ‘a’ to ‘m’
ls[a]*List all the files whose name begins with ‘a’ or ‘A’
Eg:$ls>my list Output of ‘ls’ command is stored to disk file named ‘my list’

e)**lp**
–used to take printouts
Syn:$lp filename
f)**man**
–used to provide manual help on every UNIX commands.
Syn:$man unix command
$man cat

g)**who** & **whoami**
–it displays data about all users who have logged into the system currently. The next command displays about current user only.
Syn:whowhoami

h)**uptime**
–tells you how long the computer has been running since its last reboot or power-off.
Syn:$uptime

i)**uname**
–it displays the system information such as hardware platform, system name and processor, OS type
Syn:$uname–a

j)**hostname**
–displays and set system host name
Syn:$ hostname

k)**bc**
–stands for ‘best calculator’

```
$bc
10/2*3
15
```

```
$ bc
scale =1
2.25+1
3.35
quit
```

```
$ bc
ibase=2
obase=16
11010011
89275
1010
Ā
Quit
```

```
$ bc
sqrt(196)
14 quit
```

```
$bc
for(i=1;i<3;i=i+1)I
1
2
3 quit
```

```
$ bc-l
scale=2
s(3.14)
0
```

FILE MANIPULATION COMMANDS

a)**cat**–this create, view and concatenate files.

Creation:

Syn:$cat>filename

Viewing:

Syn:$cat filename

Add text to an existing file:

Syn:$cat>>filename

Concatenate:

Syn:$catfile1file2>file3

$catfile1file2>>file3 (no over writing of file3)

b)**grep**–used to search a particular word or pattern related to that word from the file.

Syn:$grep search word filename

Eg:$grep anu student

c)**rm**–deletes a file from the file system

Syn:$rm filename

d)**touch**–used to create a blank file.

Syn:$touch file names

e)**cp–**copies the files or directories

Syn:$cpsource file destination file

Eg:$cp student stud

f)**mv–**to rename the file or directory

syn:$mv old file new file

Eg:$mv–i student student list(-i prompt when overwrite)

g)**cut**–it cuts or pickup a given number of character or fields of the file.

Syn:$cut<option><filename>

Eg: $cut –c filename

$cut–c1-10emp

$cut–f 3,6emp

$ cut –f 3-6 emp

-c cutting columns

-f cutting fields

h)**head**–displays10 lines from the head(top)of a given file

Syn:$head filename

Eg:$head student

To display the top two lines:

Syn:$head-2student

i)**tail**–displays last 10 lines of the file
Syn:$tail filename
Eg:$tail student
To display the bottom two lines;
Syn:$ tail -2 student

j)**chmod**–used to change the permissions of a file or directory.
Syn:$ch mod category operation permission file
Where, Category–is the user type
Operation–is used to assign or remove permission
Permission–is the type of permission
File–are used to assign or remove permission all

Examples:
$chmodu-wx student
Removes write and execute permission for users
$ch modu+rw,g+rwstudent
Assigns read and write permission for users and groups
$chmodg=rwx student
Assigns absolute permission for groups of all read, write and execute permissions

k)**wc**–it counts the number of lines, words, character in a specified file(s)
with the options as –l,-w,-c

Category	Operation	Permission
u– users g–group o– others	+assign -remove =assign absolutely	r– read w– write x-execute

Syn: $wc –l filename
$wc –w filename
$wc–c filename

Ex.No:1.c	BASICS OF UNIX COMMANDS
	UNIX EDITORS

AIM:

To study of various UNIX editors such as vi, ed, ex and EMACS.

CONCEPT:

Editor is a program that allows user to see a portions a file on the screen and modify characters and lines by simply typing at the current position. UNIX supports variety of Editors. They are:

ed ex vi
EMACS

Vi- vi is stands for "visual".vi is the most important and powerful editor.vi is a full screen editor that allows user to view and edit entire document at the same time.vi editor was written in the University of California, at Berkley by Bill Joy, who is one of the co-founder of Sun Microsystems.

Features of vi:

It is easy to learn and has more powerful features.
Itworksgreatspeedandiscasesensitive.vihaspowerfulundofunctionsandhas3modes:

1. Command mode
2. Insert mode
3. Escape or ex mode

In command mode, no text is displayed on the screen.
In Insert mode, it permits user to edit insert or replace text.
In escape mode, it displays commands at command line.
Moving the cursor with the help of h, l, k, j, I, etc

EMACS Editor

Motion Commands:
M-> Move to end of file
M-< Move to beginning of file
C-v Move forward a screen M –v Move
backward a screen C –n Move to next line
C-p Move to previous line
C-a Move to the beginning of the line
C-e Move to the end of the line
C-f Move forward a character
C-b Move backward a character
M-f Move forward a word
M-b Move backward a word

Deletion Commands:

DEL	delete the previous character C -d
	delete the current character M -DEL
	delete the previous word
M-d	delete the next word
C-x DEL	deletes the previous sentence
M-k	delete the rest of the current sentence
C-k	deletes the rest of the current line
C-xu	undo the lasted it change

Search and Replace in EMACS:

y	Change the occurrence of the pattern
n	Don't change the occurrence, but look for the other q Don't change. Leave query replace completely
!	Change this occurrence and all others in the file

Experiment 1

Write a Shell script to print the command line arguments in reverse order.

```
#!/bin/sh
if [ $# -eq 0 ]
then
    echo "no arguments given"
    exit
fi
echo "Total number of arguments: $#"
echo "The arguments are: $*"
echo "The arguments in reverse order:"
for (( i=$#;i>0;i-- ));do
        echo "${!i}"
done
```

Output:

```
┌──(kali㉿kali)-[~/…/4th_sem/OS_LAB/experiments/exp-1]
└─$ bash rev1.sh ab bc cd
Total number of arguments: 3
The arguments are: ab bc cd
The arguments in reverse order:
cd
bc
ab
```

Experiment 2

Write a Shell script to check whether the given number is palindrome or not.

```
#!/bin/sh

echo "Enter a number"
read num
input_num=$num

while [ $num -gt 0 ]
# -gt is an arithmetic comparison operator that returns true if the first operand is greater than the second operand.
# The reason for using -gt is that we want to check if the number is greater than 0 because palindrome is only defined for numbers greater than 0
do
        rem=$(( num % 10 ))
        num=$(( num / 10 ))
        rev=$(( rev * 10 + rem ))
done

if [ $rev -eq $input_num ]
# -eq is an arithmetic comparison operator that returns true if the first operand is equal to the second operand.
# The reason for using -eq is that we want to check if the number is equal to the reversed number.
then
        echo "$input_num is palindrome"
else
        echo "$input_num not palindrome"
fi
```

Output

```
┌──(kali㉿kali)-[~/…/4th_sem/OS_LAB/experiments/exp-2]
└─$ bash palindrome.sh
Enter a number
191
191 is palindrome
```

EXPERIMENT 3

Write a Shell script to sort the given array elements in ascending order using bubble sort

```
#!/bin/sh

echo "Enter the elements: "
read -a arr

# To iterate over the array using for loop
echo "The array is: "
for elem in ${arr[@]}
do
    printf $elem" "
done

# algorithm to sort the array using bubble sort
for (( i=0; i<${#arr[@]}; i++ ))
do
    for (( j=0; j<${#arr[@]}-1; j++ ))
    do
        if [ ${arr[$j]} -gt ${arr[$((j+1))]} ]
        then
            temp=${arr[$j]}
            arr[$j]=${arr[$((j+1))]}
            arr[$((j+1))]=$temp
        fi
    done
done

# To iterate over the array using for loop
printf "\nThe sorted array is: "
for elem in ${arr[@]}
do
    printf $elem" "
done
```

OUTPUT:

```
┌──(kali㉿kali)-[~/…/4th_sem/OS_LAB/experiments/exp-3]
└─$ bash bubble-sort.sh
Enter the elements:
10 2 5 8 9
The array is:
10 2 5 8 9
The sorted array is: 2 5 8 9 10
```

EXPERIMENT 4

Write a Shell script to perform sequential search on a given array elements.

```
#!/bin/bash

#Write a Shell script to perform sequential search on a given array elements.

echo "Enter the elements: "
read -a arr

# To iterate over the array using for loop
echo -n "The array is: "
for elem in ${arr[@]}
do
    printf $elem" "
done

index=-1

echo -e "\nEnter the element you want to search: "

read search_number


for elem in ${arr[@]}
do
    index=$((index+1))
    if [ $search_number -eq $elem ]
    then
    echo "Element found at index $index"
    exit
    fi
done
echo "$search_number is not present in the array"
```

OUTPUT:

```
┌──(kali㉿kali)-[~/…/4th_sem/OS_LAB/experiments/exp-4]
└─$ bash linear-search.sh
Enter the elements:
78 9 4 7 2 3
The array is: 78 9 4 7 2 3
Enter the element you want to search:
7
Element found at index 3
```

EXPERIMENT 5

Write a Shell script to perform binary search on a given array elements.

```
#!/bin/sh

#Write a Shell script to perform binary search on a given array elements.

echo "Enter the elements: "
read -a elements

for iteams in ${elements[@]}
do
    printf $iteams" "
done

#sorting the array

for (( i=0; i<${#elements[@]}; i++ ))
do
    for (( j=0; j<${#elements[@]}-1; j++ ))
    do
        if [ ${elements[$j]} -gt ${elements[$((j+1))]} ]
        then
            temp=${elements[$j]}
            elements[$j]=${elements[$((j+1))]}
            elements[$((j+1))]=$temp
        fi
    done
done

for nums in ${elements[@]}
do
echo $nums" "
done

#Binary search algorithm

function binary_search {
    min=0
    max=${#elements[@]}-1
    search=$1

    while [ $min -le $((max-1)) ]
    do
        mid=$(((min+max)/2))
        if [ $search -eq ${elements[$mid]} ]
        then
            echo "Element found "
            return
        elif [ $search -lt ${elements[$mid]} ]
        then
            max=$mid
        else
            min=$mid
        fi
    done

    if [ $min -gt $((max-1)) ]
    then
        echo "Element not found"
    fi
    return 0
}

echo "Enter the element you want to search: "
read search_number

#calling the function to search the element
binary_search $search_number
```

OUTPUT:

```
┌──(rootkali㉿kali)-[~/…/4th_sem/OS_LAB/experiments/exp-5]
└─$ bash binary-search.sh
Enter the elements:
56 69 87
56 69 87 56
69
87
Enter the element you want to search:
69
Element found
```

EXPERIMENT 6

Write a Shell script to accept any two file names and check their file permissions

```
#!/bin/sh

#Write a Shell script to accept any two file names and check their file permissions.

echo "Enter the file names: "
read -a files

ls -l ${files[@]}
```

OUTPUT:

```
┌──(rootkali㉿kali)-[~/…/4th_sem/OS_LAB/experiments/exp-6]
└─$ bash file-permission.sh
Enter the file names:
file1.txt
-rw-r--r-- 1 rootkali rootkali 23 Jun 22 16:46 file1.txt

┌──(rootkali㉿kali)-[~/…/4th_sem/OS_LAB/experiments/exp-6]
└─$
```

EXPERIMENT 7

Write a Shell script to read a path name, create each element in that path e.g: a/b/c i.e., ‘a’ is directory in the current working directory, under ‘a’ create ‘b’, under ‘b’ create ‘c’

```
#!/bin/sh

#Write a Shell script to read a path name, create each element in that path e.g: a/b/c
#i.e., ‘a’ is directory in the current working directory, under ‘a’ create ‘b’, under ‘b’ create
#‘c’

echo "Enter the path name: "
read path

cd $path
mkdir a
cd a
mkdir b
cd b
mkdir c
cd c
pwd
```

OUTPUT:

```
┌──(rootkali㉿kali)-[~/…/4th_sem/OS_LAB/experiments/exp-7]
└─$ bash create-dirs.sh
Enter the path name:
a
/home/rootkali/lab/4th_sem/OS_LAB/experiments/exp-7/a/a/b/c
```

EXPERIMENT 8

Write a Shell script to illustrate the case-statement.

```
#!/bin/sh

#Write a Shell script to illustrate the case-statement.

echo "Enter any character: "
read char

case $char in
    [a-z])
        echo "Lower case"
        ;;
    [A-Z])
        echo "Upper case"
        ;;
    [0-9])
        echo "Digit"
        ;;
    *)
        echo "Special character"
        ;;
esac

echo "Enter a-z: "
read a
case $a in
    a)
        echo "You entered a"
        ;;
    b)
        echo "You entered b"
        ;;
    c)
        echo "You entered c"
        ;;
    *)
        echo "other"
        ;;
esac
```

OUTPUT:

```
┌──(kali㉿kali)-[~/…/4th_sem/OS_LAB/experiments/exp-8]
└─$ bash case.sh
Enter any character:
A
Upper case
Enter a-z:
c
You entered c
```

EXPERIMENT 9

Write a Shell script to accept the file name as arguments and create another shell script, which recreates these files with its original contents

```
#!/bin/sh

#Write a Shell script to accept the file name as arguments and create another shell
#script, which recreates these files with its original contents.

echo "Enter the file names: "
read -a files

current_file=${0##*/}

for file in ${files[@]}
do
    cat $current_file > $file
done
```

OUTPUT:

$ create-dirs.sh | $ case.sh | $ duplicate-script.sh | $ gyana.sh U

4th_sem > OS_LAB > experiments > exp-10 > $ terminal-lock.sh

```
1   #!/bin/sh
2
3   #Write a Shell script to demonstrate Terminal locking
4
5   read -p "Press enter to continue"
6
7   echo "Enter a password: "
8   read -s password
9
10  echo "Terminal locked :- Enter password to unlock..."
11
12  read input
13
14  while [ $input -ne $password ]
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
┌──(rootkali㉿kali)-[~/…/4th_sem/OS_LAB/experiments/exp-9]
└─$ bash duplicate-script.sh
Enter the file names:
gyana.sh

┌──(rootkali㉿kali)-[~/…/4th_sem/OS_LAB/experiments/exp-9]
└─$ bash duplicate-script.sh
Enter the file names:
gyana.sh

┌──(rootkali㉿kali)-[~/…/4th_sem/OS_LAB/experiments/exp-9]
└─$ bash duplicate-script.sh
Enter the file names:
gyana.sh

┌──(rootkali㉿kali)-[~/…/4th_sem/OS_LAB/experiments/exp-9]
└─$
```

EXPERIMENT 10

Write a Shell script to demonstrate Terminal locking.

```
#!/bin/sh

#Write a Shell script to demonstrate Terminal locking

read -p "Press enter to continue"

echo "Enter a password: "
read -s password

echo "Terminal locked :- Enter password to unlock..."

read input

while [[ $input -ne $password ]]
do
    echo "Incorrect password"
    echo "Enter password again: "
    read input
done

echo "Terminal unlocked"
```

OUTPUT:

```
┌──(rootkali㉿kali)-[~/…/4th_sem/OS_LAB/experiments/exp-10]
└─$ bash terminal-lock.sh
Press enter to continue
Enter a password:
Terminal locked :- Enter password to unlock...
qwpo
Terminal unlocked
```

EXPERIMENT 11

Write a Shell script to accept the valid login name, if the login name is valid then print its home directory else an appropriate message.

```
#!/bin/sh

#Write a Shell script to accept the valid login name, if the login name is valid then print
#its home directory else an appropriate message.

clear

if [ $# -eq 0 ]
then
    echo "ERROR: No argument passed"
    exit
fi

while [ $1 ]
do
    cat /etc/passwd | cut -d ":" -f1 | grep "^$1" > temp
    name=`cat temp`
    if [ "$name" != "$1" ]
    then
        echo "ERROR: Invalid login name \"$1\""
    else
        echo "Home Directory for user $1 is :- "
        echo `cat /etc/passwd | grep "^$1" | cut -d ":" -f6`
    fi
    exit
done
```

OUTPUT:

```
Home Directory for user rootkali is :-
/home/rootkali

┌──(rootkali㉿kali)-[~/…/4th_sem/OS_LAB/experiments/exp-11]
└─$
```

EXPERIMENT 12

Write a Shell script to read a file name and change the existing file permissions.

```
#!/bin/sh

#Write a Shell script to read a file name and change the existing file permissions.

echo "Enter a file name: "
read file

function menu() {
    echo -e "1. Remove read, write and execute permissions"
    echo -e "2. Add read, write and execute permissions"
    echo -e "3. Exit"

    echo -e "Enter your choice: "
    read choice

    case $choice in
        1)
            echo "Removing read, write and execute permissions"
            chmod -x $file
            ;;
        2)
            echo "Adding read, write and execute permissions"
            chmod +x $file
            ;;
        3)
            echo "Exiting..."
            exit
            ;;
        *)
            echo "Invalid choice"
            ;;
    esac
    file_permissions
    menu
}
#print file permissions
function file_permissions {
    echo -e "\nFile permissions for $file are: "
    echo `stat -c %A $file`

    menu
}
file_permissions
```

OUTPUT:

```
┌──(rootkali㉿kali)-[~/…/4th_sem/OS_LAB/experiments/exp-12]
└─$ gcc exp-18.c

┌──(rootkali㉿kali)-[~/…/4th_sem/OS_LAB/experiments/exp-12]
└─$ ./a.out
Enter the number of students: 2
Enter the details of student 1:
Name: GyanaRanjan
Reg no: 01
Semester: 4
Age: 20
Enter the details of student 2:
Name: BasuDev
Reg no: 02
Semester: 6
Age: 21


.... All students list ....
Student 1:
Name: GyanaRanjan
Reg no: 1
Semester: 4
Age: 20

Student 2:
Name: BasuDev
Reg no: 2
Semester: 6
Age: 21



.... Shorted student list ....
Name: BasuDev
Reg no: 2
Semester: 6
Age: 21

Name: GyanaRanjan
Reg no: 1
Semester: 4
Age: 20
```

Experiment-13

Write a Shell script to print current month calendar and to replace the current day number by ‘*’ or ‘**’ respectively.

4th_sem > OS_LAB > experiments > exp-12 > $ calander.sh

```
1   #!/bin/sh
2
3   cal|sed -r "s/\b$(date|cut -d' ' -f4)\b/*/"
```

OUTPUT

```
└─$ bash calander.sh
cal: setlocale: No such file or directory
     July 2022
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  *  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

Experiment-14

Write a Shell Script to display a menu consisting of options to display disk space, the current users logged in, total memory usage, etc. (using functions.)

```
#!/bin/bash
# function to show memory usage
memoryUsage(){
    echo "Memory Usage:"
    free
    read -p "Press any key to Continue...."
}
# function to show disk usage
diskUsage(){
    echo "Disk Usage:"
    df
    read -p "Press any key to Continue...."
}
# function to show menu
show_menu()
{
    clear
    echo "++++++++++++ MENU ++++++++++++"
    echo "1. Show Memory Usage."
    echo "2. Show DIsk Usage."
    echo "3. Exit"
    echo "++++++++++++++++++++++++++++++"
}
# function to take input
take_input()
{
     #take the input and store it in choice variable
    local choice
    read -p "Select the option from above menu: " choice

     #using swich case statement check the choice and call function.
    case $choice in
        1) memoryUsage ;;
        2) diskUsage ;;
        3) exit 0;;
        *) echo "Enter Valid Option!!"
            read -p "Press any key to Continue...."
        esac
    }

# for loop to call the show_menu and take_input function.
while true
do
    show_menu
    take_input
done
```

OUTPUT

```
++++++++++++ MENU ++++++++++++
1. Show Memory Usage.
2. Show DIsk Usage.
3. Exit
++++++++++++++++++++++++++++++
Select the option from above menu: 1
Memory Usage:
              total        used        free      shared  buff/cache   available
Mem:        4012660     1011852     2296968      107552      703840     2674592
Swap:        998396           0      998396
Press any key to Continue....

++++++++++++ MENU ++++++++++++
1. Show Memory Usage.
2. Show DIsk Usage.
3. Exit
++++++++++++++++++++++++++++++
Select the option from above menu: 2
Disk Usage:
Filesystem     1K-blocks     Used Available Use% Mounted on
udev             1971460        0   1971460   0% /dev
tmpfs             401268      952    400316   1% /run
/dev/sda1       91697864 26665292  60328572  31% /
tmpfs            2006328    68400   1937928   4% /dev/shm
tmpfs               5120        0      5120   0% /run/lock
tmpfs             401264       64    401200   1% /run/user/1000
++++++++++++ MENU ++++++++++++
1. Show Memory Usage.
2. Show DIsk Usage.
3. Exit
++++++++++++++++++++++++++++++
Select the option from above menu: 3
```

Experiment-15

Write a C-program to fork a child process and execute the given Linux commands.

→

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

int main()
{
   fork();
   fork();
   fork();
   fork();
   printf("Using fork() system call");
   return 0;
}
```

OUTPUT

```
└─$ gcc Fork.c

┌──(kali㉿  kali)-[~]
└─$ ./a.out

Using fork() system call


Using fork() system call
Using fork() system call
Using fork() system call
Using fork() system call
Using fork() system call
Using fork() system call
Using fork() system call
Using fork() system call
Using fork() system call

Using fork() system callUsing fork() system call

Using fork() system callUsing fork() system call
Using fork() system call
┌──(kali㉿  kali)-[~]
└─$
```

Experiment-16

Write a C-program to fork a child process, print owner process ID and its parent process ID.

→

```
#include<stdio.h>
#include <unistd.h>
int main()
{
	int i;
	printf("hello before fork \n");
	printf("i : %d\n",i);

	i=fork();
	printf("\n");

	if(i==0)
	{
		printf("Child has started\n\n");
		printf("child printing first time \n");

		printf("getpid : %d getppid : %d \n",getpid(),getppid());
		sleep(5);
		printf("\nchild printing second time \n");
		printf("getpid : %d getppid : %d \n",getpid(),getppid());
	}
	else
	{
		printf("parent has started\n");
		printf("getpid : %d  getppid : %d \n",getpid(),getppid());
		printf("\n");
	}
	printf("Hi after fork i : %d\n",i);
	return 0;
}
```

OUTPUT

```
┌──(kali㉿  kali)-[~]
└─$ gcc ForkID.c

┌──(kali㉿  kali)-[~]
└─$ ./a.out
hello before fork
i : 21865

parent has started
getpid : 2771  getppid : 1109

Hi after fork i : 2772


Child has started

child printing first time
getpid : 2772 getppid : 1
┌──(kali㉿  kali)-[~]
└─$
child printing second time
getpid : 2772 getppid : 1
Hi aft  r fork i : 0


┌──(kali㉿  kali)-[~]
└─$
```

Experiment-17

Write a C-program to prompt the user for the name of the environment variable, check its validity and print an appropriate message.

→

```
#include <stdio.h>
int main(int argc, char *argv[], char * envp[])
{
   int i;
   for (i = 0; envp[i] != NULL; i++)
      printf("\n%s", envp[i]);
   getchar();
   return 0;
}
```

OUTPUT

```
ALLUSERSPROFILE=C:\ProgramData
APPDATA=C:\Users\Amit\AppData\Roaming
CommonProgramFiles=C:\Program Files (x86)\Common Files
CommonProgramFiles(x86)=C:\Program Files (x86)\Common Files
CommonProgramW6432=C:\Program Files\Common Files
COMPUTERNAME=DESKTOP-KO0VINC
ComSpec=C:\WINDOWS\system32\cmd.exe
DriverData=C:\Windows\System32\Drivers\DriverData
HOMEDRIVE=C:
HOMEPATH=\Users\Amit
IntelliJ IDEA Community Edition=C:\Program Files\JetBrains\I
java_home=C:\Program Files (x86)\Java\jdk1.7.0\bin\;
LOCALAPPDATA=C:\Users\Amit\AppData\Local
LOGONSERVER=\\DESKTOP-KO0VINC
NUMBER_OF_PROCESSORS=8
OneDrive=C:\Users\Amit\OneDrive
OneDriveConsumer=C:\Users\Amit\OneDrive
OS=Windows_NT


PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBI
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_ARCHITEW6432=AMD64
PROCESSOR_IDENTIFIER=Intel64 Family
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=8e0c
ProgramData=C:\ProgramData
ProgramFiles=C:\Program Files (x86)
ProgramFiles(x86)=C:\Program Files (
ProgramW6432=C:\Program Files
PROMPT=$P$G
PSModulePath=C:\Program Files\Windows
PT8HOME=C:\Program Files\Cisco Packe
PUBLIC=C:\Users\Public
QT_DEVICE_PIXEL_RATIO=auto
SESSIONNAME=Console
SystemDrive=C:
SystemRoot=C:\WINDOWS
TEMP=C:\Users\Amit\AppData\Local\Temp
TMP=C:\Users\Amit\AppData\Local\Temp
USERDOMAIN=DESKTOP-KO0VINC
USERDOMAIN_ROAMINGPROFILE=DESKTOP-KO
USERNAME=GP
USERPROFILE=C:\Users\Amit
VBOX_MSI_INSTALL_PATH=C:\Program File
windir=C:\WINDOWS
ZES_ENABLE_SYSMAN=1

C:\Users\Amit\Desktop>
```

Experiment-18

Write a C-program to READ details of N students such as student name, reg number, semester and age. Find the eldest of them and display his details.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    int n;
    printf("Enter the number of students: ");
    scanf("%d", &n);
    struct student {
        char name[20];
        int reg_no;
        int semester;
        int age;
    };
    struct student *students = (struct student *)malloc(n * sizeof(struct student));
    for (int i = 0; i < n; i++) {
        printf("Enter the details of student %d:\n", i + 1);
        printf("Name: ");
        scanf("%s", students[i].name);
        printf("Reg no: ");
        scanf("%d", &students[i].reg_no);
        printf("Semester: ");
        scanf("%d", &students[i].semester);
        printf("Age: ");
        scanf("%d", &students[i].age);
    }
    // print all the students
    printf("\n\n.... All students list ....\n");

    for (int i = 0; i < n; i++) {
        printf("Student %d:\n", i + 1);
        printf("Name: %s\n", students[i].name);
        printf("Reg no: %d\n", students[i].reg_no);
        printf("Semester: %d\n", students[i].semester);
        printf("Age: %d\n\n", students[i].age);
    }
    // short student accroding to name
    qsort(students, n, sizeof(struct student), (int (*)(const void *, const void *))strcmp);
    printf("\n\n.... Shorted student list ....\n");
    for(int i = 0; i < n; i++) {
        printf("Name: %s\n", students[i].name);
        printf("Reg no: %d\n", students[i].reg_no);
        printf("Semester: %d\n", students[i].semester);
        printf("Age: %d\n\n", students[i].age);
    }
    return 0;
}
```

OUTPUT

```
Unsorted Student Records:
Id = 1, Name = AA, Age = 19
Id = 2, Name = BB, Age = 20
Id = 3, Name = CC, Age = 19
Id = 4, Name = DD, Age = 18
Id = 5, Name = EE, Age = 18


Student Records sorted by Name:
Id = 1, Name = AA, Age = 19
Id = 2, Name = BB, Age = 20
Id = 3, Name = CC, Age = 19
Id = 4, Name = DD, Age = 18
Id = 5, Name = EE, Age = 18
PS D:\kali-07>
```