

Practical File of Operating System 22CS005

Submitted

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE & ENGINEERING



CHITKARA UNIVERSITY

**CHANDIGARH-PATIALA NATIONAL HIGHWAY
RAJPURA (PATIALA) PUNJAB-140401 (INDIA)**

June, 2024

Submitted To:

Dr. Vikas Solanki
Professor
Chitkara University, Punjab

Submitted By:

Gurnoor Singh
2310991582
2nd Sem. – Batch(2023)

INDEX

Sr. No.	Practical Name	Teacher Sign
1	a) Installation: Configuration & Customizations of Linux b) Introduction to GCC compiler: Basics of GCC, Compilation of program, Execution of program. c) Time stamping in Linux. d) Automating the execution using Make file.	
2	Implement the basic and user status commands like: su, sudo, man, help, history, who, whoami, id, uname, uptime, free, tty, cal, date, hostname, reboot, clear	
3	Implement the commands that is used for Creating and Manipulating files: cat, cp, mv, rm, ls and its options, touch and their options, which is, where is, what is	
4	Implement Directory oriented commands: cd, pwd, mkdir, rmdir, Comparing Files using diff, cmp, comm.	
5	Write a program and execute the same to demonstrate how to use terminal commands in C program (using system() function)	
6	Write a program to implement process concepts using C language by printing process Id.	
7	Write a program to create and execute process using fork() and exec() system calls.	
8	Write a C program to implement FCFS (First Come First Serve) and SJF (Shortest Job First) scheduling algorithms.	
9	Write a C program to implement Priority Scheduling and RR (Round-Robin) scheduling algorithms.	
10	Write a program to demonstrate the concept of deadlock in C by using shared variable.	
11	Introduction to File system, file system types and architecture and it's related commands.	
12	Write programs in C language to implement Disk Scheduling algorithms (FCFS (First Come First Serve), SSTF (Shortest Seek Time First), SCAN (Elevator Algorithm) and C-SCAN (Circular SCAN)).	

Program: 1(a)

Aim:

- a) Installation: Configuration & Customizations of Linux
- b) Introduction to GCC compiler: Basics of GCC, Compilation of program, Execution of program.
- c) Time stamping in Linux.
- d) Automating the execution using Make file.

Theory: Linux is an open-source operating system that can be run on a wide range of devices. Linux can be operated without a graphical interface through a text terminal. It is an operating system, like Windows or macOS, that manages your computer's hardware and software. It is free and open-source, meaning that anyone can use and modify it. Linux is known for its stability, security, and performance, and it can run on many different types of devices, from phones to servers.

What's in a Linux distribution?

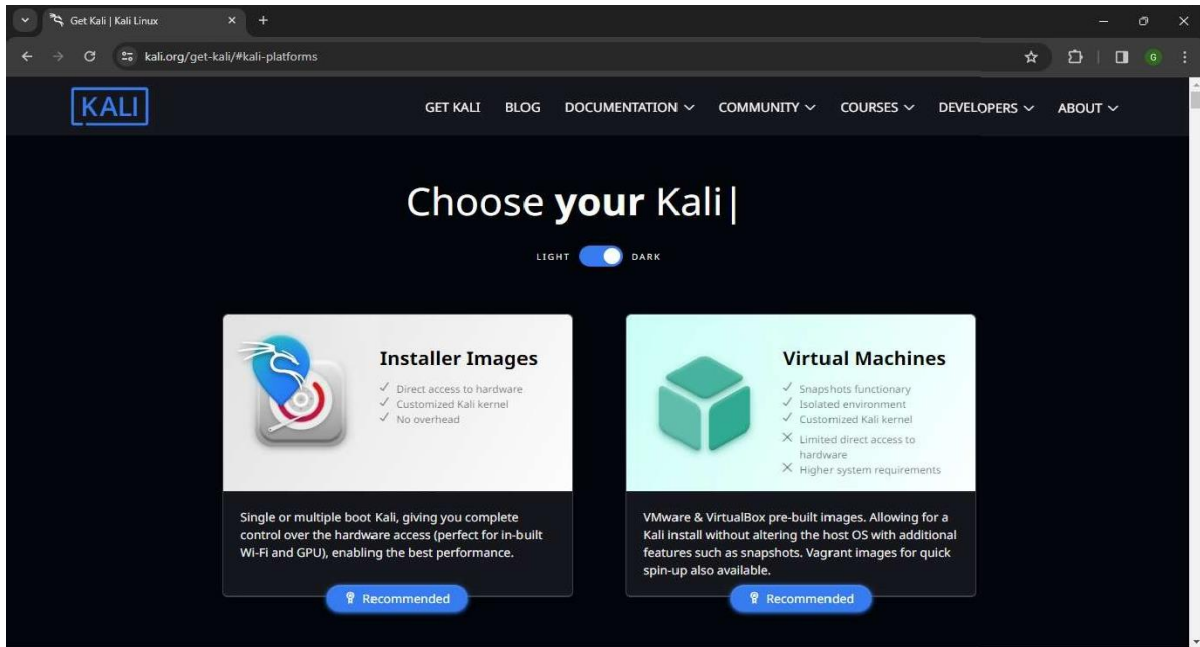
Linux refers to the family of operating systems based on the Linux kernel and each operating system is packaged as a *distribution*, or *distro*, of Linux. A Linux distribution is made from a software collection including the Linux kernel, GNU tools, and default software.

There are a large number of Linux distributions. Here is a short list of some of the most popular distributions:

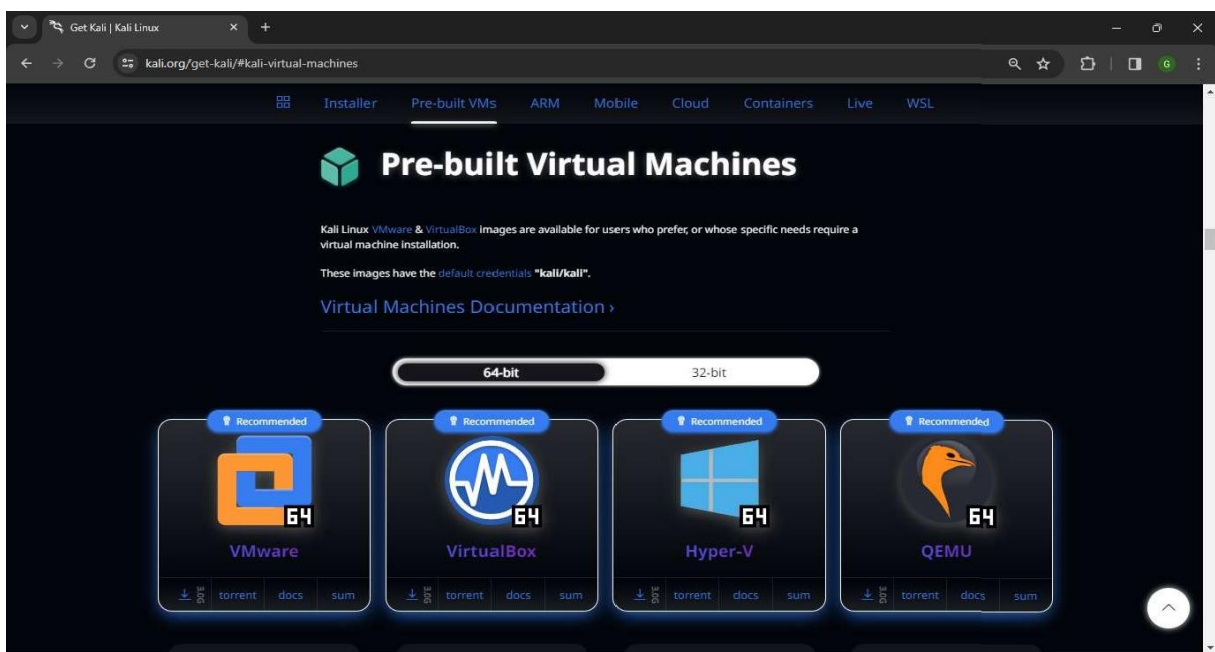
- **Debian** is one of the oldest distributions of Linux, first developed in 1993. It is a stable Linux operating system, and software updates are frequent but small.
- **Ubuntu** is the most popular distribution of Linux and is based on Debian Linux. It's widely used and supported and looks most like other operating systems like OSX and Windows in terms of usability.
- **LinuxMint** is a distribution that is based off of Ubuntu that comes with less pre-installed software than Ubuntu.
- **RedHatEnterpriseLinux**, also known as RHEL, is a distribution of Linux developed by Red Hat. It has strict rules around its trademark which prevent free distribution, but it is mostly used in enterprise environments on servers.
- **Fedora** is an open-source community-driven distribution of Linux that is backed by Red Hat. Think of it as the Ubuntu equivalent to Red Hat.
- **Arch** is a rolling release distribution of Linux that is 100% developed by its community. It has a steeper learning curve than other distributions but it is a great lightweight distribution of Linux.

Procedure:

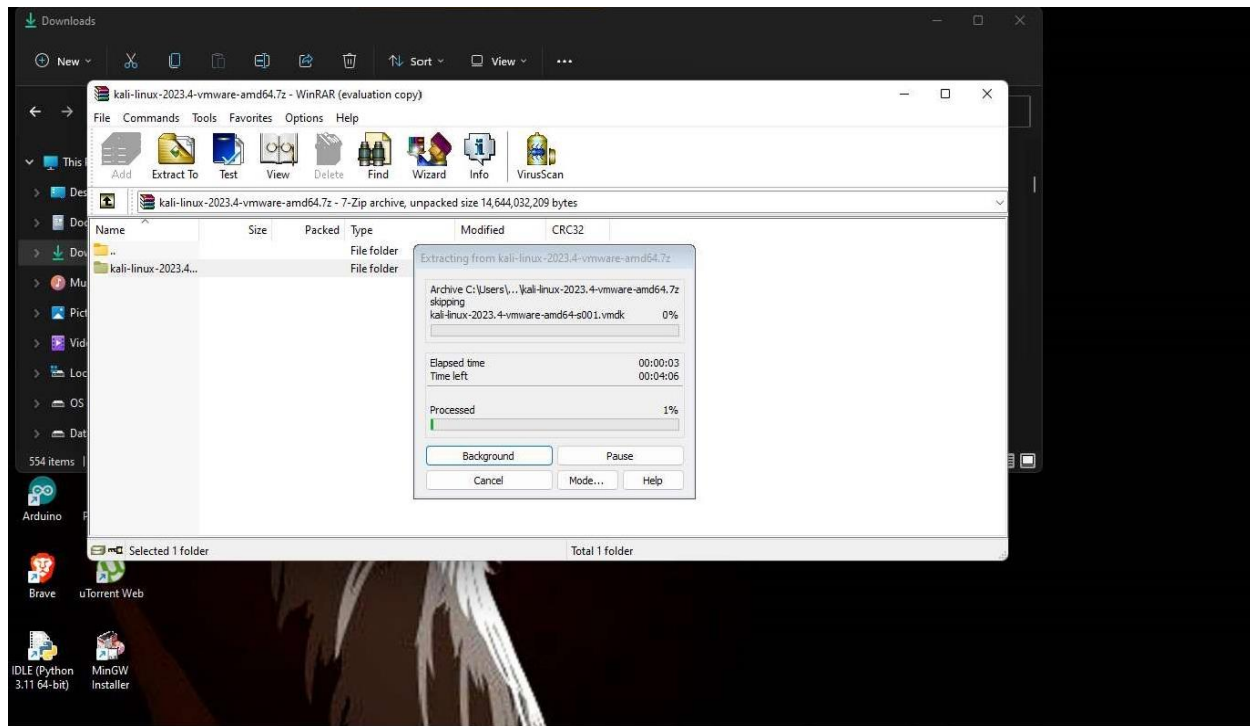
1. Go to browser and search kali (website link: <https://www.kali.org/get-kali/#kali-platforms>).



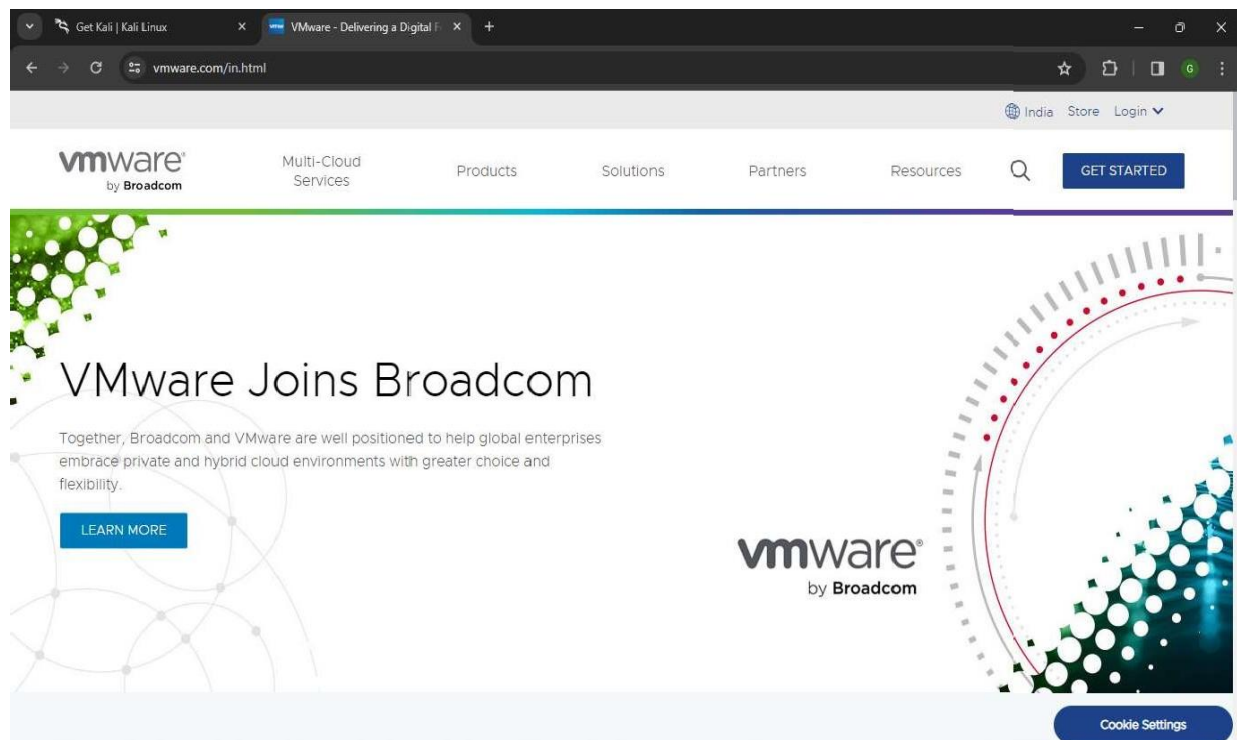
2. Install pre built virtual machines according to the system type of laptop/computer i.e. 64-bit operating system, x64-based processor, 32-bit operating system, x32-based processor.



- When Zip file was installed and open the zip file to extracte the file from zip file.



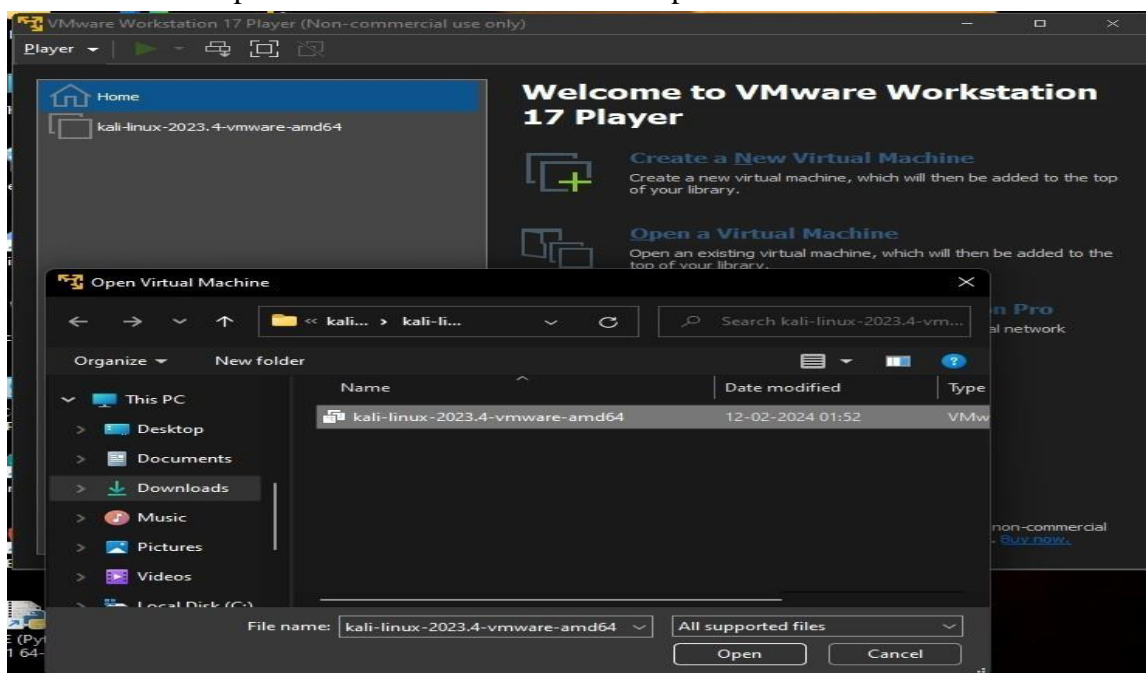
- Go to browser and search the vmware workstation(Website link: <https://www.vmware.com/in.html>).



5. Install the VMware workstation according to the system type of laptop/computer i.e. 64-bit operating system, x64-based processor, 32-bit operating system, x32-based processor. Open the VMware and finish its setting.



6. Open VMware and select the option to open the virtual machine and select the link which is extrate from the zip file.click Ok and user name and password of kali is kali.



About: VMware is a virtualization platform that enables the creation and management of virtual machines (VMs) on a host system. It provides a robust environment for running multiple operating systems concurrently, allowing users to emulate various hardware configurations within a single physical machine. VMware offers features such as snapshotting, which enables the saving and restoration of VM states, and virtual networking to simulate complex network setups.

Program: 1(b)

Aim: To illustrate basic of GCC compiler i.e. compilation and execution of a program on Linux terminal.

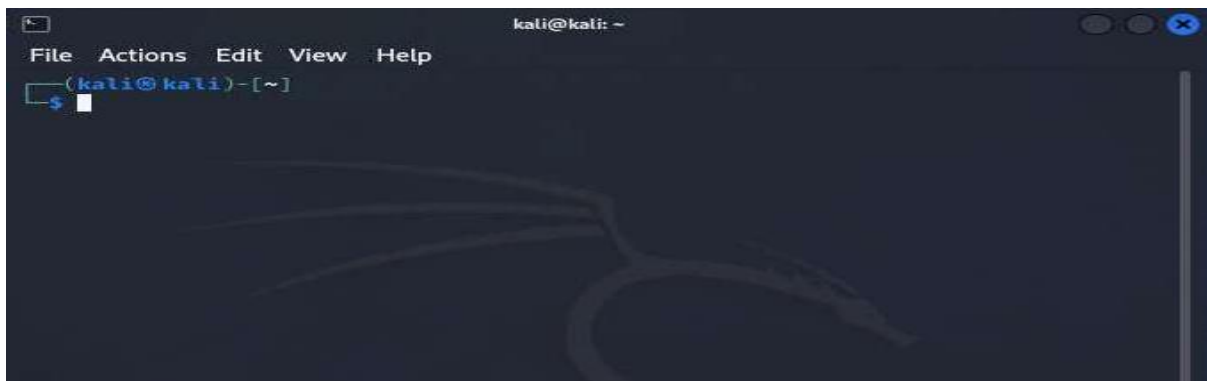
Theory: The term compiler refers to a piece of software that converts our source code from a high-level programming language to a low-level programming language (machine-level code) to build an executable program file and in Linux Operating

Systems and compile C program in Linux, we'll need to install the GCC Compiler. In Ubuntu repositories, GCC Compiler is a part of the build-essential package, and this package is exactly what we will be installing in our Linux Operating System. If you're interested in learning more about the build-essential meta-package, You can refer here.

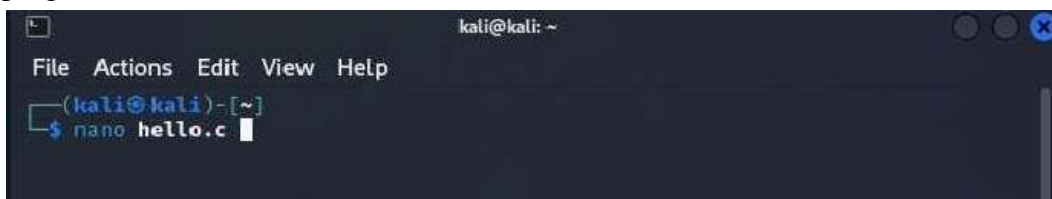
GCC Compiler (GNU Compiler Collection) is a collection of compilers and libraries for the programs written in C, C++, Ada, GO, D, Fortran, and Objective-C programming languages and is distributed under the GNU General Public License.

Procedure:

1. Open the terminal in kali.



2. Enter the command “**nano hello.c**” to open the file text editor where we will create a basic c program which we have named as “**hello.c**”.



3. You'll enter the text file editor. Type a simple program to print “Hello World!”.


```

kali@kali: ~
File Actions Edit View Help
GNU nano 7.2 hello.c *
#include<stdio.h>

int main(){
    printf("hello world");
}

[ ^W = Ctrl+W  M-W = Alt+W ]
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify
  
```

4. Save the file with your code. If using Nano editor, you can press `Ctrl + X`, shift Y and press “Enter button” to exit editor.

```

kali@kali: ~
File Actions Edit View Help
GNU nano 7.2 hello.c *
#include<stdio.h>

int main(){
    printf("hello world");
}

File Name to Write: hello.c
^G Help      M-D DOS Format  M-A Append     M-B Backup File
^C Cancel    M-M Mac Format  M-P Prepend    ^T Browse
  
```

5. Use the GCC compiler to compile the C Program. Enter the command “**gcc hello.c**” to compile the c program OR Enter the command “**gcc hello.c -o result**” to compile the c program and save the output of the c program in result(result is the output file name).

Command: gcc hello.c



```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
$ gcc hello.c
```

6. To generate the output of the c program, type “**./result**” in the terminal.
“Hello World” will be printed in the terminal.



```
(kali@kali)-[~]  
$ ./a.out  
hello world  
(kali@kali)-[~]  
$
```

Program: 1(c)

AIM: Time stamping in Linux.

Theory: In Linux, time stamping involves associating specific times with events or files, crucial for tracking modifications, access, and changes. The system clock and file system metadata are utilized to record accurate time stamps, offering valuable information for auditing, debugging, and monitoring system behavior, thus enhancing overall security and reliability. Commands like `stat` and `ls` can be employed to extract and interpret time stamp information in Linux.

The **`stat`** command in Linux is used to display detailed information about a file or file system. It provides various statistics, including file size, inode number, access permissions, and crucially for time stamping, the creation, modification, and access times of a file.

The **`stat`** command in Linux is indeed a useful tool for obtaining information about files, including their access, modification, and change timestamps. When you use the `stat` command on a file, it provides detailed information, including the timestamps associated with the file. These timestamps are:

1. Access Time: This timestamp indicates the last time the file was accessed or read.
2. Modification Time: This timestamp reflects the last time the file's content was modified or changed.
3. Change Time: This timestamp signifies the last time the file's metadata (such as permissions or ownership) was modified.

Procedure:

1. Open the terminal kali.
2. Make changes to file or create a new one file using a text editor like **nano**. You can create a file named "example.txt"



```
(kali@kali)-[~]  
$ nano example.txt
```

3. You'll enter the text file editor. Write "Hello World".



```
kali@kali: ~  
File Actions Edit View Help  
GNU nano 7.2 example.txt  
hello world  
  
[ Read 1 line ]  
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute  
^X Exit      ^R Read File  ^N Replace    ^U Paste      ^_ Justify
```

4. Save the file with your text. If using Nano editor, you can press `Ctrl + X`, shift Y and press "Enter" button to exit editor.

Use commands like **stat** or **ls** to view time stamps associated with the file.

```
(kali@kali)-[~]  
$ stat example.txt  
File: example.txt  
Size: 12          Blocks: 8          IO Block: 4096   regular file  
Device: 8,1      Inode: 3944028    Links: 1  
Access: (0644/-rw-r--r--)  Uid: ( 1000/   kali)   Gid: ( 1000/   kali)  
Access: 2024-02-11 15:54:24.623924716 -0500  
Modify: 2024-02-11 15:53:26.206845170 -0500  
Change: 2024-02-11 15:53:26.206845170 -0500  
Birth: 2024-02-11 15:53:26.206845170 -0500
```

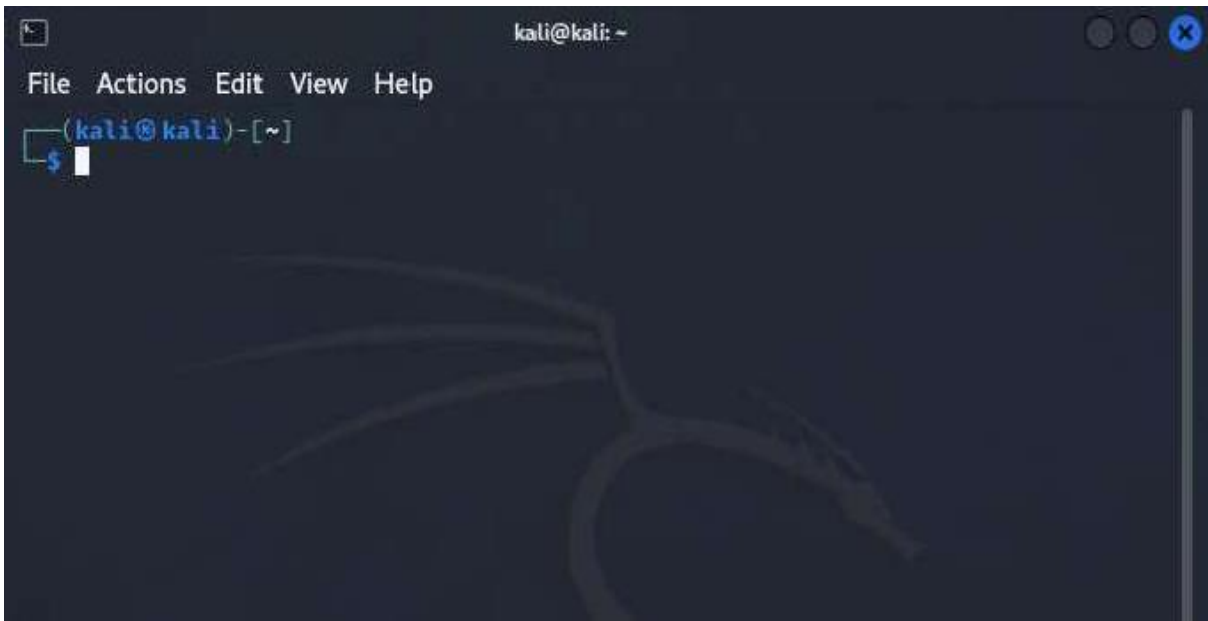
Program: 1(d)

AIM: Automating the execution using Makefile.

Theory: Makefiles automate the compilation and execution process of software projects. They contain rules specifying dependencies and commands for building target files. By defining dependencies and associated actions, Makefiles streamline tasks, ensuring only necessary components are recompiled or executed when source code changes. This automation simplifies complex build processes, saving time and reducing errors. Makefiles use a declarative syntax, enabling developers to efficiently manage and maintain project workflows, fostering consistency in software development environments.

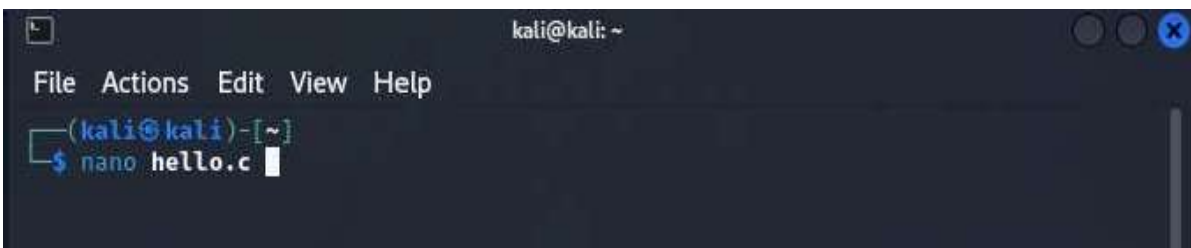
Procedure:

1. Open the terminal in kali.

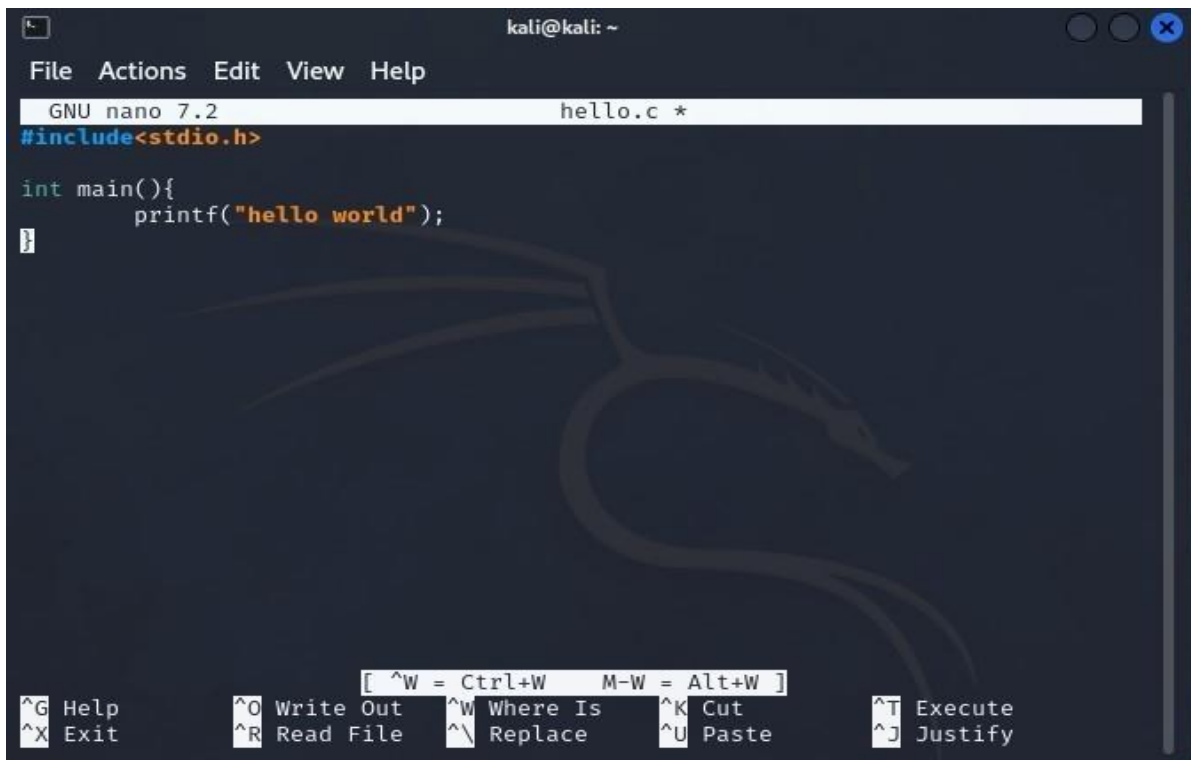


2. Enter the command “**nano hello.c**” to open the file text editor where we will create a basic c program which we have named as “**hello.c**”.

Command: nano hello.c



3. You'll enter the text file editor. Type a simple program to print "Hello World!".



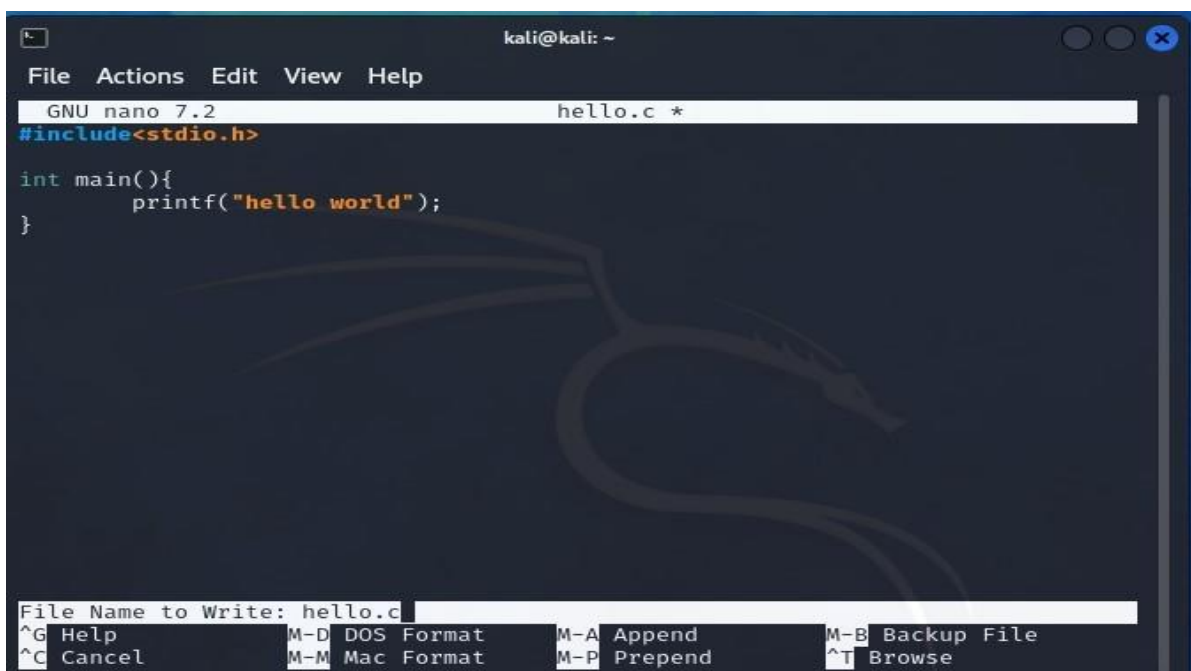
The screenshot shows the GNU nano 7.2 text editor interface. The title bar indicates the file is 'hello.c *'. The menu bar includes File, Actions, Edit, View, and Help. The code being typed is a simple C program:

```
#include<stdio.h>

int main(){
    printf("hello world");
}
```

The bottom status bar shows various keyboard shortcuts: ^G Help, ^X Exit, ^O Write Out, ^R Read File, ^W Where Is, ^\ Replace, M-W = Alt+W, ^K Cut, ^U Paste, ^T Execute, and ^J Justify.

4. Save the file with your code. If using Nano editor, you can press `Ctrl + X`, shift Y and "enter button" to exit editor.



The screenshot shows the GNU nano 7.2 text editor interface. The title bar indicates the file is 'hello.c *'. The menu bar includes File, Actions, Edit, View, and Help. The code being typed is a simple C program:

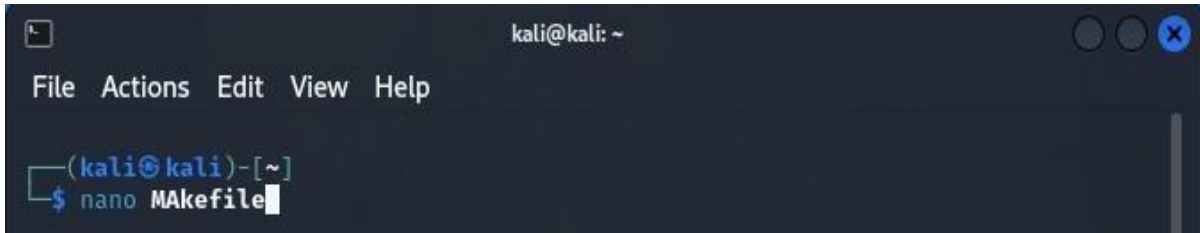
```
#include<stdio.h>

int main(){
    printf("hello world");
}
```

The bottom status bar shows various keyboard shortcuts: ^G Help, ^C Cancel, M-D DOS Format, M-M Mac Format, M-A Append, M-P Prepend, M-B Backup File, and ^T Browse. The prompt 'File Name to Write: hello.c' is visible at the bottom.

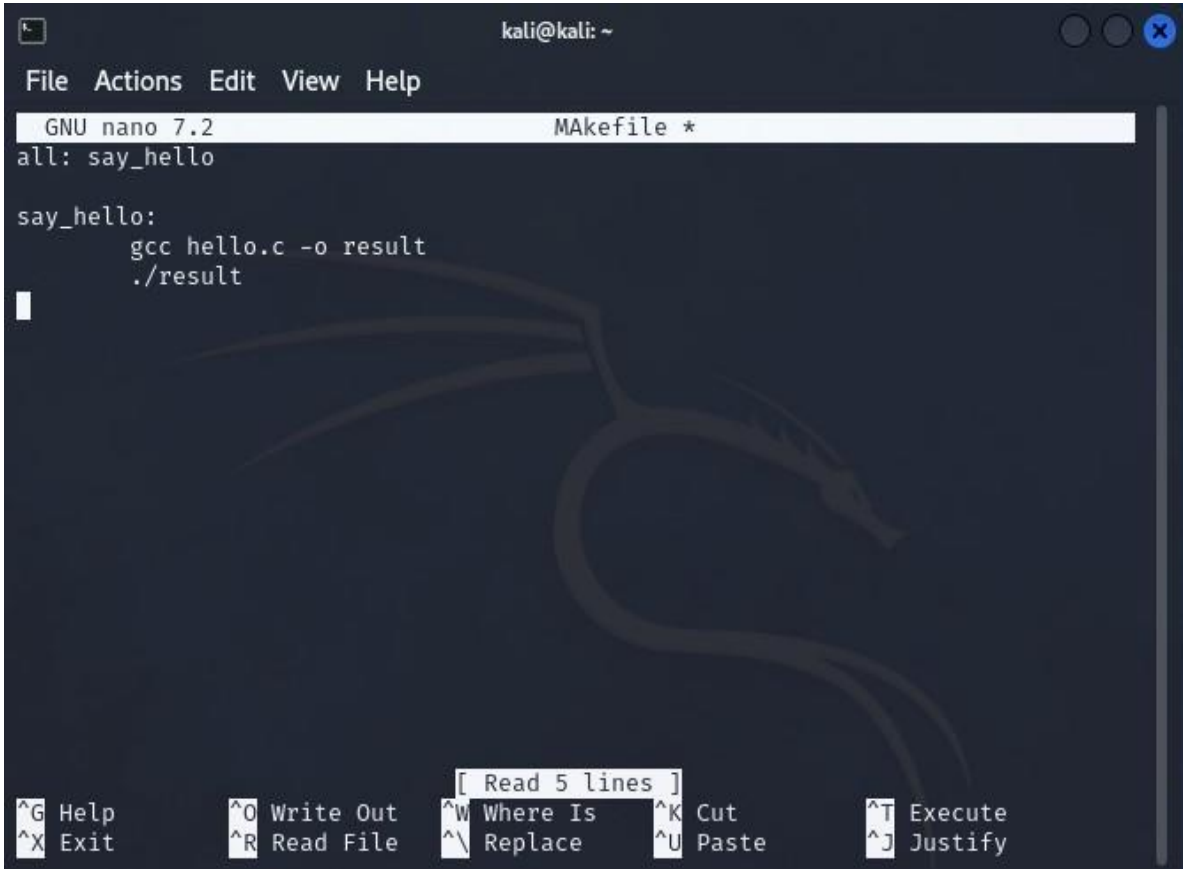
5. Enter the command “**nano Makefile**” to open the file text editor which we have named as “**Makefile**”.

Command: nano Makefile



```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
$ nano Makefile
```

6. You'll enter the text file editor. Define the target which we have named as “**say_hello**” and write the command which you want to execute.

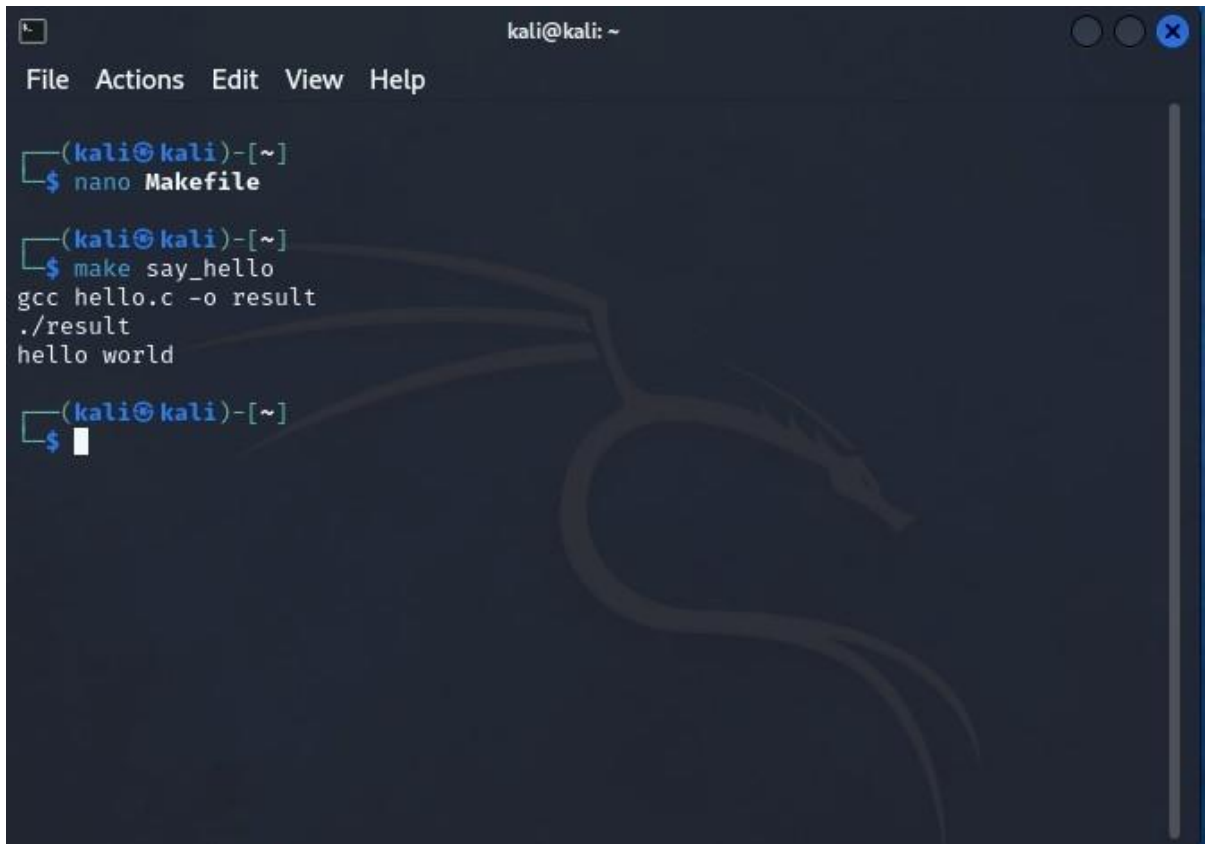


```
kali@kali: ~  
File Actions Edit View Help  
GNU nano 7.2 Makefile *  
all: say_hello  
  
say_hello:  
    gcc hello.c -o result  
    ./result  
  
[ Read 5 lines ]  
^G Help    ^O Write Out  ^W Where Is  ^K Cut       ^T Execute  
^X Exit    ^R Read File  ^\ Replace   ^U Paste     ^J Justify
```

7. Save the file with your code. If using Nano editor, you can press `Ctrl + X`, shift Y and “enter button” to exit editor.

Use command “**make**” to see your output.

Command: make say_hello

A terminal window titled 'kali@kali: ~' with a menu bar (File, Actions, Edit, View, Help) and window control buttons. The terminal shows a sequence of commands and their output. A faint Kali Linux dragon logo is visible in the background.

```
kali@kali: ~  
File Actions Edit View Help  
  
(kali@kali)-[~]  
$ nano Makefile  
  
(kali@kali)-[~]  
$ make say_hello  
gcc hello.c -o result  
./result  
hello world  
  
(kali@kali)-[~]  
$
```

program: 2

AIM: Implement the basic and user status commands like: su, sudo, man, help, history, who, whoami, id, uname, uptime, free, tty, cal, date, hostname, reboot, clear Automating the execution using Make file.

Theory:

su (Switch User): This command allows users to switch to another user account. It's commonly used to gain administrative privileges by switching to the root user account (su root). It prompts for the password of the target user account unless invoked as root.

sudo (Superuser Do): sudo allows users to execute commands with the security privileges of another user, typically the superuser (root). It's often used to perform administrative tasks while logged in as a regular user. It provides a more controlled and audited way to execute commands as root.

man (Manual): The man command displays the manual pages for Unix and Unix-like operating systems. It provides detailed information about commands, functions, and file formats. It's an essential tool for learning about command usage and syntax.

help: Help provides built-in help for commands in the shell. It offers brief descriptions and usage information for shell built-ins and commands. It's specific to the shell being used (e.g., Bash, Zsh) and provides quick access to command syntax and options.

history: This command displays a list of previously executed commands. It helps users to recall and reuse commands from the command-line history. By default, it shows a numbered list of commands executed in the current shell session.

who: Shows information about currently logged-in users on the system, including their usernames, terminal sessions, and login times. It displays details such as user names, terminal line numbers, and login times.

whoami: Displays the username of the current user. It's useful for scripting and checking user identities. It provides a quick way to identify the current user without needing to inspect environment variables or configuration files.

id: Prints real and effective user and group IDs, along with additional information such as group memberships. It provides detailed information about the user and group identities associated with the current process.

uname (Unix Name): This command prints basic system information such as the operating system name, network node hostname, kernel release, version, and machine hardware. It's often used in scripts to determine the system's characteristics.

uptime: Displays how long the system has been running, along with system load averages. It provides insights into system performance and availability by showing how busy the system has been over various time intervals.

free: Shows the amount of free and used memory in the system, including physical and swap memory. It helps in monitoring memory usage and system performance by providing information about

available resources.

tty (Teletypewriter): This command prints the file name of the terminal connected to the standard input. It's often used to check the terminal device a user is using or to determine whether a process is running in the background.

cal (Calendar): Displays a calendar for the specified month or year. It's a simple tool for checking dates, planning, and viewing a calendar without leaving the command line interface.

date: Prints or sets the system date and time. It's commonly used for scripting, logging, and displaying the current date and time. It provides options to format the output according to specific requirements.

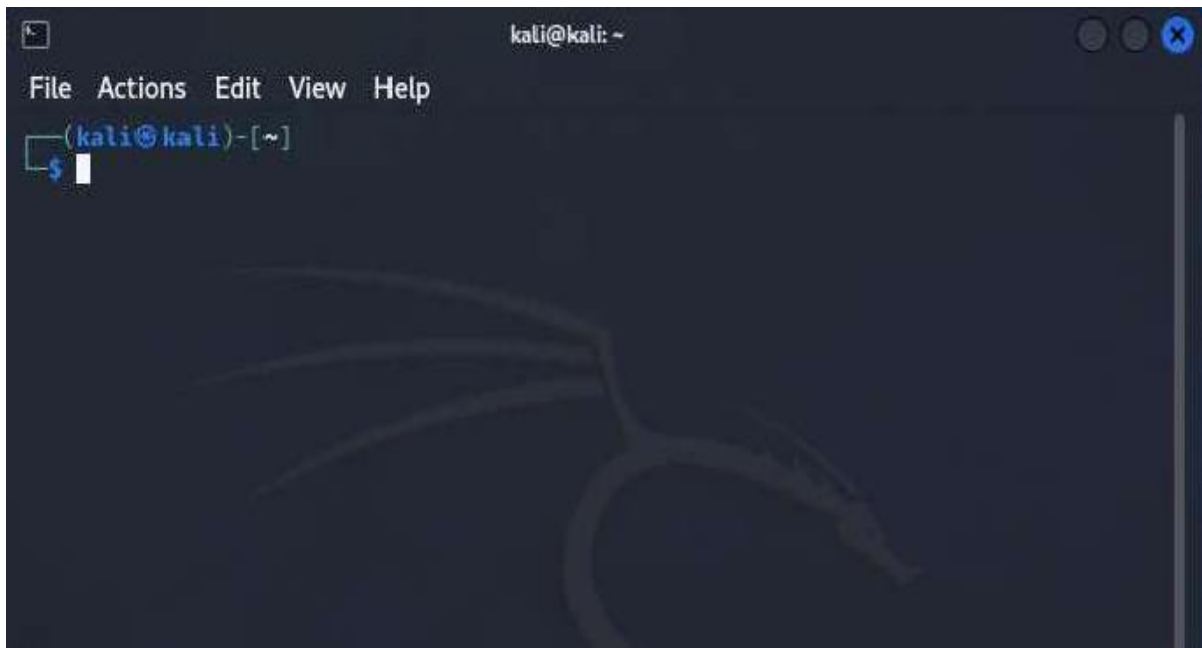
hostname: Prints the name of the current host system. It helps in identifying the system within a network and is often used in network configuration scripts or diagnostics.

reboot: Restarts the system. It's used to gracefully reboot the system, allowing for system updates or troubleshooting. It's a critical command but requires appropriate permissions to execute.

clear: Clears the terminal screen. It's used to remove previous output from the terminal, providing a clean workspace. It doesn't delete any history or command logs but simply clears the visible screen content.

Procedure:

1. Open the terminal in kali.



2. Enter the “su” command in terminal to switch the user e.g. for verifying enter the kali password.

command: su

```
kali@kali: ~
File Actions Edit View Help
Ign:8 http://kali.mirror.net.in/kali kali-rolling/non-free-firmware amd64 Packa
ges
Ign:2 http://kali.mirror.net.in/kali kali-rolling/main amd64 Packages
Err:2 http://http.kali.org/kali kali-rolling/main amd64 Packages
503 Service Unavailable [IP: 103.195.68.3 80]
Ign:4 http://kali.mirror.net.in/kali kali-rolling/contrib amd64 Packages
Ign:5 http://kali.mirror.net.in/kali kali-rolling/contrib amd64 Contents (deb)
Ign:6 http://kali.mirror.net.in/kali kali-rolling/non-free amd64 Packages
Ign:7 http://kali.mirror.net.in/kali kali-rolling/non-free amd64 Contents (deb)
Ign:8 http://kali.mirror.net.in/kali kali-rolling/non-free-firmware amd64 Packa
ges
Fetched 46.6 MB in 2min 42s (288 kB/s)
Reading package lists... Done
E: Failed to fetch http://http.kali.org/kali/dists/kali-rolling/main/binary-amd
64/Packages 503 Service Unavailable [IP: 103.195.68.3 80]
E: Some index files failed to download. They have been ignored, or old ones use
d instead.

(kali@kali)-[~]
$ su
Password:
su: Authentication failure
```

3. Enter the “**sudo**” command in terminal

Command : sudo apt update

```
kali@kali: ~
File Actions Edit View Help

(kali@kali)-[~]
$ sudo apt update
Ign:1 http://kali.mirror.net.in/kali kali-rolling InRelease
Ign:1 http://kali.mirror.net.in/kali kali-rolling InRelease
Ign:1 http://kali.mirror.net.in/kali kali-rolling InRelease
Err:1 http://kali.mirror.net.in/kali kali-rolling InRelease
503 Service Unavailable [IP: 103.195.68.3 80]
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
1077 packages can be upgraded. Run 'apt list --upgradable' to see them.
W: Failed to fetch http://http.kali.org/kali/dists/kali-rolling/InRelease 503
Service Unavailable [IP: 103.195.68.3 80]
W: Some index files failed to download. They have been ignored, or old ones use
d instead.

(kali@kali)-[~]
$
```

4. Enter the ” **man**” command in terminal for providing detailed information and usage instructions.

command : man man

```
(kali@kali)-[~]
$ man man
```

```
kali@kali: ~
File Actions Edit View Help
MAN(1) Manual pager utils MAN(1)

NAME
    man - an interface to the system reference manuals

SYNOPSIS
    man [man options] [[section] page ...] ...
    man -k [apropos options] regexp ...
    man -K [man options] [section] term ...
    man -f [whatis options] page ...
    man -l [man options] file ...
    man -w|-W [man options] page ...

DESCRIPTION
    man is the system's manual pager. Each page argument given to man is
    normally the name of a program, utility or function. The manual page
    associated with each of these arguments is then found and displayed.
    A section, if provided, will direct man to look only in that section
    of the manual. The default action is to search in all of the avail-
    able sections following a pre-defined order (see DEFAULTS), and to
    show only the first page found, even if page exists in several sec-
    tions.

    The table below shows the section numbers of the manual followed by
    Manual page man(1) line 1 (press h for help or q to quit)
```

5. Enter the "who" command in terminal.

Command : who

```
kali@kali: ~
File Actions Edit View Help

(kali@kali)-[~]
$ who
kali      tty7      2024-02-11 12:12 (:0)
```

6. Enter the "whoami" command in terminal for display the current user.

Command : whoami

```
kali@kali: ~
File Actions Edit View Help

(kali@kali)-[~]
$ whoami
kali
```

7. Enter the "id" command in terminal for user or group id.

Command : id


```
(kali㉿kali)-[~]  
$ id  
uid=1000(kali) gid=1000(kali) groups=1000(kali),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),100(users),101(netdev),106(bluetooth),113(scanner),135(wireshark),137(kaboxer)
```

8. Enter the “uname” command in terminal for basic information about the operating system.

Command : uname

```
(kali㉿kali)-[~]  
$ uname  
Linux
```

9. Enter the “uptime” command in terminal for how long the system has been running

Command : uptime

```
(kali㉿kali)-[~]  
$ uptime  
13:15:09 up 1:02, 1 user, load average: 0.11, 0.16, 0.11
```

10. Enter the “free” command in terminal for free and used memory.

Command : free

```
(kali㉿kali)-[~]  
$ free  


|       | total   | used   | free    | shared | buff/cache | available |
|-------|---------|--------|---------|--------|------------|-----------|
| Mem:  | 2003024 | 803604 | 638252  | 13640  | 727408     | 1199420   |
| Swap: | 1048572 | 0      | 1048572 |        |            |           |


```

11. Enter the “tty” command in terminal for file name of the terminal connected to the standard input.

Command : tty

```
(kali㉿kali)-[~]  
$ tty  
/dev/pts/0
```

12. Enter the “cal” command in terminal for print the month.

Command : cal

```
(snoopy@juily)-[~/Desktop/test]
$ cal
February 2024
Su Mo Tu We Th Fr Sa
    1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29
```

13. Enter the “date” command in terminal for print the current date.

Command : date

```
(kali@kali)-[~]
$ date
Sun Feb 11 01:26:34 PM EST 2024
```

14. Enter the “hostname” command in terminal.

Command : hostname

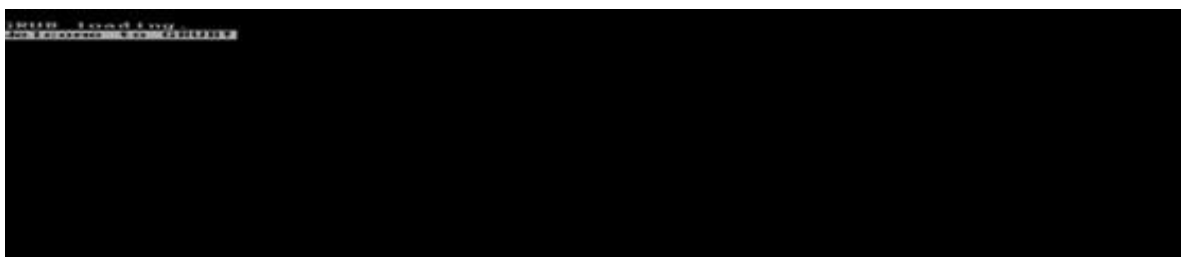
```
(kali@kali)-[~]
$ hostname
kali
```

15. Enter the “reboot” command in terminal for restart.

Command : reboot

```
kali@kali: ~
File Actions Edit View Help
(kali@kali)-[~]
$ reboot
```

Before



On Enter



16. Enter the “clear” command in terminal for clear the terminal screen.

Command : clear

```

kali@kali: ~
File Actions Edit View Help

(kali@kali)-[~]
$ sudo apt update
Get:1 http://kali.download/kali kali-rolling InRelease [41.5 kB]
Get:2 http://kali.download/kali kali-rolling/main amd64 Packages [19.6 MB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [46.5 MB]
Get:4 http://kali.download/kali kali-rolling/contrib amd64 Packages [122 kB]
Get:5 http://kali.download/kali kali-rolling/contrib amd64 Contents (deb) [247 kB]
Get:6 http://kali.download/kali kali-rolling/non-free amd64 Packages [194 kB]
Get:7 http://kali.download/kali kali-rolling/non-free amd64 Contents (deb) [902 kB]
Get:8 http://kali.download/kali kali-rolling/non-free-firmware amd64 Packages [33.0 kB]
Fetched 21.1 MB in 24s (883 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
1200 packages can be upgraded. Run 'apt list --upgradable' to see them.

(kali@kali)-[~]
$ clear
  
```

Before



After

Program: 3

AIM: Implement the commands that is used for Creating and Manipulating files: cat, cp, mv, rm, ls and its options, touch and their options, which is, where is, what is.

Theory:

- **cat (Concatenate):** The cat command in Linux concatenates and displays the contents of files. It is commonly used to view text files, concatenate multiple files, or create new files by combining existing ones. The output can be redirected or appended to another file.
- **cp (Copy):** It copies files or directories from one location to another. It preserves file attributes and can recursively copy entire directory structures. Commonly employed for creating backups, duplicating files, or transferring data within a filesystem.
- **mv (Move):** The mv command moves files or directories from one location to another or renames them. It facilitates efficient file organization and renaming while preserving file attributes, making it a versatile tool for file manipulation.
- **rm (Remove):** It removes files or directories from the filesystem. It is used to delete files permanently, and with the -r option, it can recursively delete directories. Users must exercise caution as deleted data is typically unrecoverable.
- **ls (List):** It lists the contents of a directory. Commonly employed with various options like -l for detailed information, -a to display hidden files, and -h for human-readable file sizes. It aids users in navigating and exploring directory structures.
- **touch:** The touch command creates empty files or updates the timestamp of existing files. It is useful for quickly creating placeholders or modifying timestamps to reflect current time. With no arguments, it creates an empty file or updates the access and modification times.
- **whereis:** It locates the binary, source, and manual page files for a specified command. It provides the absolute paths to these files, aiding users in identifying the locations of executable programs, their source code, and associated documentation.
- **which:** The which command identifies the full path of an executable in the user's PATH environment. It helps users determine the location of a specific command or program, facilitating command-line execution and troubleshooting.
- **whatis:** It provides a brief description of a specified command based on its manual page entry. It is a quick reference tool for obtaining concise information about the purpose and functionality of a command, aiding users in command comprehension..

Procedure:

1. Open the terminal in kali.



2. Enter the “**cat**” in terminal for display the one or more files.

Command : cat

```
kali@kali: ~
File Actions Edit View Help
(kali@kali)-[~]
$ cat a.txt
hello!
world!
hello world
```

3. Enter the “**cp**” in terminal for Copies files or directories from one location to another.

Command : cp print.c print1.c

```
kali@kali: ~
File Actions Edit View Help
(kali@kali)-[~]
$ cp print.c print1.c

(kali@kali)-[~]
$ ls
1.c      Documents  Hello      Music      Public      Templates
abc.txt  Downloads  hello.c    pattern.c  result      text.txt
a.out    file_{1..10}.txt Hello.c    Pictures   sum.c       txt_file
a.txt    headers.h  main.c    print1.c  sum.c.save  Videos
Desktop  hello      Makefile  print.c   TBomb
```

4. Enter the “**mv**” in terminal for Moves files or directories from one location to another, and can also be used to rename files.

Command : mv header.h headers.h

```
kali@kali: ~
File Actions Edit View Help
(kali@kali)-[~]
$ ls
1.c      a.out  Desktop  Downloads  header.h  Hello  Hello.c  makefi
abc.txt  a.txt  Documents file_{1..10}.txt hello    hello.c  main.c  Makefi

(kali@kali)-[~]
$ mv header.h headers.h

(kali@kali)-[~]
$ ls
1.c      a.out  Desktop  Downloads  headers.h  Hello  Hello.c  makef
abc.txt  a.txt  Documents file_{1..10}.txt hello    hello.c  main.c  Makef

(kali@kali)-[~]
$
```

5. Enter the “**rm**” in terminal for Deletes files or directories.

Command : rm t1.txt

```
(kali@kali)-[~]
$ rm t1.txt

(kali@kali)-[~]
$ ls
1.c      Documents      Hello      Music      Public      Templates
abc.txt  Downloads      hello.c    pattern.c  result      text.txt
a.out    file_{1..10}.txt Hello.c    Pictures   sum.c       txt_file
a.txt    headers.h      main.c    print1.c  sum.c.save  Videos
Desktop  hello          Makefile  print.c   TBomb
```

6. Enter the “**ls**” in terminal for Lists files and directories in the current directory.

Command : ls

```
(kali@kali)-[~]
$ ls
1.c      Documents      Hello      Music      Public      TBomb
abc.txt  Downloads      hello.c    pattern.c  result      Templates
a.out    file_{1..10}.txt Hello.c    Pictures   sum.c       text.txt
a.txt    headers.h      main.c    print1.c  sum.c.save  txt_file
Desktop  hello          Makefile  print.c   t1.txt     Videos
```

7. Enter the “**touch**” in terminal for Creates an empty file or updates the timestamp of an existing file.

Command : touch t1.txt

```
kali@kali: ~
File Actions Edit View Help

(kali@kali)-[~]
$ touch t1.txt

(kali@kali)-[~]
$ ls
1.c      Documents      Hello      Music      Public      TBomb
abc.txt  Downloads      hello.c    pattern.c  result      Templates
a.out    file_{1..10}.txt Hello.c    Pictures   sum.c       text.txt
a.txt    headers.h      main.c    print1.c  sum.c.save  txt_file
Desktop  hello          Makefile  print.c   t1.txt     Videos
```

8. Enter the “**which**” in terminal for Displays the path of the executable file.

Command : which python

```
(snoopy@juily)-[~]  
$ which python  
/usr/bin/python
```

9. Enter the “**whereis**” in terminal for Locates the binary, source, and manual page files.

Command : whereis python

```
(snoopy@juily)-[~]  
$ whatis python  
python (1)          - an interpreted, interactive, object-oriented programming language
```

10. Enter the “**whatis**” in terminal for Provides a brief description on file.

Command : whatis python

```
(snoopy@juily)-[~]  
$ whereis python  
python: /usr/bin/python /usr/share/python /usr/share/man/man1/python.1.gz
```


Program: 4

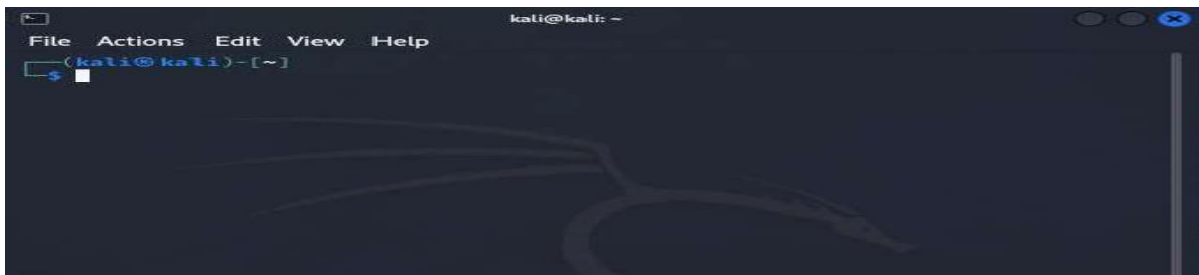
AIM: Implement Directory oriented commands: cd, pwd, mkdir, rmdir, Comparing Files using diff, cmp, comm.

Theory:

- **cd (Change Directory):** The 'cd' command is used to navigate between directories in a file system. By specifying the desired directory as an argument, users can move to different locations in the file hierarchy. For example, 'cd Documents' changes the current directory to "Documents."
- **pwd (Print Working Directory):** It displays the absolute path of the current working directory. It helps users identify their location within the file system. Simply entering 'pwd' in the terminal provides the full path, aiding in orientation.
- **mkdir (Make Directory):** It creates a new directory with the specified name. For instance, 'mkdir NewFolder' generates a folder named "NewFolder" in the current directory.
- **rmdir (Remove Directory):** It removes an empty directory. It is utilized as 'rmdir OldFolder' to delete the directory named "OldFolder," but it only works if the folder is empty.
- **diff (File Comparison):** It compares the content of two files line by line and highlights the differences. It's employed as 'diff File1.txt File2.txt,' revealing discrepancies between the two text files.
- **cmp (Compare):** It compares two files byte by byte and indicates the first differing byte. If no differences are found, it remains silent. To use, type 'cmp File1.txt File2.txt.'
- **comm (Compare and Identify Common Lines):** It compares two sorted files line by line. It displays lines unique to each file and common lines. With 'comm File1.txt File2.txt,' users can analyze the differences and similarities between the two text files.

Procedure:

1. Open the terminal in kali.



2. Enter the “cd” in terminal for change the current working directory.

Command : cd


```
(kali@kali)-[~]
$ cd Videos

(kali@kali)-[~/Videos]
$ pwd
/home/kali/Videos
```

3. Enter the “**pwd**” in terminal for change the current working directory.

Command : pwd

```
kali@kali: ~
File Actions Edit View Help

(kali@kali)-[~]
$ pwd
/home/kali
```

4. Enter the “**mkdir**” in terminal for change the current working directory.

Command : mkdir practice

```
(kali@kali)-[~]
$ mkdir a1.txt

(kali@kali)-[~]
$ ls
1.c      Desktop  hello    Makefile  print.c  TBomb
a1.txt   Documents Hello     Music     Public   Templates
abc.txt  Downloads hello.c   pattern.c result    text.txt
a.out    file_{1..10}.txt Hello.c  Pictures  sum.c    txt_file
a.txt    headers.h  main.c   print1.c  sum.c.save Videos

(kali@kali)-[~]
$
```

5. Enter the “**rmdir**” in terminal for change the current working directory.

Command : rmdir practice

```
(kali@kali)-[~]
$ rmdir a1.txt

(kali@kali)-[~]
$ ls
1.c      Documents  Hello      Music      Public     Templates
abc.txt  Downloads  hello.c    pattern.c  result     text.txt
a.out    file_{1..10}.txt Hello.c    Pictures   sum.c      txt_file
a.txt    headers.h  main.c     print1.c   sum.c.save Videos
Desktop  hello      Makefile   print.c    TBomb
```

6. Enter the “**diff**” in terminal for change the current working directory.

Command : diff -c print.c print1.c

```
File  Actions  Edit  View  Help

(kali@kali)-[~]
$ diff -c print.c print1.c
*** print.c      2024-02-05 00:55:47.467274473 -0500
--- print1.c     2024-02-05 01:21:13.451416325 -0500
*****
*** 1,11 ****
! #include<stdio.h>
! #include<stdlib.h>
! int main(){
!     system("ls");
!     system("echo 'some content'> text.txt");
!     if(system("abc")!=0){
!         printf("error executing");
!     }
!     return 0;
! }

--- 1,14 ---
! #include<stdio.h>
! #include<unistd.h>

+ int main()
+ {
+     int pid;
+     pid = fork();
+     if(pid == 0){
+         printf("Process id %d \n",getpid());
+         printf("Calling Process id %d \n",getppid());
+     }
+     return 0;
+ }
```

7. Enter the “**cmp**” in terminal for change the current working directory.

Command : `cmp -c print.c sum.c`

```
(kali㉿kali)-[~]  
$ cmp -c print.c sum.c  
print.c sum.c differ: byte 27, line 2 is 74 < 42 "
```

8. Enter the “**com**” in terminal for change the current working directory.

Command : `com --c print.c sum.c`

```
(kali㉿kali)-[~]  
$ comm --c print.c sum.c  
#include<stdio.h>  
comm: file 2 is not in sorted order
```

```
(kali㉿kali)-[~]  
$
```

Program: 5

AIM: Write a program and execute the same to demonstrate how to use terminal commands in C program (using system () function)

Theory: The system () function in C provides a way to execute shell commands from within a C program. The system () function takes a string command as an argument and executes it as if it were typed in the terminal. The return value represents the termination status of the command. A return value of -1 indicates an error in command execution.

The **system ()** function in C allows for the execution of terminal commands directly from a C program. By providing a string containing the desired command as an argument to **system ()**, the program can interact with the underlying operating system. This functionality is useful for automating tasks, executing system commands, and integrating terminal functionalities into C applications.

However, caution should be exercised when using **system ()** to prevent security vulnerabilities, as it directly runs commands without sanitization. It is essential to validate and sanitize input to avoid potential security risks associated with executing arbitrary commands from user input.

Procedure:

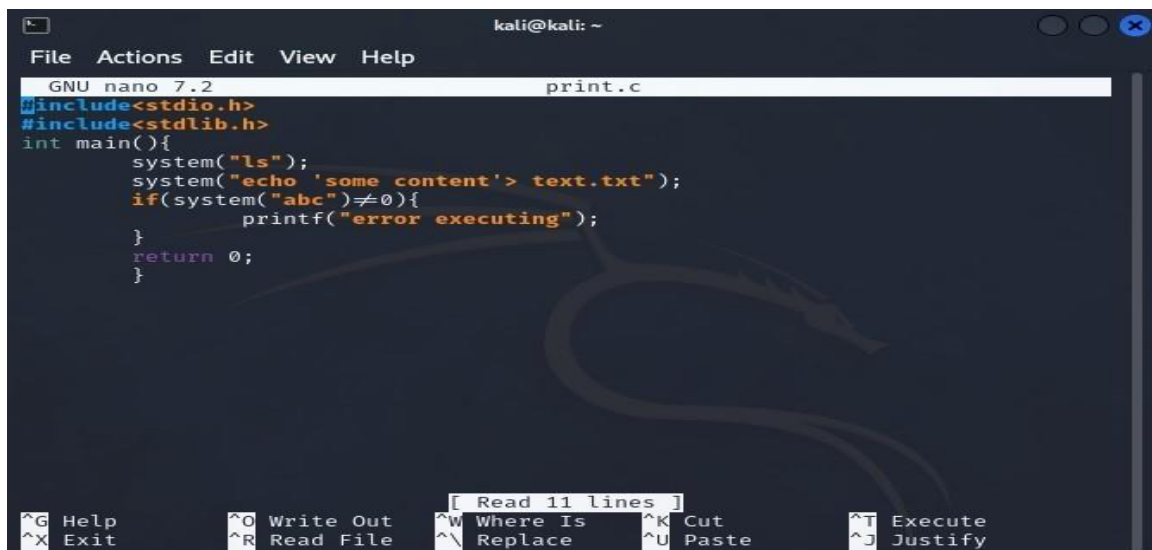
1. Open the terminal in kali. Enter the command “**nano print.c**” to open the file text editor where we will create a basic c pro-gram which we have named as “**print.c**”.

Command : nano print.c



```
(kali@kali)-[~]
$ nano print.c
```

2. You'll enter the text file editor. Type a c program to excute the terminal commands using system function under 'stdlib.h'.



```
kali@kali: ~
File Actions Edit View Help
GNU nano 7.2 print.c
#include<stdio.h>
#include<stdlib.h>
int main(){
    system("ls");
    system("echo 'some content'> text.txt");
    if(system("abc")!=0){
        printf("error executing");
    }
    return 0;
}
```

^G Help ^O Write Out [Read 11 lines] ^K Cut ^T Execute
^X Exit ^R Read File ^W Where Is ^U Paste ^J Justify



Save the file with your code. If using Nano editor, you can press `Ctrl + X`, shift Y and press “Enter button” to exit editor.

3. Use the GCC compiler to compile the C Program. Enter the command “**gcc print.c**” to compile the c program OR Enter the command “**gcc print.c -o result**” to compile the c program and save the output of the c program in result(result is the output file name).

```
(kali@kali)-[~]  
$ gcc print.c -o result
```

4. To generate the output of the c program, type “**./result**” in the terminal.

Command : ./result

```
(kali@kali)-[~]  
$ ./result  
1.c      Documents      Hello      Music      Public      Templates  
abc.txt   Downloads      hello.c    pattern.c   result      text.txt  
a.out     file_{1..10}.txt  Hello.c    Pictures    sum.c       txt_file  
a.txt     headers.h        main.c     print1.c    sum.c.save  Videos  
Desktop   hello            Makefile   print.c     TBomb  
sh: 1: abc: not found  
error executing  
  
(kali@kali)-[~]  
$
```

Program: 6

AIM: Write a program to implement process concepts using C language by printing process Id.

Theory: In C programming, process management is facilitated by system calls provided by the operating system. The fork() system call is crucial for creating a new process. The basic theory involves the following concepts:

- **fork() System Call:** The fork() system call creates a new process by duplicating the existing process. After the fork, there are two identical processes (parent and child) with different process IDs.
- **Process ID (PID):** Each process in a Unix-like operating system is identified by a unique numerical identifier called the Process ID (PID). The parent and child processes have distinct PIDs.
- The getpid() function retrieves the PID of the current process , and the getppid() function retrieves the PID of the parent process.

Procedure:

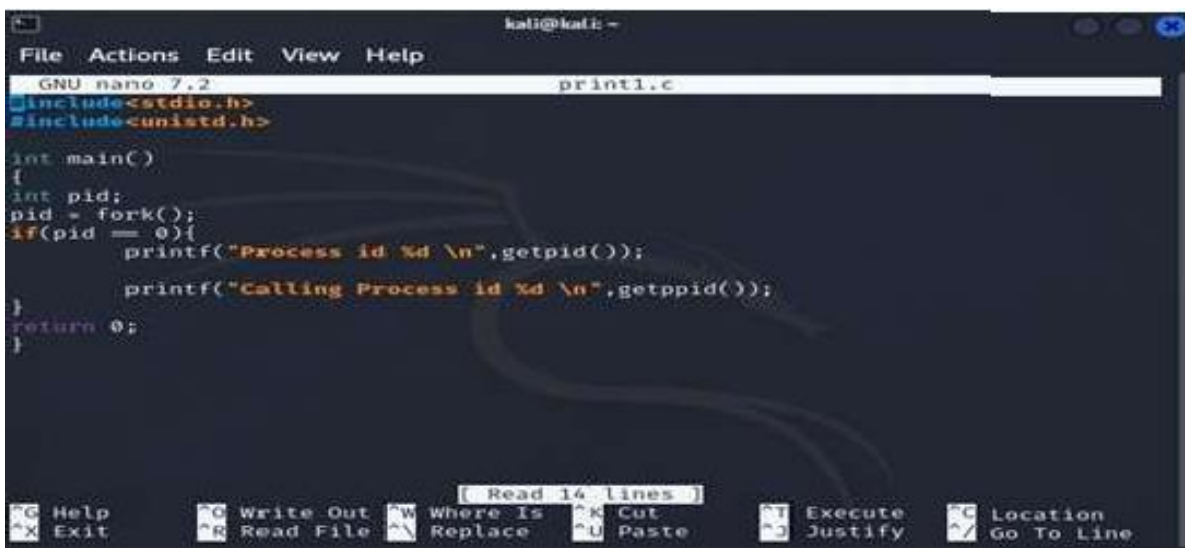
1. Open the terminal in kali. Enter the command “**nano print1.c**” to open the file text editor where we will create a basic c program which we have named as “**print1.c**”.

Command: nano print1.c



```
(kali@kali)-[~]
$ nano print1.c
```

2. You'll enter the text file editor. Type a c program to execute the terminal commands using system function under 'stdlib.h'.



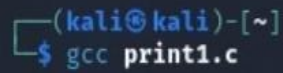
```
GNU nano 7.2 print1.c
#include<stdio.h>
#include<unistd.h>

int main()
{
    int pid;
    pid = fork();
    if(pid == 0){
        printf("Process id %d \n",getpid());
        printf("Calling Process id %d \n",getppid());
    }
    return 0;
}
```

Save the file with your code. If using Nano editor, you can press `Ctrl + X`, shift Y and press

“Enter button” to exit editor.

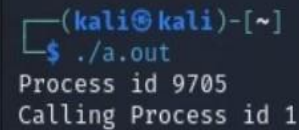
3. Use the GCC compiler to compile the C Program. Enter the command **“gcc print1.c”** to compile the c program OR Enter the command **“gcc print1.c -o result”** to compile the c program and save the output of the c program in result(result is the output file name).



```
(kali㉿kali)-[~]  
$ gcc print1.c
```

4. To generate the output of the c program, type **“./result”** in the terminal.

Command : ./result



```
(kali㉿kali)-[~]  
$ ./a.out  
Process id 9705  
Calling Process id 1
```


Program: 7

AIM: Write a program to create and execute process using fork() and exec() system calls.

Theory:

fork():

- The fork() system call is used to create a new process, which is called the child process, from the calling process, which is called the parent process.
- When fork() is called, a new process is created by duplicating the calling process. Both the parent and child processes then continue execution from the point where the fork() call was made.
- The child process receives a copy of the parent's address space, including all memory segments (code, data, stack).
- After fork() returns, it can be used to differentiate between the parent and child processes. The return value is different for each process: it returns 0 in the child process and the process ID (PID) of the child in the parent process.

exec():

- The exec() system call is used to execute a new program in the current process space. It replaces the current process image with a new one.
- There are several variants of the exec() system call, such as execl(), execv(), execl(), execve(), etc., which provide different ways to specify the arguments and environment for the new program.
- When exec() is called, the operating system loads the new program into the current process's memory space, replacing the previous program.
- After exec() successfully loads the new program, the control flow starts from the beginning of the new program's main() function.

Combining fork() and exec() allows for creating new processes and then replacing their memory image with a new program. This is often used in shell scripting and in creating new processes in Unix-based systems.

Here's a typical sequence of steps when using fork() and exec() together:

The parent process calls fork(), creating a new child process.

The child process, after successful creation, calls one of the exec() functions to load a new program into its memory space.

The new program starts executing in the child process, replacing the previous program.

Meanwhile, the parent process may continue executing its own code or may wait for the child process to complete using wait() or waitpid().

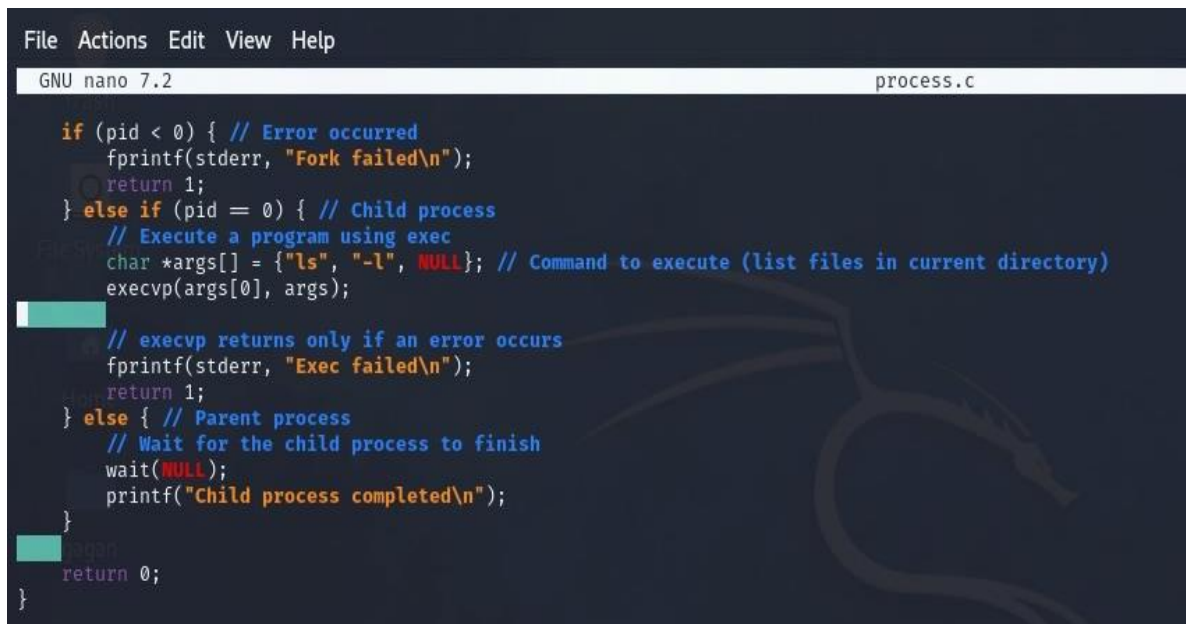
Procedure:

1. Open the terminal in kali. Enter the command “**nano process.c**” to open the file text editor where we will create a basic c program which we have named as “**process.c**”.

command: nano process.c

```
$ nano process.c
```

2. You'll enter the text file editor. Type a c program to execute the terminal commands using system function under 'stdlib.h'.



```
File Actions Edit View Help
GNU nano 7.2 process.c

if (pid < 0) { // Error occurred
    fprintf(stderr, "Fork failed\n");
    return 1;
} else if (pid == 0) { // Child process
    // Execute a program using exec
    char *args[] = {"ls", "-l", NULL}; // Command to execute (list files in current directory)
    execvp(args[0], args);

    // execvp returns only if an error occurs
    fprintf(stderr, "Exec failed\n");
    return 1;
} else { // Parent process
    // Wait for the child process to finish
    wait(NULL);
    printf("Child process completed\n");
}

return 0;
}
```

Save the file with your code. If using Nano editor, you can press `Ctrl + X`, shift Y and press “Enter button” to exit editor.

3. Use the GCC compiler to compile the C Program. Enter the command “**gcc process.c**” to compile the c program OR Enter the command “**gcc process.c -o result**” to compile the c program and save the output of the c program in result(result is the output file name).

command : gcc process.c -o result

```
$ gcc process.c -o result
```

4. To generate the output of the c program, type “./result” in the terminal.

Command : ./result

```
$ ./result
total 184
-rw-r--r-- 1 Gagan ryan 0 Jan 22 01:01 1.c
drwxr-xr-x 3 Gagan ryan 4096 Feb 12 00:40 a
drwxr-xr-x 2 Gagan ryan 4096 Feb 12 00:38 'a1.txt*4'
-rwxr--r-- 1 Gagan ryan 28 Jan 15 00:14 abc.txt
-rwxr-xr-x 1 Gagan ryan 16096 Mar 4 00:16 a.out
-rwxr--r-- 1 Gagan ryan 17 Feb 12 00:33 a.txt
drwxr-xr-x 2 Gagan ryan 4096 Feb 12 00:38 b
drwxr-xr-x 2 Gagan ryan 4096 Feb 12 00:38 c
drwxr-xr-x 3 Gagan ryan 4096 Jan 9 01:59 Desktop
drwxr-xr-x 2 Gagan ryan 4096 Jan 8 14:04 Documents
drwxr-xr-x 2 Gagan ryan 4096 Jan 8 14:04 Downloads
-rw-r--r-- 1 Gagan ryan 12 Feb 11 15:53 example.txt
-rw-r--r-- 1 Gagan ryan 0 Feb 6 02:55 file_{1..10}.txt
-rw-r--r-- 1 Gagan ryan 1865 Mar 4 00:16 first.c
-rw-r--r-- 1 Gagan ryan 27 Feb 5 00:39 headers.h
-rwxr-xr-x 1 Gagan ryan 15952 Jan 29 00:45 hello
-rwxr-xr-x 1 Gagan ryan 15952 Feb 2 00:57 Hello
-rw-r--r-- 1 Gagan ryan 57 Feb 11 12:25 hello.c
-rw-r--r-- 1 Gagan ryan 71 Feb 2 00:41 Hello.c
-rw-r--r-- 1 Gagan ryan 197 Feb 5 00:40 main.c
-rw-r--r-- 1 Gagan ryan 61 Feb 11 12:32 Makefile
drwxr-xr-x 2 Gagan ryan 4096 Jan 8 14:04 Music
-rw-r--r-- 1 Gagan ryan 207 Jan 22 01:04 pattern.c
drwxr-xr-x 3 Gagan ryan 4096 Feb 1 09:55 Pictures
-rw-r--r-- 1 Gagan ryan 206 Feb 11 15:10 print1.c
-rw-r--r-- 1 Gagan ryan 176 Feb 5 00:55 print.c
-rw-r--r-- 1 Gagan ryan 763 Mar 18 08:32 process.c
drwxr-xr-x 2 Gagan ryan 4096 Jan 8 14:04 Public
-rwxr-xr-x 1 Gagan ryan 16200 Mar 18 08:32 result
-rw-r--r-- 1 Gagan ryan 210 Feb 5 00:40 sum.c
-rw-r--r-- 1 Gagan ryan 273 Jan 30 04:14 sum.c.save
drwxr-xr-x 5 Gagan ryan 4096 Jan 15 01:17 TBomb
```

Program: 8

AIM: Write a C program to implement FCFS (First Come First Serve) and SJF(Shortest Job First) scheduling algorithms.

Theory:

Scheduling algorithms manage the allocation of CPU time among processes. FCFS executes processes in the order of arrival, ensuring fairness but potentially leading to longer waiting times. SJF prioritizes shorter processes to minimize waiting times, making it optimal for reducing average waiting time. FCFS is simple to implement but may result in inefficient resource utilization. SJF requires knowledge of burst times and may lead to starvation for longer processes. FCFS is non-preemptive, while SJF can be preemptive or non-preemptive. Preemptive SJF, known as SRTF, can further reduce waiting times by preempting longer processes. Both algorithms play critical roles in optimizing system performance and responsiveness. Understanding their characteristics helps in designing efficient scheduling systems. FCFS is suitable for batch processing, while SJF is beneficial for interactive systems. Analyzing the trade-offs between fairness and efficiency is essential when selecting a scheduling algorithm. Both FCFS and SJF contribute to the effective management of processes in operating systems.

Procedure:

(A) FCFS (First Come First Serve) scheduling algorithms.

1. Open the terminal in kali. Enter the command “**nano FCFS.c**” to open the file text editor where we will create a basic c program which we have named as “**FCFS.c**”.

command: nano FCFS.c



2. You'll enter the text file editor. Type a c program to execute the terminal commands using system function under 'stdlib.h'.

Save the file with your code. If using Nano editor, you can press `Ctrl + X`, shift Y and press “Enter button” to exit editor.

```
GNU nano 7.2 FCFS
#include <stdio.h>
#include <stdlib.h>

// Process structure
typedef struct {
    int id;
    int arrival_time;
    int burst_time;
} Process;

// Function to perform FCFS scheduling
void fcfs(Process processes[], int n) {
    int total_waiting_time = 0;
    int total_turnaround_time = 0;
    int current_time = 0;

    printf("FCFS Scheduling:\n");
    printf("Process\t Arrival Time\t Burst Time\t Waiting Time\t Turnaround Time\n");

    for (int i = 0; i < n; i++) {
        int waiting_time = current_time - processes[i].arrival_time;
        if (waiting_time < 0)
            waiting_time = 0;

        int turnaround_time = waiting_time + processes[i].burst_time;

        printf("%d\t\t %d\t\t %d\t\t %d\t\t %d\n",
            processes[i].id, processes[i].arrival_time, processes[i].burst_time,
            waiting_time, turnaround_time);

        total_waiting_time += waiting_time;
        total_turnaround_time += turnaround_time;
    }
}
```

```
        current_time += processes[i].burst_time;
    }

    printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / n);
    printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);
}

int main() {
    int n;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    Process *processes = (Process *)malloc(n * sizeof(Process));

    printf("Enter arrival time and burst time for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d: ", i + 1);
        scanf("%d %d", &processes[i].arrival_time, &processes[i].burst_time);
        processes[i].id = i + 1;
    }

    // Perform FCFS scheduling
    fcfs(processes, n);

    free(processes);

    return 0;
}
```

3. Use the GCC compiler to compile the C Program . Enter the command **“gcc FCFS.c”** to compile the c program OR Enter the command **“gcc FCFS.c -o result”** to compile the c program and save the output of the c program in result(result is the output file name).
command : gcc FCFS.c -o result



```
$ gcc FCFS.c -o result
```

4. To generate the output of the c program, type “./result” in the terminal.

Command : ./result

```
$ ./result
Enter the number of processes: 3
Enter arrival time and burst time for each process:
Process 1: 1
2
Process 2: 2
3
Process 3: 3
4
FCFS Scheduling:
Process  Arrival Time    Burst Time    Waiting Time    Turnaround Time
1           1           2           0             2
2           2           3           0             3
3           3           4           2             6
Average Waiting Time: 0.67
Average Turnaround Time: 3.67
```

(B) SJF(Shortest Job First) scheduling algorithms.

1. Open the terminal in kali. Enter the command “**nano SJF.c**” to open the file text editor where we will create a basic c program which we have named as “**SJF.c**”.

command: nano SJF.c

```
$ nano SJF.c
```

2. You'll enter the text file editor. Type a c program to execute the terminal commands using system function under 'stdlib.h'.


```
GNU nano 7.2 SJF.c
#include <stdio.h>
#include <stdlib.h>

// Process structure
typedef struct {
    int id;
    int arrival_time;
    int burst_time;
} Process;

// Function to perform SJF scheduling
void sjf(Process processes[], int n) {
    int total_waiting_time = 0;
    int total_turnaround_time = 0;
    int current_time = 0;

    printf("\nSJF Scheduling:\n");
    printf("Process\t Arrival Time\t Burst Time\t Waiting Time\t Turnaround Time\n");

    // Sort processes based on burst time (shortest job first)
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (processes[j].burst_time > processes[j + 1].burst_time) {
                // Swap processes
                Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }

    for (int i = 0; i < n; i++) {
        int waiting_time = current_time - processes[i].arrival_time;
        if (waiting_time < 0)
            waiting_time = 0;

        int turnaround_time = waiting_time + processes[i].burst_time;

        printf("%d\t\t %d\t\t %d\t\t %d\t\t %d\n",
            processes[i].id, processes[i].arrival_time, processes[i].burst_time,
            waiting_time, turnaround_time);

        total_waiting_time += waiting_time;
        total_turnaround_time += turnaround_time;
        current_time += processes[i].burst_time;
    }

    printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / n);
    printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);
}
```

```
int main() {
    int n;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    Process *processes = (Process *)malloc(n * sizeof(Process));

    printf("Enter arrival time and burst time for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d: ", i + 1);
        scanf("%d %d", &processes[i].arrival_time, &processes[i].burst_time);
        processes[i].id = i + 1;
    }

    sjf(processes, n);
}
```



```
printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / n);
printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);
}

int main() {
    int n;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    Process *processes = (Process *)malloc(n * sizeof(Process));

    printf("Enter arrival time and burst time for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d: ", i + 1);
        scanf("%d %d", &processes[i].arrival_time, &processes[i].burst_time);
        processes[i].id = i + 1;
    }

    // Perform SJF scheduling
    sjf(processes, n);

    free(processes);

    return 0;
}
```

Save the file with your code. If using Nano editor, you can press `Ctrl + X`, shift Y and press "Enter button" to exit editor.

3. Use the GCC compiler to compile the C Program . Enter the command **"gcc SJF.c"** to compile the c program OR Enter the command **"gcc SJF.c -o result"** to compile the c program and save the output of the c program in result(result is the output file name).

command : gcc SJF.c -o result

```
$ gcc SJF.c -o result
```

4. To generate the output of the c program, type **"./result"** in the terminal.

Command : ./result

```
$ ./result
Enter the number of processes: 3
Enter arrival time and burst time for each process:
Process 1: 0
1
Process 2: 0
9
Process 3: 0
3

SJF Scheduling:
Process  Arrival Time    Burst Time    Waiting Time    Turnaround Time
1             0             1             0              1
3             0             3             1              4
2             0             9             4             13
Average Waiting Time: 1.67
Average Turnaround Time: 6.00
```