Ex. No.: 1 **Booting Process of Linux Operating System**
Date: 11/08/2022

AIM:

To Demonstrate the 6 Steps of Linux Booting Process in detail.

Procedure:

- An operating system is the low-level software that manages resources, controls peripherals, and provides basic services to software. In Linux,there are 6 distinct stages in the booting process.

  STEP-1

- BIOS stands for Basic Input/Output System. In simple terms, the BIOS loads and executes the Master Boot Record (MBR) boot loader.
- When you first turn on your computer, the BIOS first performs some integrity checks of the HDD or SSD.
- Then, the BIOS searches for, loads, and executes the boot loader program, which can be found in the Master Boot Record (MBR).
- Once the boot loader program is detected, it's then loaded into memory and the BIOS gives control of the system to it.

  STEP-2

- MBR stands for Master Boot Record, and is responsible for loading and executing the GRUB boot loader.
- The MBR is located in the 1st sector of the bootable disk, which is typically /dev/hda depending on your hardware. The MBR also contains information about GRUB, or LILO in very old systems.

  STEP-3

- Sometimes called GNU GRUB, which is short for GNU Grand Unified Bootloader, is the typical boot loader for most modern Linux systems.

- The GRUB splash screen is often the first thing you see when you boot your computer.
- If you have multiple kernel images installed, you can use your keyboard to select the one you want your system to boot with. By default, the latest kernel image is selected.
- The splash screen will wait a few seconds for you to select and option. If you don't, it will load the default kernel image.
- In many systems you can find the GRUB configuration file at /boot/grub/grub.conf. Here's an example of a simple grub.conf file:

```
#boot=/dev/sda
default=0
timeout=5
splashimage=(hd0,0)/boot/grub/splash.xpm.gz
hiddenmenu
title CentOS (2.6.18-194.el5PAE)
   root (hd0,0)
   kernel /boot/vmlinuz-2.6.18-194.el5PAE ro root=LABEL=/
initrd /boot/initrd-2.6.18-194.el5PAE.img
```

STEP-4

- The kernel is often referred to as the core of any operating system, Linux included. It has complete control over everything in your system.
- In this stage of the boot process, the kernel that was selected by GRUB first mounts the root file system that's specified in the grub.conf file. Then it executes the /sbin/init.
- You can confirm this with its process id (PID), which should always be 1.The kernel then establishes a temporary root file system using Initial RAM Disk (initrd) until the real file system is mounted.

STEP-5

- At this point, your system executes runlevel programs. At one point it would look for an init file, usually found at /etc/inittab to decide the Linux run level.
- Modern Linux systems use systemd to choose a run level instead. According to TecMint, these are the available run levels:
- **Run level 0** is matched by **poweroff.target** (and **runlevel0.target** is a symbolic link to **poweroff.target**).
- **Run level 1** is matched by **rescue.target** (and **runlevel1.target** is a symbolic link to **rescue.target**).
- **Run level** 3 is emulated by **multi-user.target** (and **runlevel3.target** is a symbolic link to **multi-user.target**).
- **Run level 5** is emulated by **graphical.target** (and **runlevel5.target** is a symbolic link to **graphical.target**).
- **Run level 6** is emulated by **reboot.target** (and **runlevel6.target** is a symbolic link to **reboot.target**).
- **Emergency** is matched by **emergency.target**.systemd will then begin executing runlevel programs.

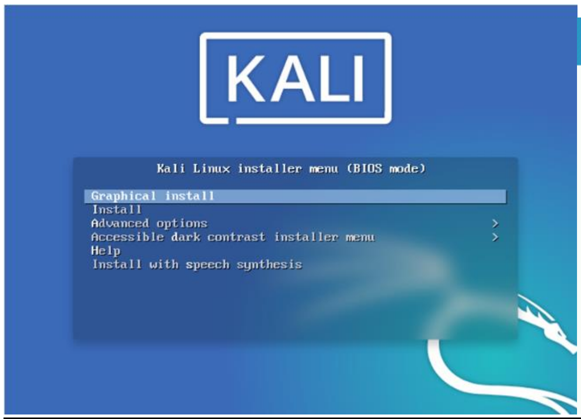
STEP-7

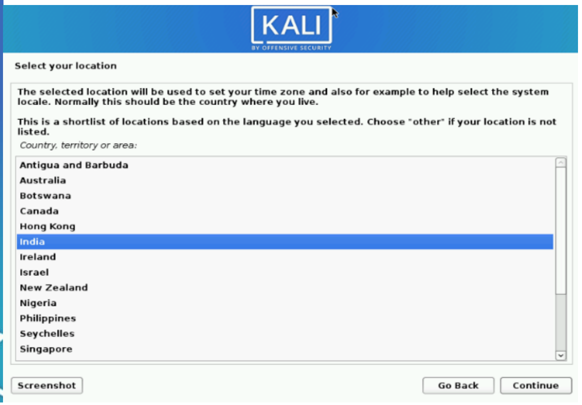
- Depending on which Linux distribution you have installed, you may be able to see different services getting started. For example, you might catch starting sendmail …. OK.
- These are known as runlevel programs, and are executed from different directories depending on your run level. Each of the 6 runlevels described above has its own directory:
  - Run level 0 – /etc/rc0.d/
  - Run level 1 – /etc/rc1.d/
  - Run level 2 – /etc/rc2.d/
  - Run level 3 – /etc/rc3.d/

- Run level 4 – /etc/rc4.d/
- Run level 5 – /etc/rc5.d/
- Run level 6 – /etc/rc6.d/
- Note that the exact location of these directories varies from distribution to distribution.

➢ If you look in the different run level directories, you'll find programs that start with either an "S" or "K" for startup and kill, respectively. Startup programs are executed during system startup, and kill programs during shutdown.

SCREENSHOTS:


KALI
Kali Linux installer menu (BIOS mode)
Graphical install
Install
Advanced options
Accessible dark contrast installer menu
Help
Install with speech synthesis



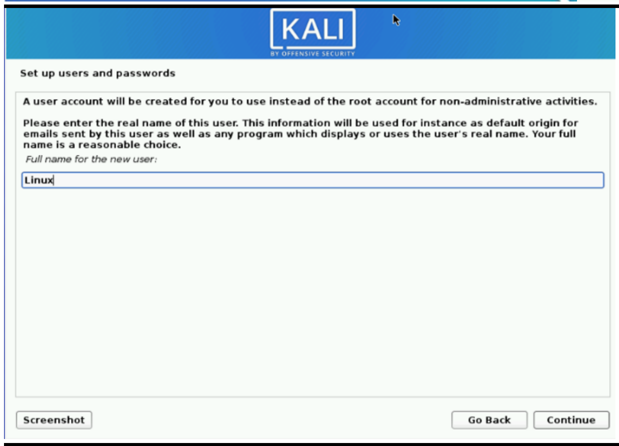
Select your location

The selected location will be used to set your time zone and also for example to help select the system locale. Normally this should be the country where you live.

This is a shortlist of locations based on the language you selected. Choose "other" if your location is not listed.

Country, territory or area:

Antigua and Barbuda
Australia
Botswana
Canada
Hong Kong
India
Ireland
Israel
New Zealand
Nigeria
Philippines
Seychelles
Singapore



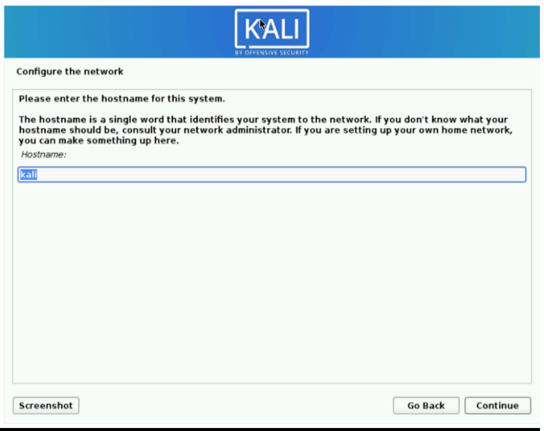
Set up users and passwords

A user account will be created for you to use instead of the root account for non-administrative activities.

Please enter the real name of this user. This information will be used for instance as default origin for emails sent by this user as well as any program which displays or uses the user's real name. Your full name is a reasonable choice.

Full name for the new user:

Linux



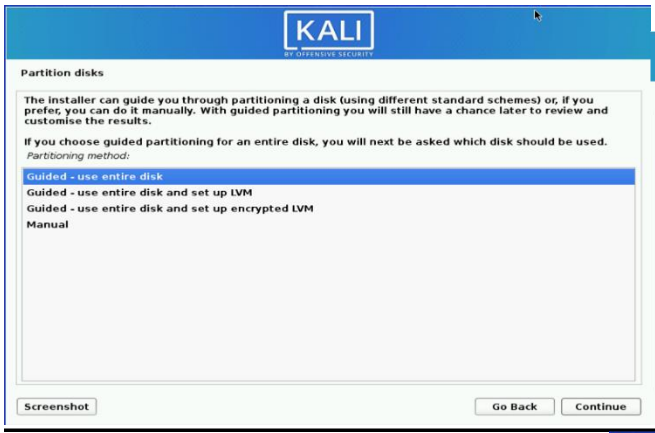
Configure the network

Please enter the hostname for this system.

The hostname is a single word that identifies your system to the network. If you don't know what your hostname should be, consult your network administrator. If you are setting up your own home network, you can make something up here.

Hostname:

kali



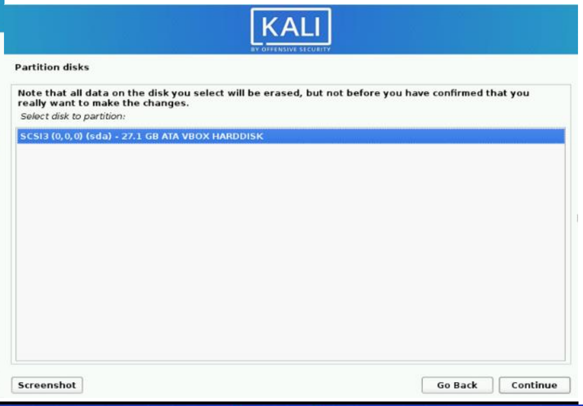
Partition disks

The installer can guide you through partitioning a disk (using different standard schemes) or, if you prefer, you can do it manually. With guided partitioning you will still have a chance later to review and customise the results.

If you choose guided partitioning for an entire disk, you will next be asked which disk should be used.

Partitioning method:

Guided - use entire disk
Guided - use entire disk and set up LVM
Guided - use entire disk and set up encrypted LVM
Manual

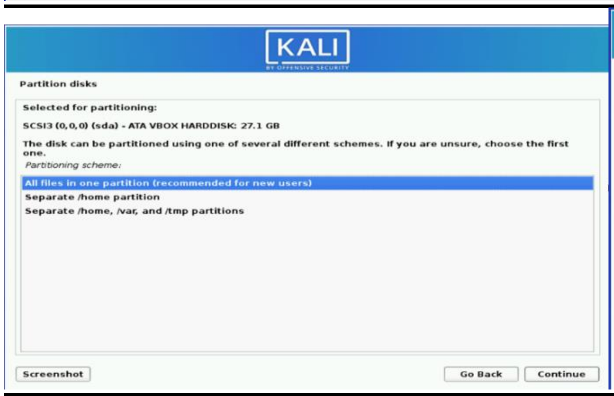


Partition disks

Note that all data on the disk you select will be erased, but not before you have confirmed that you really want to make the changes.

Select disk to partition:

SCSI3 (0,0,0) (sda) - 27.1 GB ATA VBOX HARDDISK



Partition disks

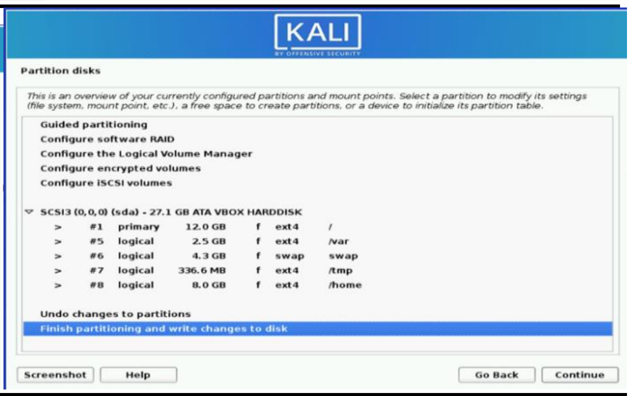
Selected for partitioning:

SCSI3 (0,0,0) (sda) - ATA VBOX HARDDISK: 27.1 GB

The disk can be partitioned using one of several different schemes. If you are unsure, choose the first one.

Partitioning scheme:

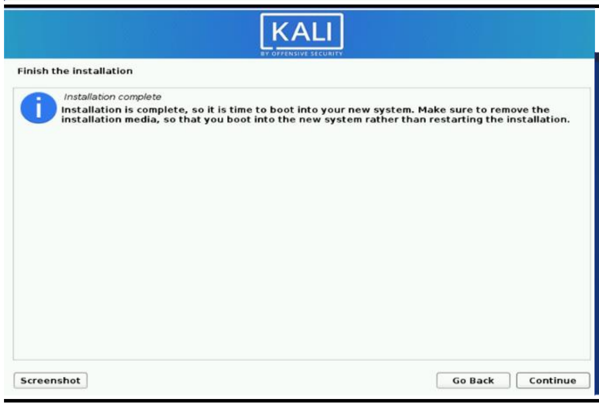
All files in one partition (recommended for new users)
Separate /home partition
Separate /home, /var, and /tmp partitions



Partition disks

This is an overview of your currently configured partitions and mount points. Select a partition to modify its settings (file system, mount point, etc.), a free space to create partitions, or a device to initialize its partition table.

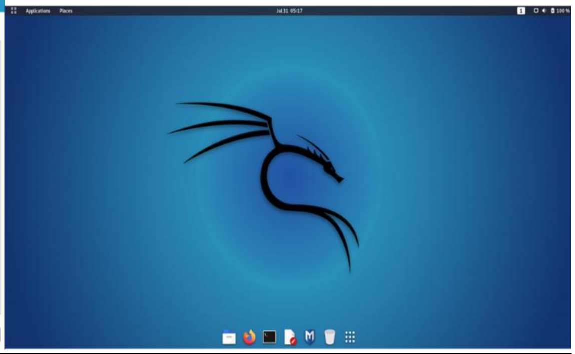
```
Guided partitioning
Configure software RAID
Configure the Logical Volume Manager
Configure encrypted volumes
Configure iSCSI volumes

SCSI3 (0,0,0) (sda) - 27.1 GB ATA VBOX HARDDISK
     >    #1   primary    12.0 GB    f   ext4    /
     >    #5   logical     2.5 GB    f   ext4    /var
     >    #6   logical     4.3 GB    f   swap    swap
     >    #7   logical   336.6 MB    f   ext4    /tmp
     >    #8   logical     8.0 GB    f   ext4    /home

Undo changes to partitions
Finish partitioning and write changes to disk
```



Finish the installation

Installation complete

Installation is complete, so it is time to boot into your new system. Make sure to remove the installation media, so that you boot into the new system rather than restarting the installation.



CONCLUSION:

Hence the above steps of Booting process of linux is understood clearly.

Ex. No: 2 **Basic Linux Commands**

Date : 11/08/2022

AIM:

To describe the Linux System Admin Commands in detail.

PROCEDURE:

STEP-1: A system administrator manages configuration, upkeep and reliable operations of computer operations. Sysadmin handles servers, and has to manage system performance and security without exceeding the budget to meet users' needs.

- A system administrator only deals with terminal interface and hence it is very important to learn and become master in commands to operate from terminal.
- **Some important commands for system administrators**

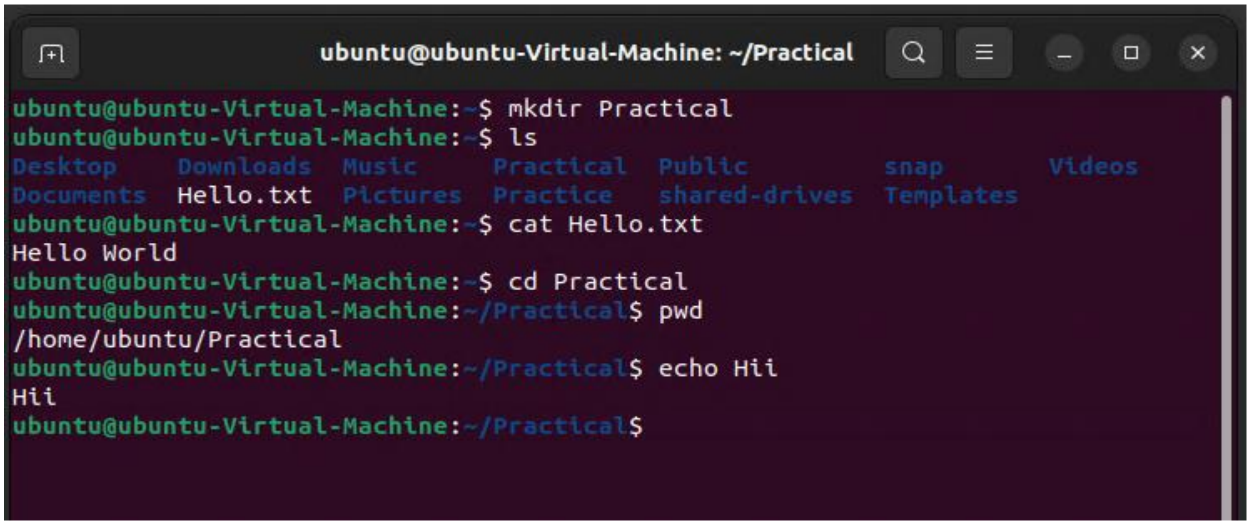
Command	**Function**
man	Display information about all commands
uptime	Show how long system is running
users	Show username who are currently logged in
service	Call and execute script
pkill	Kill a process

pmap	Memory map of a process
wget	Download file from network
ftp or sftp	Connect remote ftp host
free	Show memory status
top	Display processor activity of system
last	Display user's activity in the system
ps	Display about processes running on the system
Shutdown commands	Shutdown and reboot system
info	Display information about given command
env	Display environment variable for currently logged-in user
netstat	Display network status
arp	Check ethernet connectivity and IP address
df	Display filesystem information
du	Display usage

init	Allow to change server bootup
nano	A command line editor
nslookup	Check domain name and IP information
shred	Delete a file by over writing its content
cat	Display, copy or combine text files
pwd>	Print path of current working directory
locate	Finding files by name on system
chown	Change ownership of a file
>alias	To short a command
echo	Display text
cmp	Compare two files byte by byte
mount	Mount a filesystem
ifconfig	Display configuration
traceroute>	Trace existing network

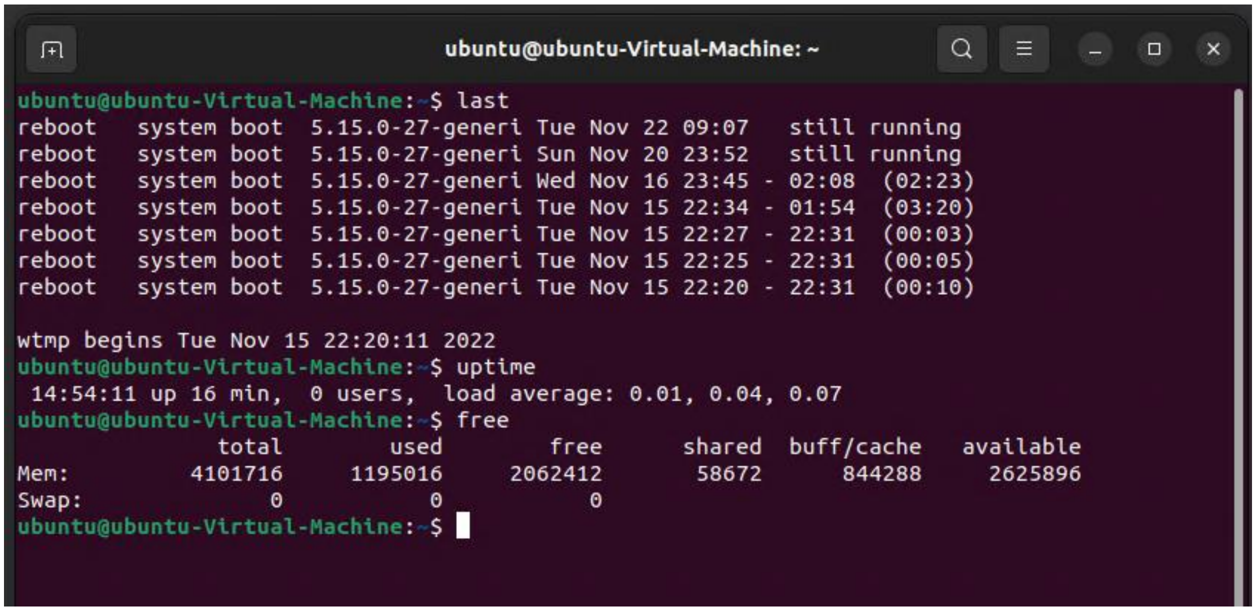
sudo	Run a command as a root user
route	List routing table for your server
ping	Check connection by sending packet test packet
find	Find location of files/directories
users	Show current logged in user
who	Same as w but doesn't show current process
ls	List all the files
tar	Compress directories
grep	Search for a string in a file
su	Switch from one to another user
awk	Search lines for a given pattern

OUTPUT:



ubuntu@ubuntu-Virtual-Machine: ~/Practical

```
ubuntu@ubuntu-Virtual-Machine:~$ mkdir Practical
ubuntu@ubuntu-Virtual-Machine:~$ ls
Desktop    Downloads  Music     Practical  Public         snap       Videos
Documents  Hello.txt  Pictures  Practice   shared-drives  Templates
ubuntu@ubuntu-Virtual-Machine:~$ cat Hello.txt
Hello World
ubuntu@ubuntu-Virtual-Machine:~$ cd Practical
ubuntu@ubuntu-Virtual-Machine:~/Practical$ pwd
/home/ubuntu/Practical
ubuntu@ubuntu-Virtual-Machine:~/Practical$ echo Hii
Hii
ubuntu@ubuntu-Virtual-Machine:~/Practical$
```



ubuntu@ubuntu-Virtual-Machine: ~

```
ubuntu@ubuntu-Virtual-Machine:~$ last
reboot   system boot  5.15.0-27-generi Tue Nov 22 09:07   still running
reboot   system boot  5.15.0-27-generi Sun Nov 20 23:52   still running
reboot   system boot  5.15.0-27-generi Wed Nov 16 23:45 - 02:08  (02:23)
reboot   system boot  5.15.0-27-generi Tue Nov 15 22:34 - 01:54  (03:20)
reboot   system boot  5.15.0-27-generi Tue Nov 15 22:27 - 22:31  (00:03)
reboot   system boot  5.15.0-27-generi Tue Nov 15 22:25 - 22:31  (00:05)
reboot   system boot  5.15.0-27-generi Tue Nov 15 22:20 - 22:31  (00:10)

wtmp begins Tue Nov 15 22:20:11 2022
ubuntu@ubuntu-Virtual-Machine:~$ uptime
 14:54:11 up 16 min,  0 users,  load average: 0.01, 0.04, 0.07
ubuntu@ubuntu-Virtual-Machine:~$ free
               total        used        free      shared  buff/cache   available
Mem:         4101716     1195016     2062412       58672      844288     2625896
Swap:              0           0           0
ubuntu@ubuntu-Virtual-Machine:~$ 
```

CONCLUSION:

Hence Linux System admin commands are executed Successfully.

Ex No.: 3 **Understanding the Concept of Linux File System**

Date: 26/08/2022

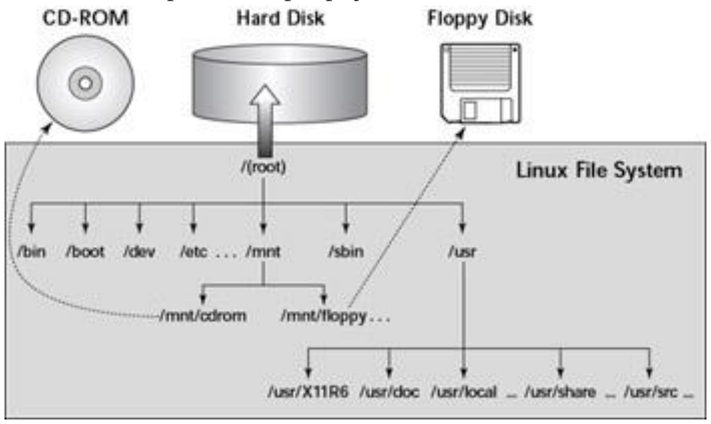
AIM:

To describe the Linux File System in detail.

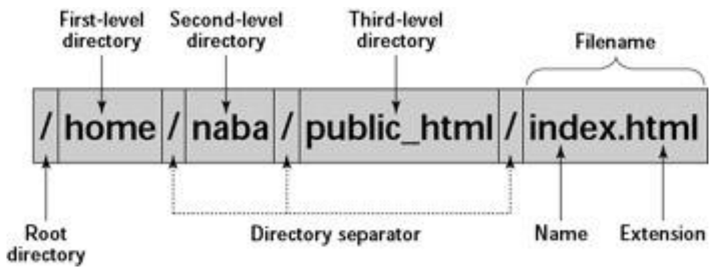
PROCEDURE:

STEP-1:

- Like any other operating system, Linux organizes information in files and directories. The files, in turn, are contained in directories (a directory is a special file that can contain other files and directories).
- A directory can contain other directories, giving rise to a hierarchical structure. This hierarchical organization of files is called the file system.
- The Linux file system provides a unified model of all storage in the system. The file system has a single root directory, indicated by a forward slash (/).
- Then there is a hierarchy of files and directories. Parts of the file system can reside in different physical media, such as hard disk, floppy disk, and CD-ROM. Figure 1: illustrates the concept of the Linux file system and how it spans multiple physical devices.



- Figure 1: The Linux File System Provides a Unified View of Storage that May Span Multiple drives.
- If you are familiar with MS-DOS or Windows, notice that there is no concept of a drive letter in UNIX.
- You can have long filenames (up to 256 characters), and filenames are case sensitive. Often, UNIX filenames have multiple extensions, such as sample.tar.Z.
- Some UNIX filenames include the following: index.html, Makefile, kernel-enterprise-2.4.18-3.i686.rpm, .bash_profile, and httpd_src.tar.gz.
- To locate a file, you need more than just the file's name; you also need information about the directory hierarchy.
- The term *pathname* refers to the complete specification necessary to locate a file-the complete hierarchy of directories leading to the file-and the filename. Figure 2: shows a typical Linux pathname for a file.



STEP-2:

- A Typical Linux Pathname.As you can see from STEP-2, a Linux pathname consists of the following parts:
- The root directory, indicated by a forward slash (/) character.
- The directory hierarchy, with each directory name separated from the previous one by a forward slash (/) character. A / appears after the last directory name.
- The filename, with a name and one or more optional extensions.
- Many directories have specific purposes. If you know the purpose of specific directories, finding your way around Linux directories is easier. Another benefit of knowing the typical use of directories is that you can guess where to look for specific types of files when you face a new situation.

- Table 7-2 briefly describes the directories in a Linux system:

Directory	**Description**
/	Root directory that forms the base of the file system. All files and directories are contained logically in the root directory, regardless of their physical locations.
/bin	Contains the executable programs that are part of the Linux operating system. Many Linux commands, such as cat, cp, ls, more, and tar, are located in /bin.
/boot	Contains the Linux kernel and other files the LILO and GRUB boot managers need (the kernel and other files can be anywhere, but it is customary to place them in the /boot directory).
/dev	Contains all device files. Linux treats each device as a special file; all such files are located in the device directory /dev.

/etc	Contains most system configuration files and the initialization scripts (in the /etc/rc.d subdirectory)
/home	Conventional location of the home directories of all users. User naba's home directory, for example, is /home/naba.
/lib	Contains library files, including the loadable driver modules, needed to boot the system
/lost+found	Directory for lost files. Every disk partition has a lost+found directory.
/mnt	A directory, typically used to mount devices temporarily, such as floppy disks and disk partitions. Also contains the /mnt/floppy directory for mounting floppy disks and the /mnt/cdrom directory for mounting the CD-ROM drive. (Of course, you can mount the CD-ROM drive on another directory as well.)
/proc	A special directory that contains information about various aspects of the Linux system
/root	The home directory for the root user
/sbin	Contains executable files representing commands typically used for system-administration tasks. Commands such as mount, halt, umount, and shutdown reside in the /sbin directory.
/tmp	Temporary directory that any user can use as a scratch directory, meaning that the contents of this directory are considered unimportant and usually are deleted every time the system boots

/usr	Contains the subdirectories for many important programs, such as the X Window System, and the online manual
/var	Contains various system files (such as logs), as well as directories for holding other information, such as files for the Web server and anonymous FTP server

- The /usr and /var directories also contains a host of useful subdirectories. Table 7-3 lists a few of the important subdirectories in /usr. Table 7-4 shows a similar breakdown for the /var directory.

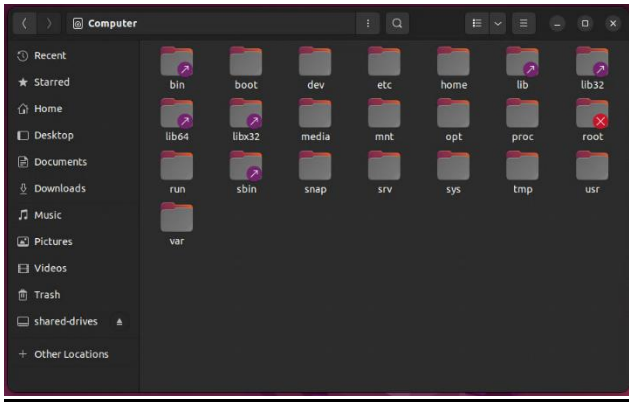
Subdirectory	**Description**
/usr/X11R6	Contains the XFree86 (X Window System) software
/usr/bin	Contains executable files for many more Linux commands, including utility programs commonly available in Linux, but is not part of the core Linux operating system
/usr/games	Contains some Linux games such as Chromium, and Maelstrom
/usr/include	Contains the header files (files with names ending in .h) for the C and C++ programming languages; also includes the X11 header files in the /usr/include/X11 directory and the kernel header files in the /usr/include/linux directory
/usr/lib	Contains the libraries for C and C++ programming languages; also contains many other libraries, such as database libraries, graphical toolkit libraries, and so on
/usr/local	Contains local files. The /usr/local/bin directory, for example, is supposed to be the location for any executable program developed on your system.

/usr/sbin	Contains many administrative commands, such as commands for electronic mail and networking
/usr/share	Contains shared data, such as default configuration files and images for many applications. For example, /usr/share/gnome contains various shared files for the GNOME desktop; and /usr/share/doc has the documentation files for many Linux applications (such as the Bash shell, mtools, and the GIMP image processing program).
/usr/share/man	Contains the online manual (which you can read by using the man command)
/usr/src	Contains the source code for the Linux kernel (the core operating system)

Subdirectory	**Description**
/var/cache	Storage area for cached data for a applications
/var/lib	Contains information relating to the current state of applications
/var/lock	Contains lock files to ensure that a resource is used by one application only
/var/log	Contains log files organized into subdirectories. The syslogd server stores its log files in /var/log and the exact content of the files depend on the syslogd configuration file: /etc/syslog.conf. For example, /var/log/messages is the main system log file, /var/log/secure contains log messages from secure services such as sshd and xinetd, and /var/log/maillog contains the log of mail messages.
/var/mail	Contains user mailbox files

/var/opt	Contains variable data for packages stored in /opt directory
/var/run	Contains data describing the system since it was booted
/var/spool	Contains data that's waiting for some kind of processing
/var/tmp	Contains temporary files preserved between system reboots
/var/yp	Contains Network Information Service (NIS) database files

SCREENSHOTS:



CONCLUSION:

Hence we discussed the Linux File System sucessfully.

Ex. No.:4 **Display the Processor Information**

Date: 26/08/2022

AIM:

To write a C program to display the Processor Information.

PROCEDURE:

Step 1: start the program

Step2: First create a pointer variable of cpuinfo which is pointing to the address of the cpuinfo File.

Step3: Then create character type pointer variable named as arg defined as 0.

Step4: Then create a size_t type variable named size which represent the size of the object in bytes.

Step5: Then we use while loop which checks that the delimiter (getdelim()) of the stream of cpuinfo should not equals to -1 then the statement of while loop (puts(arg)) use to print line by line.

Step6: Then we use free() to deallocate the memory which is allocated in arg variable.

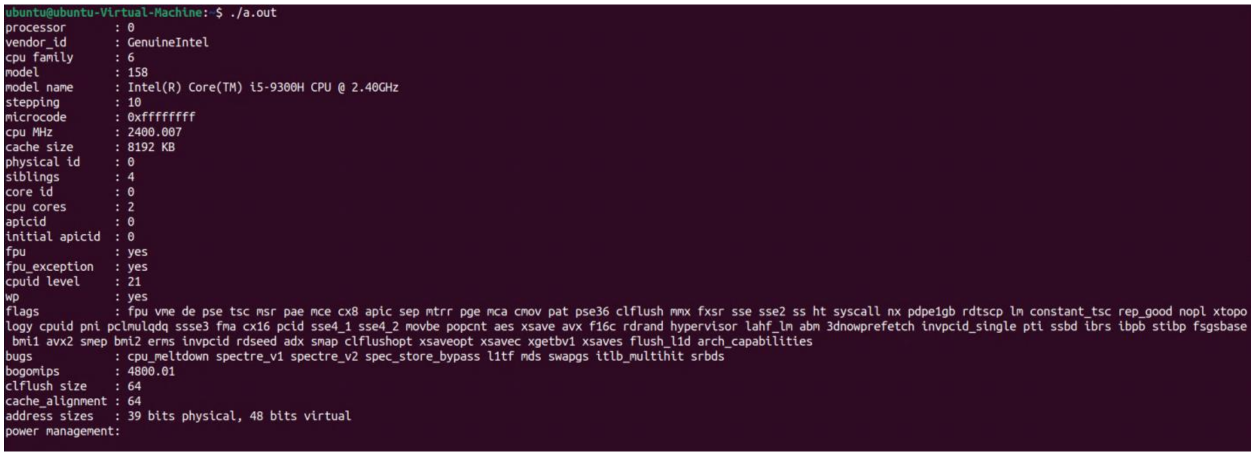
Step7: Then we use fclose to close the file stream of cpuinfo.

Step 8: Then the program is compiled and the output is displayed

PROGRAM:

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(int argc, char **argv)
{
  FILE *cpuinfo = fopen("/proc/cpuinfo", "rb");
  char *arg = 0;
  size_t size = 0;
  while(getdelim(&arg, &size, 0, cpuinfo) != -1)
  {
    puts(arg);
  }
  free(arg);
  fclose(cpuinfo);
  time_t end = time(NULL);
      return 0;
}
```

OUTPUT:



```
ubuntu@ubuntu-Virtual-Machine:~$ ./a.out
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz
stepping        : 10
microcode       : 0xffffffff
cpu MHz         : 2400.007
cache size      : 8192 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 21
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm constant_tsc rep_good nopl xtopo
logy cpuid pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch invpcid_single pti ssbd ibrs ibpb stibp fsgsbase
 bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap clflushopt xsaveopt xsavec xgetbv1 xsaves flush_l1d arch_capabilities
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs itlb_multihit srbds
bogomips        : 4800.01
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:
```

RESULT:

The given program is compiled and executed successfully on Ubuntu OS

Ex. No: 5 **Demonstration of System Calls**

Date: 05/09/2022

AIM:

To write a C Program to Create a Process using System Call.

PROCEDURE:

Step1: start the program

Step2: First we created a pid_t type variable named pid which is capable of representing process ID, it is defined as fork() function which use to create a new process and the ID of the process is been stored in pid variable.

Step3: Then we use if condition that if(pid==0) then it prints parent process ID (PPID) using getppid() and process ID using getpid() then it exit from the program by printing EXIT_SUCCESS.

Step4:Else if(pid>0) condition is been true then it will print process ID using getpid() and waiting for child process to finish using wait(NULL) then it prints “Child process finished”.Else it prints “Unable to create child process.”

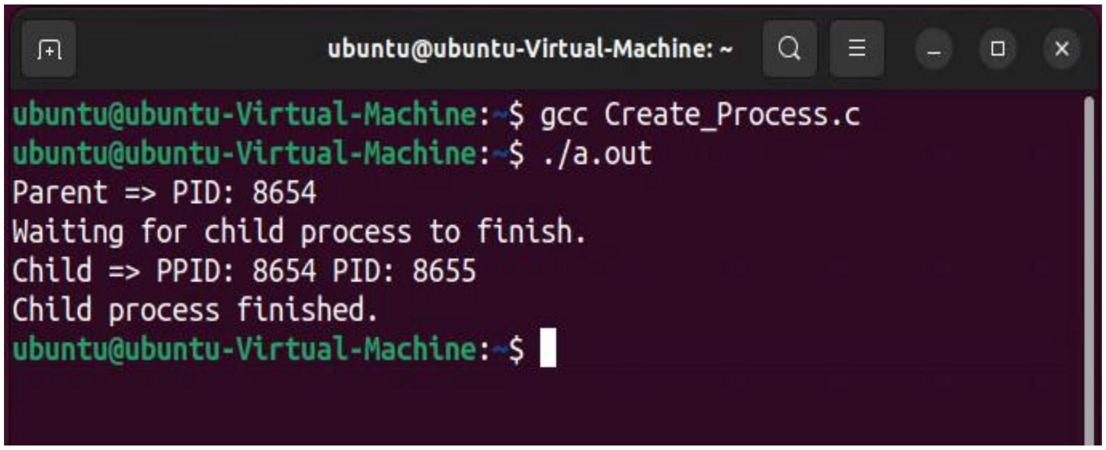
Step5: Program will return EXIT_SUCCESS.

Step 6: Then the program is compiled and the output is displayed

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
 int main(void) {
  pid_t pid = fork();

  if(pid == 0) {
    printf("Child => PPID: %d PID: %d\n", getppid(), getpid());
    exit(EXIT_SUCCESS);
  }
  else if(pid > 0) {
    printf("Parent => PID: %d\n", getpid());
    printf("Waiting for child process to finish.\n");
    wait(NULL);
    printf("Child process finished.\n");
  }
  else {
    printf("Unable to create child process.\n");
  }

  return EXIT_SUCCESS;
}
```

OUTPUT:



```
ubuntu@ubuntu-Virtual-Machine:~$ gcc Create_Process.c
ubuntu@ubuntu-Virtual-Machine:~$ ./a.out
Parent => PID: 8654
Waiting for child process to finish.
Child => PPID: 8654 PID: 8655
Child process finished.
ubuntu@ubuntu-Virtual-Machine:~$ 
```

RESULT:

The given program is compiled and executed successfully on Ubuntu OS

Ex. No: 6 **Creating Multiple Child Process**

Date: 06/09/2022

AIM:

To write a C Program to Create more than One Child.

PROCEDURE:

Step1: start the program

Step2: First we will use for loop which will run n times (n=5).

Step3: In for loop if (fork () ==0) then it will print the child pid using getpid(), and parent process ID using getppid() then it will exit(0).

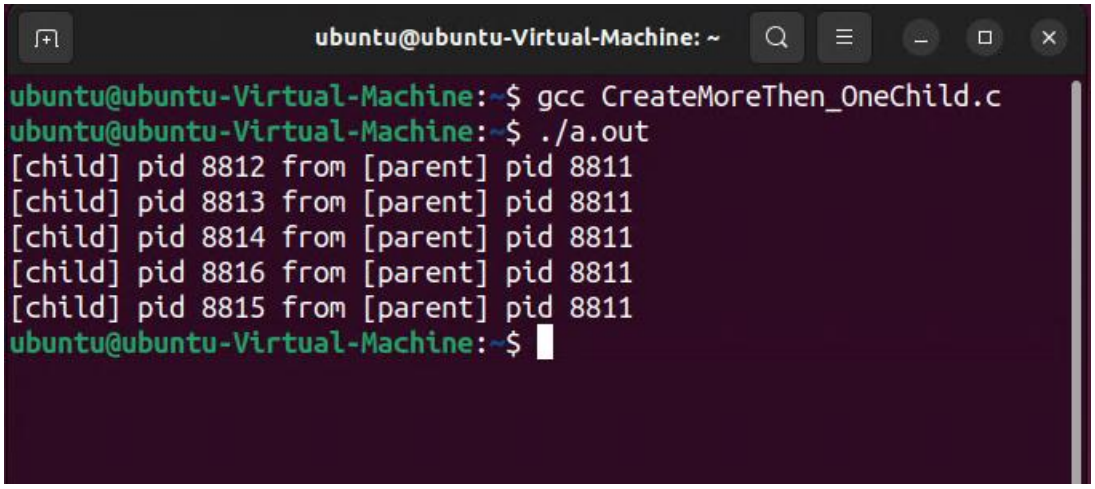
Step4: Then outside the for loop we will create another for loop which will also run n times (n=5), In which wait (NULL) statement is been executed.

Step 5: Then the program is compiled and the output is displayed

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
  int main()
{
int i;
    for( i=0;i<5;i++) // loop will run n times (n=5)
    {
        if(fork() == 0)
        {
            printf("[child] pid %d from [parent] pid %d\n",getpid(),getppid());
            exit(0);
        }
    }
    for( i=0;i<5;i++) // loop will run n times (n=5)
    wait(NULL);
 }
```

OUTPUT:



```
ubuntu@ubuntu-Virtual-Machine:~$ gcc CreateMoreThen_OneChild.c
ubuntu@ubuntu-Virtual-Machine:~$ ./a.out
[child] pid 8812 from [parent] pid 8811
[child] pid 8813 from [parent] pid 8811
[child] pid 8814 from [parent] pid 8811
[child] pid 8816 from [parent] pid 8811
[child] pid 8815 from [parent] pid 8811
ubuntu@ubuntu-Virtual-Machine:~$
```

CONCLUSION:

The given program is compiled and executed successfully on Ubuntu OS

Ex. No: 7 **Inter-process Communication through Shared Memory**

Date: 07/09/2022

AIM:

To write a C Program for IPC through Shared Memory.

PROCEDURE:

Writer:

Step1: start the program

Step 2: First we define a key_t type variable name key which use to request the resource and it will define with ftok() function which takes a character string that identifies a path and an integer value.

Step 3: Then we initialize a variable shmid which stores identifier of the shared memory using shmget().

Step 4: Then we create a character type pointer variable named 'str' which use to attach the shared memory segment using shmat() function.

Step 5: Then we use print statement to instruct user to write data which will be written in the shared memory segment using gets(str).

Step 6: Then we will print the written data by the user in the shared memory.

Step 7: Then we use shmdt(str) to detach the shared memory segment.

Step 8: Then the program is compiled and the output is displayed

Reader:

Step1: start the program

Step 2: First we define a key_t type variable name key which use to request the resource and it will define with ftok () function which takes a character string that identifies a path and an integer value.

Step 3: Then we initialize a variable named shmid which stores identifier of the shared memory using shmget ().

Step 4: Then we create a character type pointer variable named 'str' which use to attach the shared memory segment using shmat () function.

Step 5: Then we will print the data which was written on the shared memory segment.

Step 6: Then we use shmdt(str) to detach the shared memory segment.

Step 7: Then the program is compiled and the output is displayed

PROGRAM:

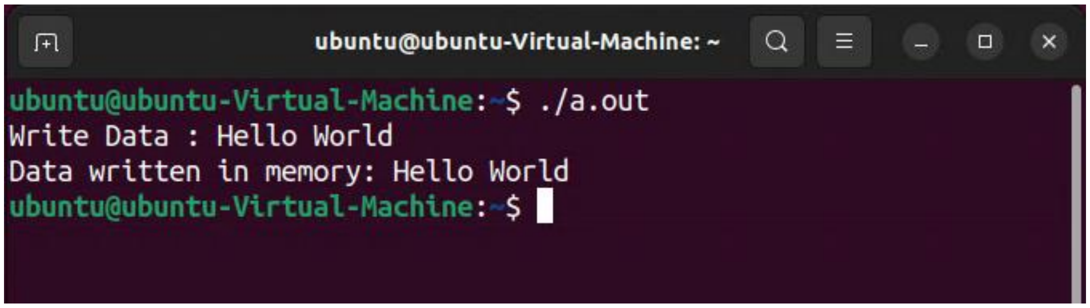
Writer

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
int main()
{
   // ftok to generate unique key
   key_t key = ftok("shmfile",65);

   // shmget returns an identifier in shmid
   int shmid = shmget(key,1024,0666|IPC_CREAT);

   // shmat to attach to shared memory
   char *str = (char*) shmat(shmid,(void*)0,0);

   printf("Write Data : ");
   gets(str);

   printf("Data written in memory: %s\n",str);

   //detach from shared memory
   shmdt(str);

   return 0;
}
```

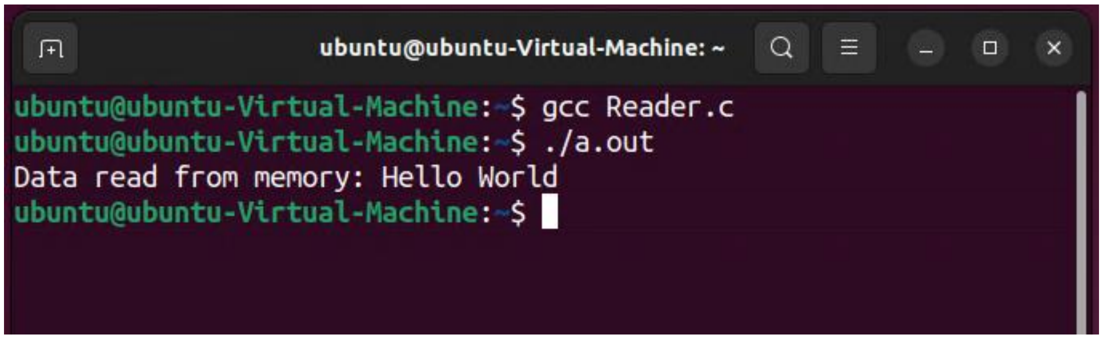
Reader

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
int main()
{
        // ftok to generate unique key
        key_t key = ftok("shmfile",65);

        // shmget returns an identifier in shmid
        int shmid = shmget(key,1024,0666|IPC_CREAT);

        // shmat to attach to shared memory
        char *str = (char*) shmat(shmid,(void*)0,0);

        printf("Data read from memory: %s\n",str);

        //detach from shared memory
        shmdt(str);

        // destroy the shared memory
        shmctl(shmid,IPC_RMID,NULL);

        return 0;
}
```

OUTPUT:



```
ubuntu@ubuntu-Virtual-Machine:~$ ./a.out
Write Data : Hello World
Data written in memory: Hello World
ubuntu@ubuntu-Virtual-Machine:~$
```



```
ubuntu@ubuntu-Virtual-Machine:~$ gcc Reader.c
ubuntu@ubuntu-Virtual-Machine:~$ ./a.out
Data read from memory: Hello World
ubuntu@ubuntu-Virtual-Machine:~$
```

RESULT:

The given program is compiled and executed successfully on Ubuntu

Ex. No: 8 **Inter-process Communication through Pipe**

Date: 12/09/2022

AIM:

To write a C Program for IPC through Pipe.

PROCEDURE:

Step1: start the program

Step 2: First we initialize two integer variable one is an array of size 2 named fd[2] and a variable named n.

Step 3: Then we use a character type variable which is an array as buffer [100].

Step 4: Then we create a pid_t type variable named p which is capable of representing process ID.

Step 5: Then we use pipe(fd), which is use to creates a unidirectional pipe with two end fd[0] and fd[1].

Step 6: Then we create a new process using fork () and stored the Process ID into variable p.

Step 7: We used an if condition if(p>0) then it will print "Parent Passing value to the child" and it will write the data in fd[1] which is the end of the pipe

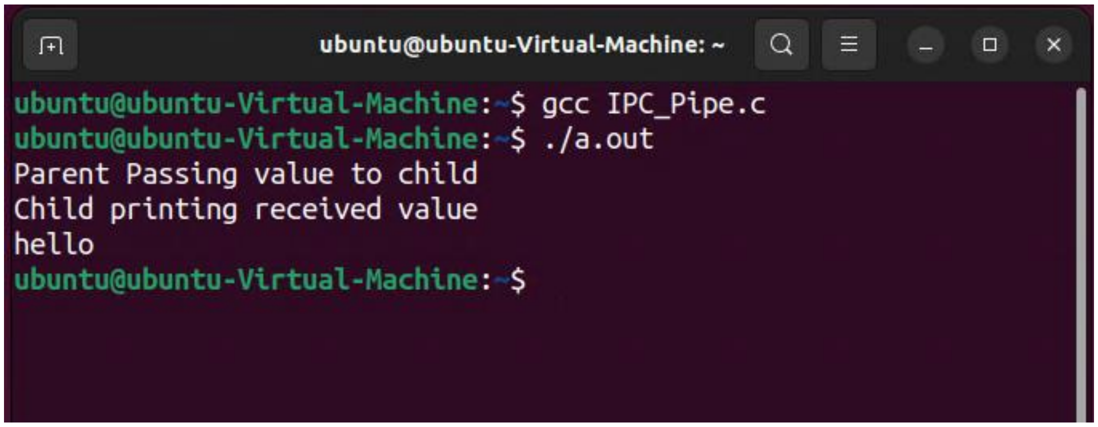
Step 8:Else it will print "Child printing received value" and it will read the end of the pipe which is fd[0] and store into n then we will use write() function to write the stored value on the variable n.

Step 9: Then the program is compiled and the output is displayed

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
int main()
{
int fd[2],n;
char buffer[100];
pid_t p;
pipe(fd); //creates a unidirectional pipe with two end fd[0] and fd[1]
p=fork();
if(p>0) //parent
{
printf("Parent Passing value to child\n");
write(fd[1],"hello\n",6); //fd[1] is the write end of the pipe
wait(NULL);
}
else // child
{
printf("Child printing received value\n");
n=read(fd[0],buffer,100); //fd[0] is the read end of the pipe
write(1,buffer,n);
}
}
```

OUTPUT:



```
ubuntu@ubuntu-Virtual-Machine:~$ gcc IPC_Pipe.c
ubuntu@ubuntu-Virtual-Machine:~$ ./a.out
Parent Passing value to child
Child printing received value
hello
ubuntu@ubuntu-Virtual-Machine:~$
```

RESULT:

The given program is compiled and executed successfully on Ubuntu

Ex. No: 9 **Display Shell Script Parameters**

Date: 19/09/2022

AIM:

To write a bash shell script that has parameters. These parameters start from **$1** to **$9.**

Parameters	**Function**
$1-$9	Represent positional parameters for arguments one to nine
${10}-${n}	Represent positional parameters for arguments after nine
$0	Represent name of the script
$∗	Represent all the arguments as a single string
$@	Same as $∗, but differ when enclosed in (")
$#	Represent total number of arguments
$$	PID of the script
$?	Represent last return code

When we pass arguments into the command line interface, a positional parameter is assigned to these arguments through the shell.

If there are more than 9 arguments, then **tenth** or onwards arguments can't be assigned as $10 or $11.x

You have to either process or save the $1 parameter, then with the help of **shift** command drop parameter 1 and move all other arguments down by one. It will make $10 as $9, $9 as $8 and so on.

PEOCEDURE:

Step1: Represent positional parameters for arguments one to nine

Step2: First statement will print “this script name is $0”, $0 represent the name of the script.

Step3: Second statement will print “the first argument is $1”, $1 represent the first argument.

Step4: Third statement will print “the second argument is $2”, $2 represent the second argument.

Step5: Fourth statement will print “the third argument is $3”, $3 represent the third argument.

Step6: Fifth statement will print "$$ PID of the script", $$ represent the Process ID of the script.

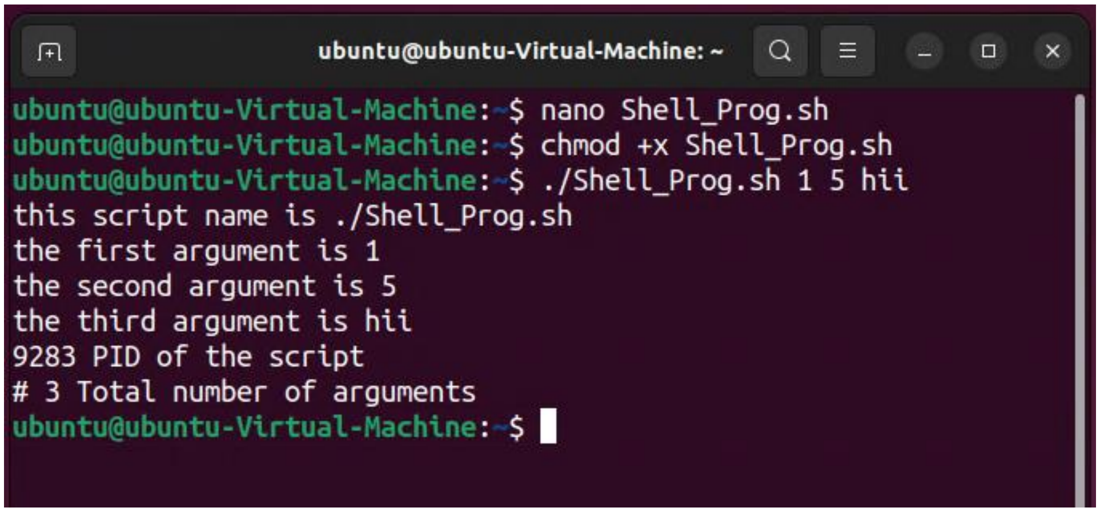
Step7: Sixth statement will print "\# $# Total number of arguments", as we know '#' is the special character used for comments so we used '\' to print '#' and $# use to count the arguments given by the user.

Step 9: Then the program is compiled and the output is displayed

<u>PROGRAM:</u>

```
echo this script name is $0

echo the first argument is $1

echo the second argument is $2

echo the third argument is $3

echo $$ PID of the script

echo \# $# Total number of arguments
```

<u>OUTPUT:</u>



```
ubuntu@ubuntu-Virtual-Machine:~$ nano Shell_Prog.sh
ubuntu@ubuntu-Virtual-Machine:~$ chmod +x Shell_Prog.sh
ubuntu@ubuntu-Virtual-Machine:~$ ./Shell_Prog.sh 1 5 hii
this script name is ./Shell_Prog.sh
the first argument is 1
the second argument is 5
the third argument is hii
9283 PID of the script
# 3 Total number of arguments
ubuntu@ubuntu-Virtual-Machine:~$
```

<u>RESULT:</u>

The given program is compiled and executed successfully on Ubuntu

Ex. No: 10 Shell Script Program to Print the Value of Variables

Date: 03/10/2022

AIM:

To create a shell script program of two variables and then print values of variables on the console screen.

PROCEDURE:

Step1: create a text editor with .sh extension

Step2: create two variables

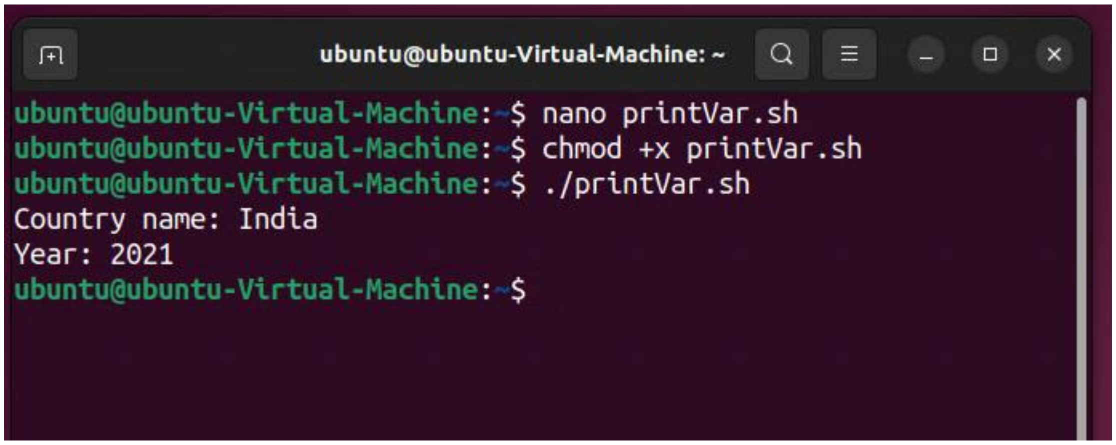
Step3: where initialize both the variables using echo command

Step4: hence now the two variables are printed

PROGRAM:

```
country="India"
year=2021

echo "Country name: $country"
echo "Year: $year"
```

OUTPUT:



```
ubuntu@ubuntu-Virtual-Machine:~$ nano printVar.sh
ubuntu@ubuntu-Virtual-Machine:~$ chmod +x printVar.sh
ubuntu@ubuntu-Virtual-Machine:~$ ./printVar.sh
Country name: India
Year: 2021
ubuntu@ubuntu-Virtual-Machine:~$
```

RESULT:

The given program is compiled and executed successfully on Ubuntu

Ex. No: 11 **Shell Script to Add Two Numbers**

Date: 12/10/2022

AIM:

To create a shell script program to add two integer numbers and then print the sum of numbers on the console screen.

PROCEDURE:

Step 1: get two numbers num1and num2

Step 2: initialize both the numbers in num3

Step 3: using expr $num1+$num2

Step 4: then execute using echo scriprt

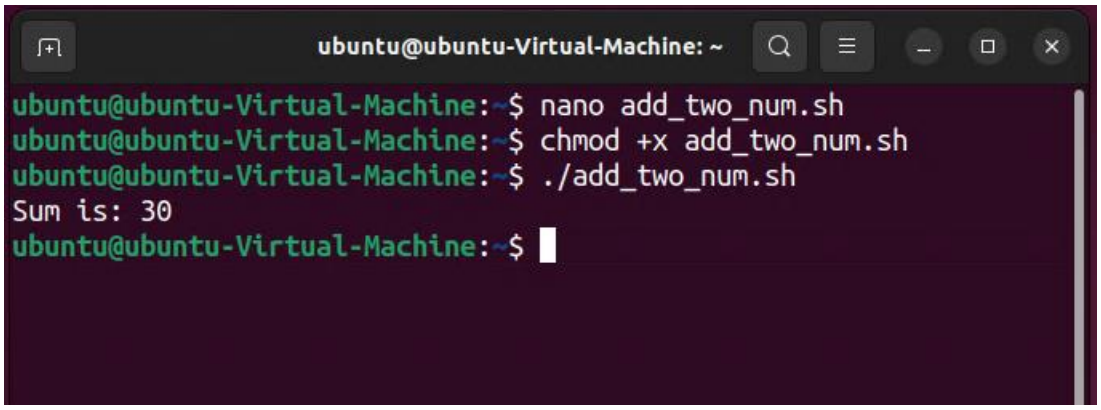
Step 5: Now, we will save the shell script program with the "add_two_num.sh" name

Step 6: hence now the addition of two numbers will be displayed

PROGRAM:

```
#!/bin/bash

# Program name: "add_two_num.sh"
# Shell script program to add two numbers.

num1=10
num2=20

num3=`expr $num1 + $num2`

echo "Sum is: $num3"
```

OUTPUT:


```
ubuntu@ubuntu-Virtual-Machine:~$ nano add_two_num.sh
ubuntu@ubuntu-Virtual-Machine:~$ chmod +x add_two_num.sh
ubuntu@ubuntu-Virtual-Machine:~$ ./add_two_num.sh
Sum is: 30
ubuntu@ubuntu-Virtual-Machine:~$
```

RESULT:

The given program is compiled and executed successfully on Ubuntu

Ex. No: 12 **Shell script to find greatest of three numbers**

Date: 19/10/2022

AIM:

To create a shell script program to find greatest of three numbers and then print the three numbers on the console screen.

PROCEDURE:

Step 1: Get three numbers. Say num1, num2, num2

Step 2: If (num1 > num2) and (num1 > num3) then echo value of num1

Step3: elif (num2 > num1) and (num2 > num3) then echo value of num2

step 4: Otherwise, echo value of num3

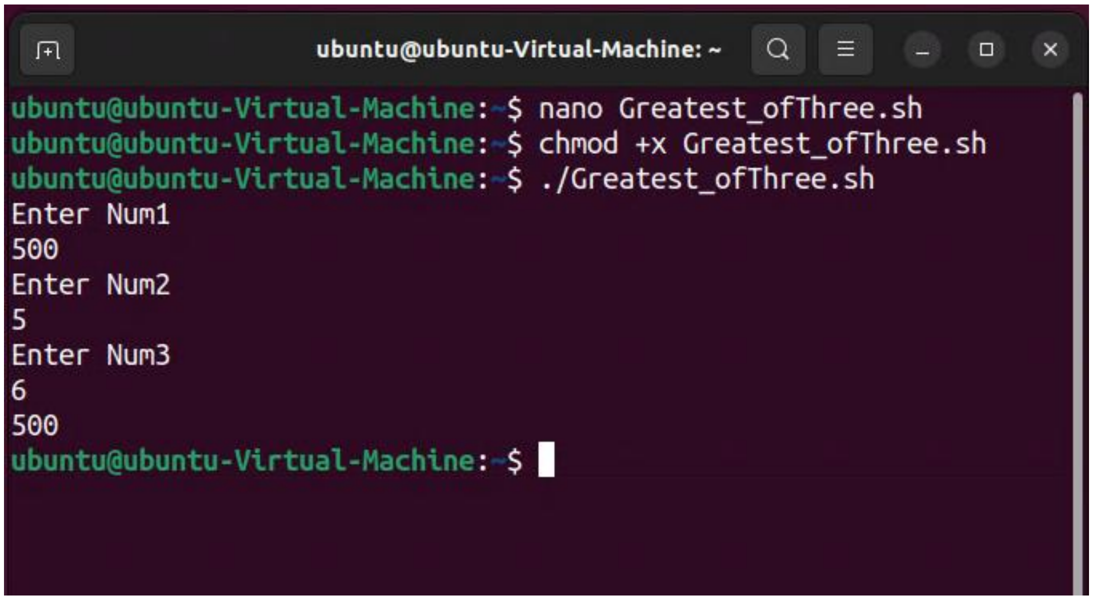
step 5: here we use -gt for greater than and && for logical AND condition.

Step 6: hence the output will be executed

PROGRAM:

```
echo "Enter Num1"
read num1
echo "Enter Num2"
read num2
echo "Enter Num3"
read num3

if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]
then
   echo $num1
elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]
then
   echo $num2
else
   echo $num3
fi
```

OUTPUT:



```
ubuntu@ubuntu-Virtual-Machine:~$ nano Greatest_ofThree.sh
ubuntu@ubuntu-Virtual-Machine:~$ chmod +x Greatest_ofThree.sh
ubuntu@ubuntu-Virtual-Machine:~$ ./Greatest_ofThree.sh
Enter Num1
500
Enter Num2
5
Enter Num3
6
500
ubuntu@ubuntu-Virtual-Machine:~$
```

RESULT:

The given program is compiled and executed successfully on Ubuntu OS

Ex. No: 13 **Shell program to find sum of n numbers**

Date: 27/10/2022

AIM:

To create a shell script program to find sum of n numbers and then print the n numbers.

PROCEDURE:

Step1: Get N (Total Numbers).

Step2: Get N numbers using loop.

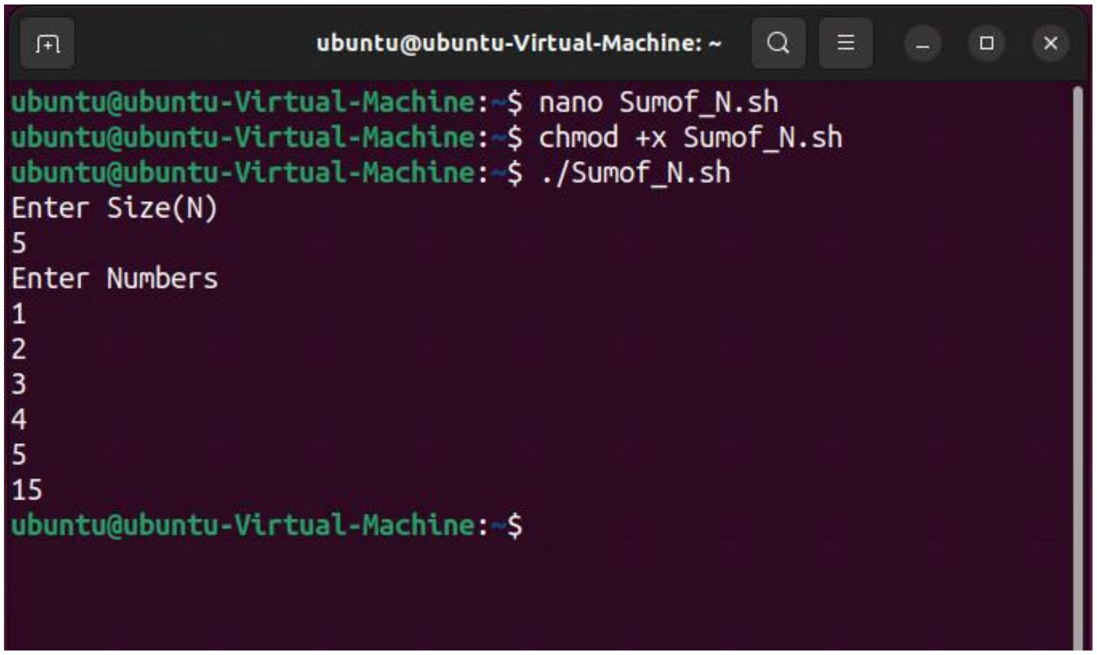
Step3: Calculate the sum

Step4: Print the result.

PROGRAM:

```
echo "Enter Size(N)"
read N
i=1
sum=0
echo "Enter Numbers"
while [ $i -le $N ]
do
  read num
  sum=$((sum + num)) #sum+=num
  i=$((i + 1))
done
echo $sum
```

OUTPUT:



```
ubuntu@ubuntu-Virtual-Machine:~$ nano Sumof_N.sh
ubuntu@ubuntu-Virtual-Machine:~$ chmod +x Sumof_N.sh
ubuntu@ubuntu-Virtual-Machine:~$ ./Sumof_N.sh
Enter Size(N)
5
Enter Numbers
1
2
3
4
5
15
ubuntu@ubuntu-Virtual-Machine:~$
```

RESULT:

The given program is compiled and executed successfully on Ubuntu OS

Ex. No: 14 **Shell script to find factorial of a number**

Date: 28/10/2022

AIM:

To write a shell script program to find factorial of a number print the numbers on the console screen.

PROCEDURE:

step1: Get a number

step2: Use for loop or while loop to compute the factorial by using the below formula

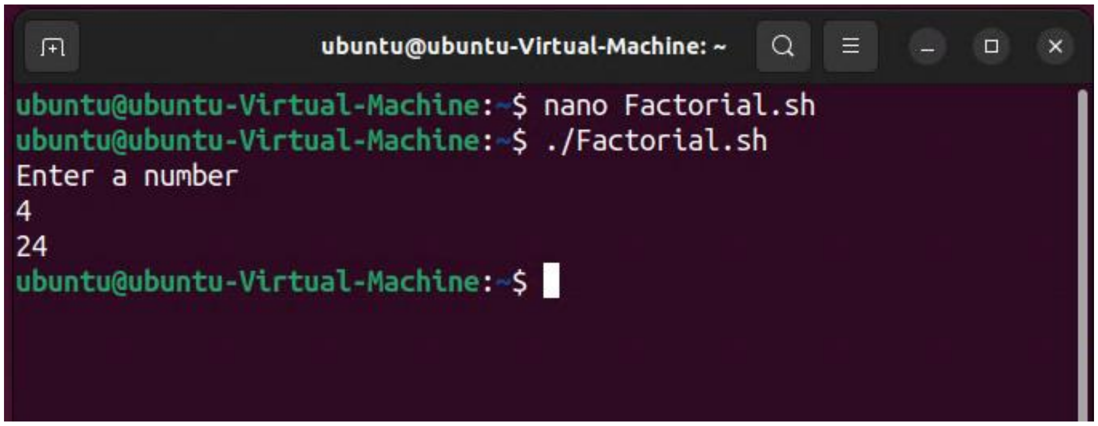
step3: fact(n) = n * n-1 * n-2 *. 1

step4: Display the result.

PROGRAM:

```
echo "Enter a number"
read num
fact=1
for((i=2;i<=num;i++))
{
  fact=$((fact * i))  #fact = fact * i
}
echo $fact
```

OUTPUT:



```
ubuntu@ubuntu-Virtual-Machine:~$ nano Factorial.sh
ubuntu@ubuntu-Virtual-Machine:~$ ./Factorial.sh
Enter a number
4
24
ubuntu@ubuntu-Virtual-Machine:~$
```

RESULT:

The given program is compiled and executed successfully on Ubuntu OS

Ex. No: 15 **Shell program to add two numbers using user defined function**

Date: 31/10/2022

AIM:

To write a shell script program to add two numbers using functions and print the numbers on the console screen.

PROCEDURE:

step1: Initialize two variables.

step2: Declare and implement the addition function.

step3: Call the add function with two arguments.

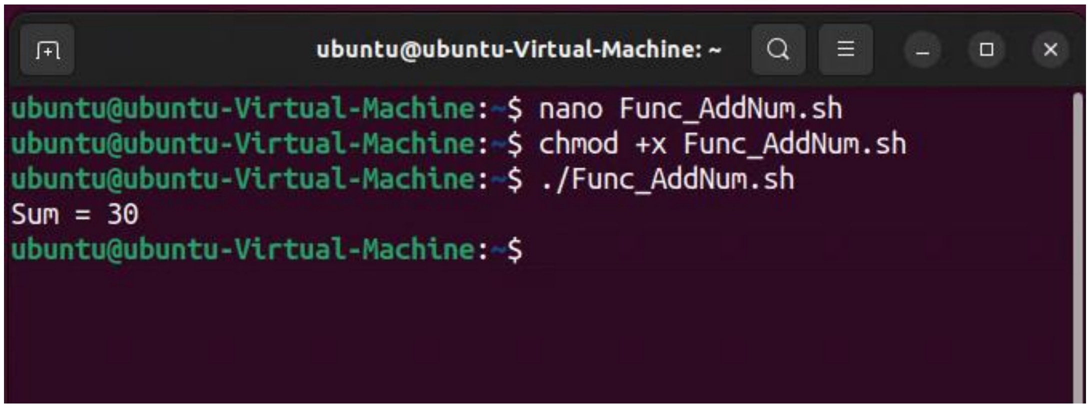
Step4: Add two variables using function in shell script

Step5: call the add function and pass the values

PROGRAM:

```
function add()
{
   sum=$(($1 + $2))
   echo "Sum = $sum"
}

a=10
b=20
add $a $b
```

OUTPUT:



```
ubuntu@ubuntu-Virtual-Machine:~$ nano Func_AddNum.sh
ubuntu@ubuntu-Virtual-Machine:~$ chmod +x Func_AddNum.sh
ubuntu@ubuntu-Virtual-Machine:~$ ./Func_AddNum.sh
Sum = 30
ubuntu@ubuntu-Virtual-Machine:~$
```

RESULT:

The given program is compiled and executed successfully on Ubuntu OS

Ex. No: 16 **First Come First Serve Scheduling Algorithm**

Date : 03/11/2022

<u>AIM:</u>

To calculate average waiting time and average turnaround time using First Come First Serve Scheduling Algorithm

<u>PROCEDURE:</u>

STEP1: - First take user input of Total number of process (maximum 20).

STEP2: - use for loop to take user input Burst time.

STEP3: - use for loop to calculate the Burst time and Waiting time.

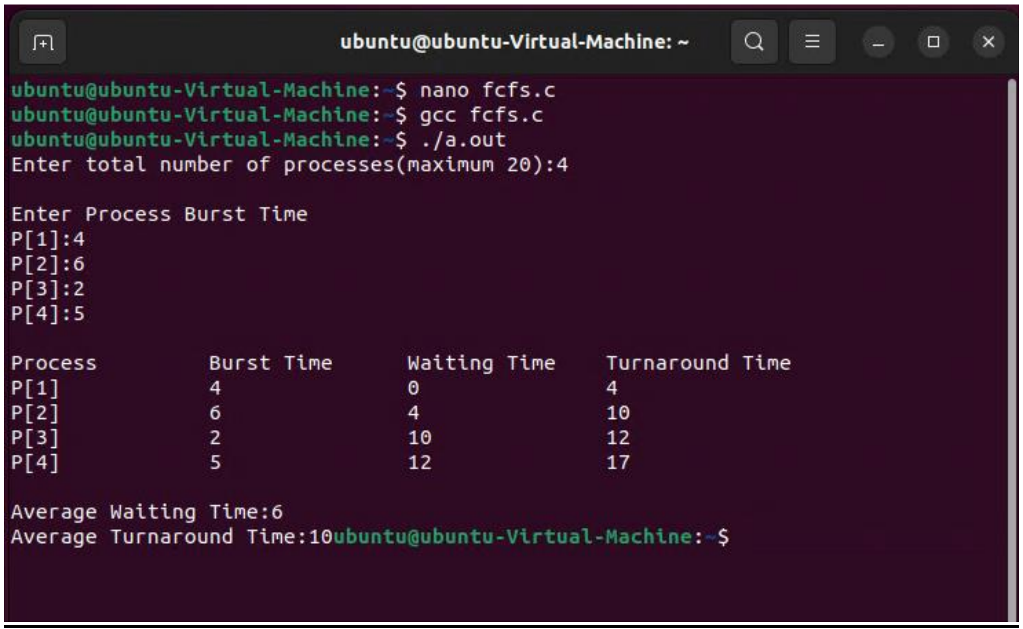
STEP4: - after that arrange the process, burst time, waiting time, and Turnaround time in tabular format and print it using for loop.

STEP5: - then calculate and print the average waiting time and average turnaround time.

<u>PROGRAM:</u>

```
#include<stdio.h>

int main()
{
   int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
   printf("Enter total number of processes(maximum 20):");
   scanf("%d",&n);

   printf("\nEnter Process Burst Time\n");
   for(i=0;i<n;i++)
   {
      printf("P[%d]:",i+1);
      scanf("%d",&bt[i]);
   }

   wt[0]=0;    //waiting time for first process is 0

   //calculating waiting time
   for(i=1;i<n;i++)
   {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];
    }

    printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

    //calculating turnaround time
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        avwt+=wt[i];
        avtat+=tat[i];
        printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
    }

    avwt/=i;
    avtat/=i;
    printf("\n\nAverage Waiting Time:%d",avwt);
    printf("\nAverage Turnaround Time:%d",avtat);

    return 0;
}
```

OUTPUT:



```
ubuntu@ubuntu-Virtual-Machine:~$ nano fcfs.c
ubuntu@ubuntu-Virtual-Machine:~$ gcc fcfs.c
ubuntu@ubuntu-Virtual-Machine:~$ ./a.out
Enter total number of processes(maximum 20):4

Enter Process Burst Time
P[1]:4
P[2]:6
P[3]:2
P[4]:5

Process         Burst Time      Waiting Time    Turnaround Time
P[1]            4               0               4
P[2]            6               4               10
P[3]            2               10              12
P[4]            5               12              17

Average Waiting Time:6
Average Turnaround Time:10ubuntu@ubuntu-Virtual-Machine:~$
```

RESULT:

The given program is compiled and executed successfully on Ubuntu OS

Ex. No: 17 **Priority Scheduling Algorithm**

Date: 10/11/2022

AIM:

To arrange the process in terms of priority for the execution using priority scheduling algorithm.

PROCEDURE:

STEP1: - First create a swap function to swap two variables using third variable.

STEP2: - then in main function take user input of number of processes.

STEP3: - create three arrays for Burst time, priority and index and by using for loop enter the Burst time and Priority value of each process.

STEP4: - use for loop to re-arrange the process on the basis of priority by using swap function we were created in the starting.

STEP5: - after that print the scheduled processes Process id, burst time, Wait time, Turnaround time.

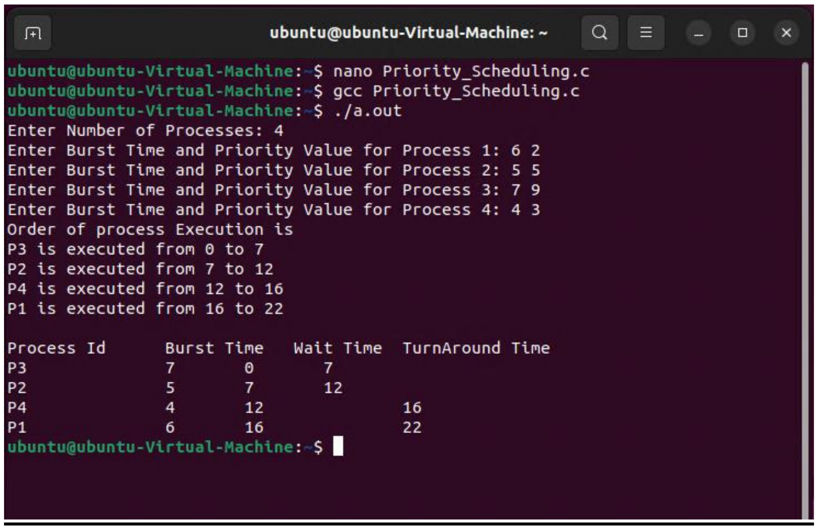
PROGRAM:

```
#include <stdio.h>

 //Function to swap two variables

void swap(int *a,int *b)

{

        int temp=*a;

        *a=*b;

        *b=temp;

}

int main()

{

        int n;
```

```
            printf("Enter Number of Processes: ");

            scanf("%d",&n);

// b is array for burst time, p for priority and index for process id
            int b[n],p[n],index[n];

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

            {

            printf("Enter Burst Time and Priority Value for Process %d: ",i+1);

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

            index[i]=i+1;

            }

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

            {

            int a=p[i],m=i;


            //Finding out highest priority element and placing it at its desired position

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

            {

            if(p[j] > a)

            {

            a=p[j];

            m=j;

            }

            }


            //Swapping processes
```

```
        swap(&p[i], &p[m]);

        swap(&b[i], &b[m]);

        swap(&index[i],&index[m]);

        }


        // T stores the starting time of process

        int t=0;


        //Printing scheduled process

        printf("Order of process Execution is\n");

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

        {

        printf("P%d is executed from %d to %d\n",index[i],t,t+b[i]);

        t+=b[i];

        }

        printf("\n");

        printf("Process Id          Burst Time   Wait Time TurnAround Time\n");

        int wait_time=0;

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

        {

        printf("P%d      %d       %d       %d\n",index[i],b[i],wait_time,wait_time + b[i]);

        wait_time += b[i];

        }

        return 0;

}
```

OUTPUT:



```
ubuntu@ubuntu-Virtual-Machine:~$ nano Priority_Scheduling.c
ubuntu@ubuntu-Virtual-Machine:~$ gcc Priority_Scheduling.c
ubuntu@ubuntu-Virtual-Machine:~$ ./a.out
Enter Number of Processes: 4
Enter Burst Time and Priority Value for Process 1: 6 2
Enter Burst Time and Priority Value for Process 2: 5 5
Enter Burst Time and Priority Value for Process 3: 7 9
Enter Burst Time and Priority Value for Process 4: 4 3
Order of process Execution is
P3 is executed from 0 to 7
P2 is executed from 7 to 12
P4 is executed from 12 to 16
P1 is executed from 16 to 22

Process Id      Burst Time   Wait Time  TurnAround Time
P3              7       0       7
P2              5       7       12
P4              4       12              16
P1              6       16              22
ubuntu@ubuntu-Virtual-Machine:~$ 
```

RESULT:

The given program is compiled and executed successfully on Ubuntu OS

Ex. No: 18 **Round Robin Scheduling Algorithm**

Date: 17/11/2022

AIM:

To calculate average waiting time and average turnaround time using Round Robin Scheduling Algorithm

PROCEDURE:

STEP1: - First take user input of Total number of process (maximum 20).

STEP2: - Then instruct user to enter Arrival time and Burst time.

STEP3: - Then take input of Burst time.

STEP4: - use for loop to calculate the Turnaround time and Waiting time. After that we will print it in a tabular format.

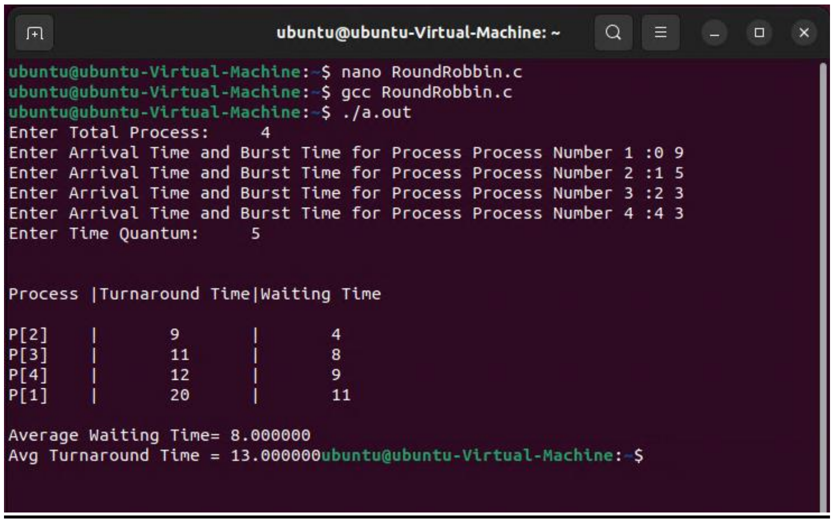
STEP5: - After that calculate and print the Average Waiting time and Average Turnaround time.

PROGRAM:

```
#include<stdio.h>

int main()
{

  int count,j,n,time,remain,flag=0,time_quantum;
  int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
  printf("Enter Total Process:\t ");
  scanf("%d",&n);
  remain=n;
  for(count=0;count<n;count++)
  {
    printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);
    scanf("%d",&at[count]);
    scanf("%d",&bt[count]);
    rt[count]=bt[count];
  }
```

```
 printf("Enter Time Quantum:\t");
 scanf("%d",&time_quantum);
 printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
 for(time=0,count=0;remain!=0;)
 {
  if(rt[count]<=time_quantum && rt[count]>0)
  {
   time+=rt[count];
   rt[count]=0;
   flag=1;
  }
  else if(rt[count]>0)
  {
   rt[count]-=time_quantum;
   time+=time_quantum;
  }
  if(rt[count]==0 && flag==1)
  {
   remain--;
   printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);
   wait_time+=time-at[count]-bt[count];
   turnaround_time+=time-at[count];
   flag=0;
  }
  if(count==n-1)
   count=0;
  else if(at[count+1]<=time)
   count++;
  else
   count=0;
 }
 printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
 printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);

 return 0;
}
```

OUTPUT:



```
ubuntu@ubuntu-Virtual-Machine:~$ nano RoundRobbin.c
ubuntu@ubuntu-Virtual-Machine:~$ gcc RoundRobbin.c
ubuntu@ubuntu-Virtual-Machine:~$ ./a.out
Enter Total Process:     4
Enter Arrival Time and Burst Time for Process Process Number 1 :0 9
Enter Arrival Time and Burst Time for Process Process Number 2 :1 5
Enter Arrival Time and Burst Time for Process Process Number 3 :2 3
Enter Arrival Time and Burst Time for Process Process Number 4 :4 3
Enter Time Quantum:     5


Process |Turnaround Time|Waiting Time

P[2]    |       9       |       4
P[3]    |       11      |       8
P[4]    |       12      |       9
P[1]    |       20      |       11

Average Waiting Time= 8.000000
Avg Turnaround Time = 13.000000ubuntu@ubuntu-Virtual-Machine:~$
```

RESULT:

The given program is compiled and executed successfully on Ubuntu OS