Operating System

Course Instructor: Muhammad Ahsan
Lab Instructor: Muhammad Usman

Session: Fall 2018
School of Systems and Technology
UMT Lahore Pakistan

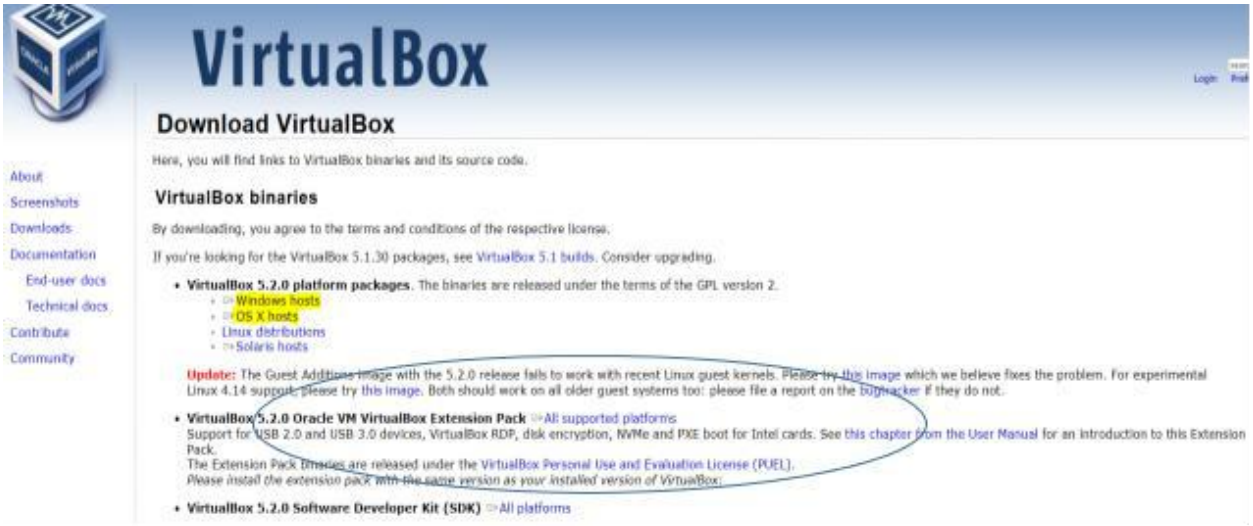
LAB 00

Virtual Machine Installation

Setting up Ubuntu with VirtualBox

Following is an install guide for setting up VirtualBox with Ubuntu 16.04.3 on your system. If you have problems, more detailed instruction and troubleshooting tips can be found on the Ubuntu site.

1. Download the version of VirtualBox (https://www.virtualbox.org/wiki/Downloads) for your machine (under "VirtualBox platform packages", choose the host package that corresponds to your operating system (i.e. if you're installing on Mac, choose the package "VirtualBox 5.2.0 for OS X hosts", if you're installing on Windows, choose the package "VirutalBox 5.2.0 for Windows Hosts).



Download VirtualBox

Here, you will find links to VirtualBox binaries and its source code.

VirtualBox binaries

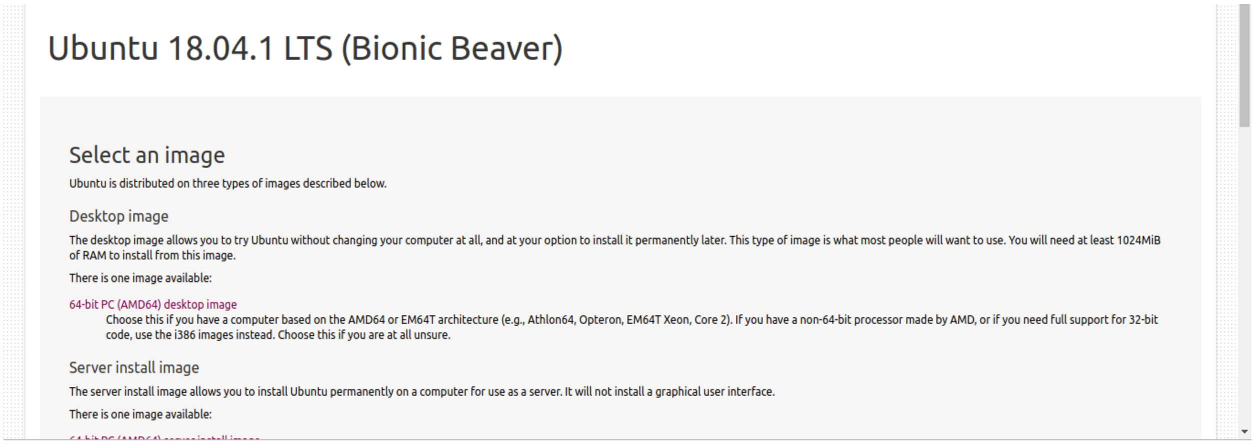
By downloading, you agree to the terms and conditions of the respective license.

If you're looking for the VirtualBox 5.1.30 packages, see VirtualBox 5.1 builds. Consider upgrading.

- **VirtualBox 5.2.0 platform packages**. The binaries are released under the terms of the GPL version 2.
  - Windows hosts
  - OS X hosts
  - Linux distributions
  - Solaris hosts

  Update: The Guest Additions image with the 5.2.0 release fails to work with recent Linux guest kernels. Please try this image which we believe fixes the problem. For experimental Linux 4.14 support, please try this image. Both should work on all older guest systems too: please file a report on the bugtracker if they do not.
- **VirtualBox 5.2.0 Oracle VM VirtualBox Extension Pack** All supported platforms
  Support for USB 2.0 and USB 3.0 devices, VirtualBox RDP, disk encryption, NVMe and PXE boot for Intel cards. See this chapter from the User Manual for an introduction to this Extension Pack.
  The Extension Pack binaries are released under the VirtualBox Personal Use and Evaluation License (PUEL).
  Please install the extension pack with the same version as your installed version of VirtualBox.
- **VirtualBox 5.2.0 Software Developer Kit (SDK)** All platforms

2. Download the 64 bit version of Ubuntu Linux 18.04 LTS (http://releases.ubuntu.com/releases/).
3. If your system has less than 2GB RAM select the 32 bit version



Ubuntu 18.04.1 LTS (Bionic Beaver)

Select an image

Ubuntu is distributed on three types of images described below.

Desktop image

The desktop image allows you to try Ubuntu without changing your computer at all, and at your option to install it permanently later. This type of image is what most people will want to use. You will need at least 1024MiB of RAM to install from this image.

There is one image available:

64-bit PC (AMD64) desktop image

Choose this if you have a computer based on the AMD64 or EM64T architecture (e.g., Athlon64, Opteron, EM64T Xeon, Core 2). If you have a non-64-bit processor made by AMD, or if you need full support for 32-bit code, use the i386 images instead. Choose this if you are at all unsure.

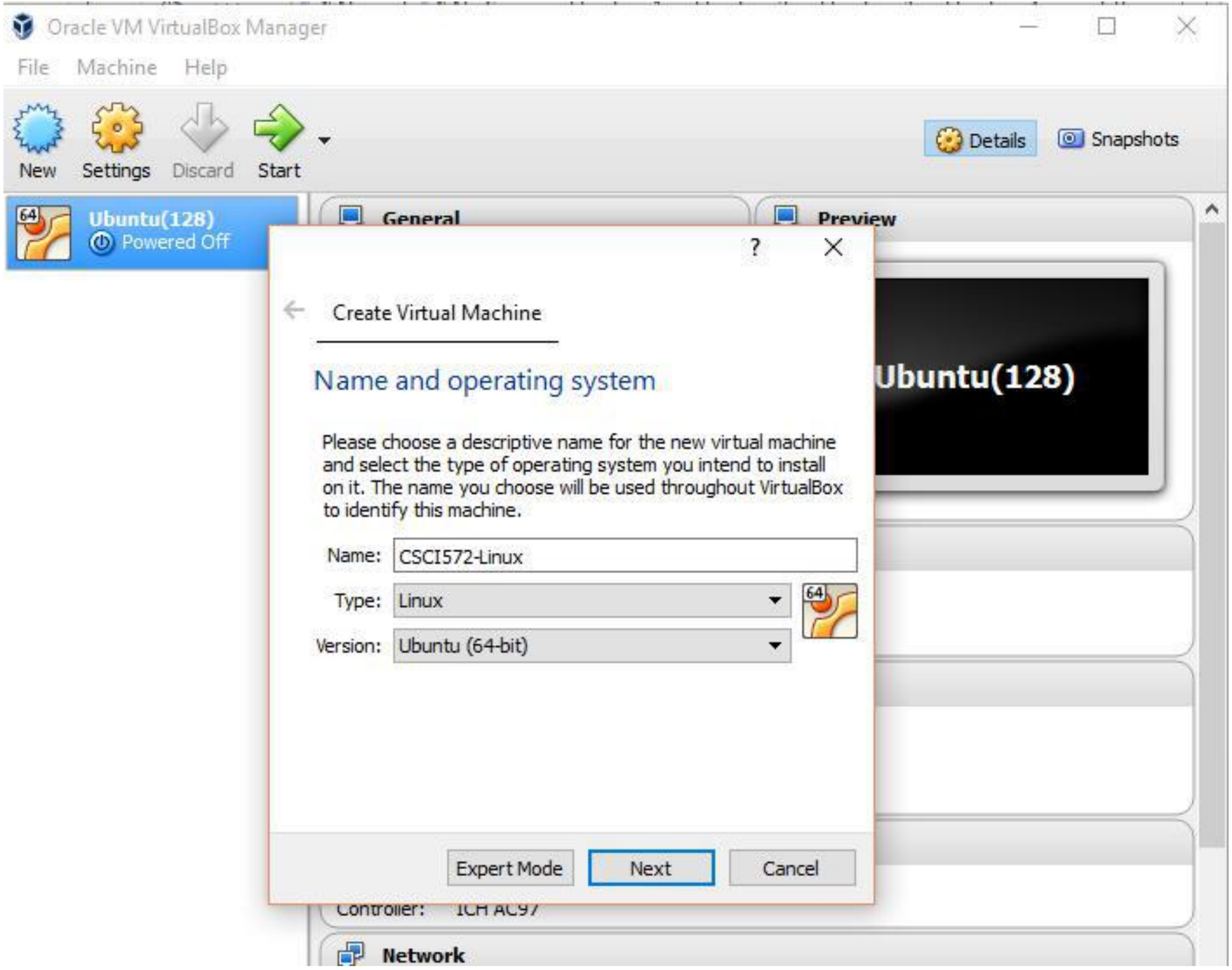
Server install image

The server install image allows you to install Ubuntu permanently on a computer for use as a server. It will not install a graphical user interface.

There is one image available:

64-bit PC (AMD64) server install image

4. Run the VirtualBox-5.2.0-118431-Win.exe file and follow the installer wizard
5. After the installation, open the VirtualBox applications
6. Select "New" from the application ribbon, choose a name for your system, and select Type: Linux and Version Ubuntu (64bit). Remember to select Version: Ubuntu (64-bit)



Name and operating system

Please choose a descriptive name for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

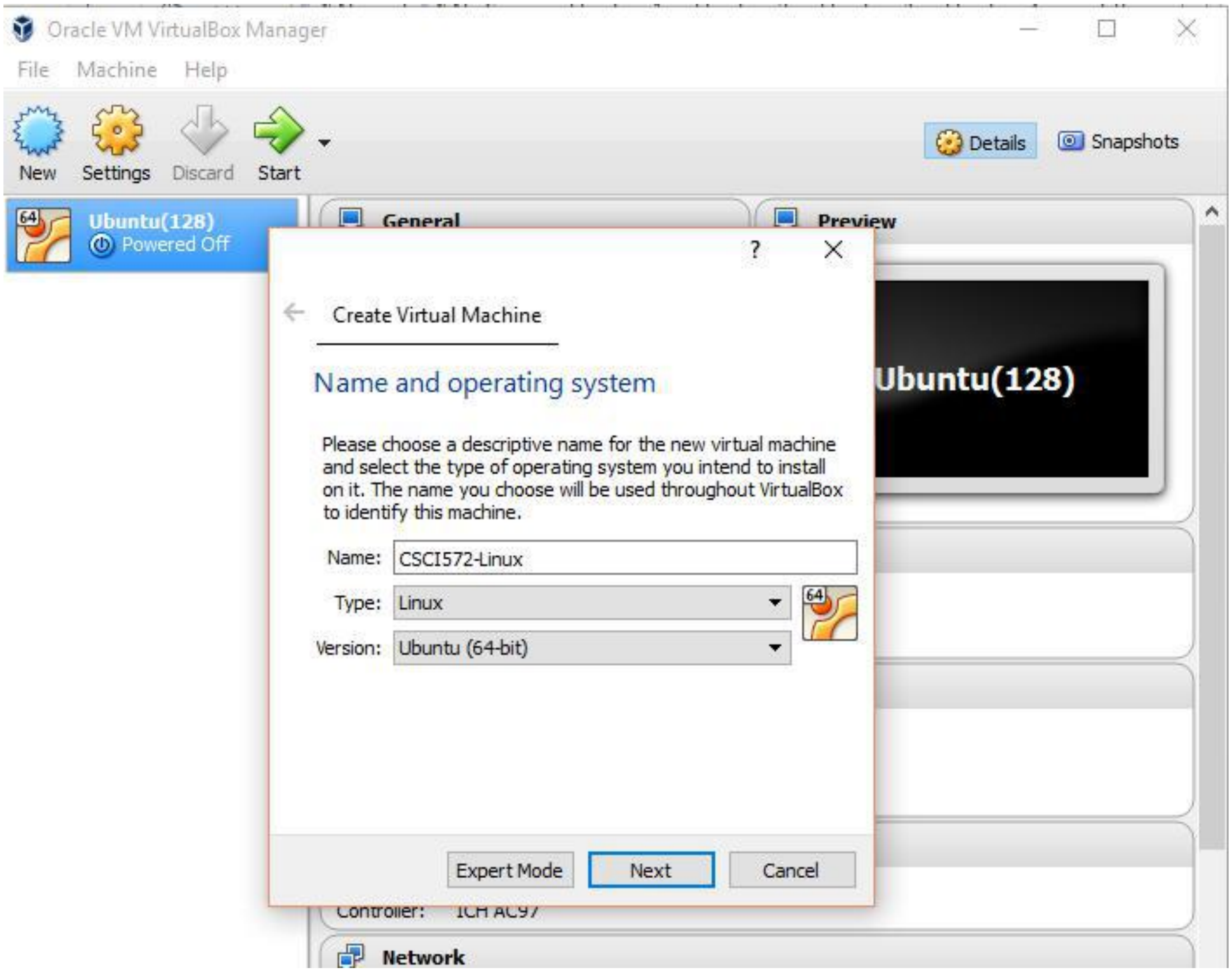
Name: CSCI572-Linux

Type: Linux

Version: Ubuntu (64-bit)

Controller: ICH AC97

Select the amount of memory for your virtual machine (If you have 4GB of RAM or more, generally set this to 2048MB or half your system RAM, whichever is greater).



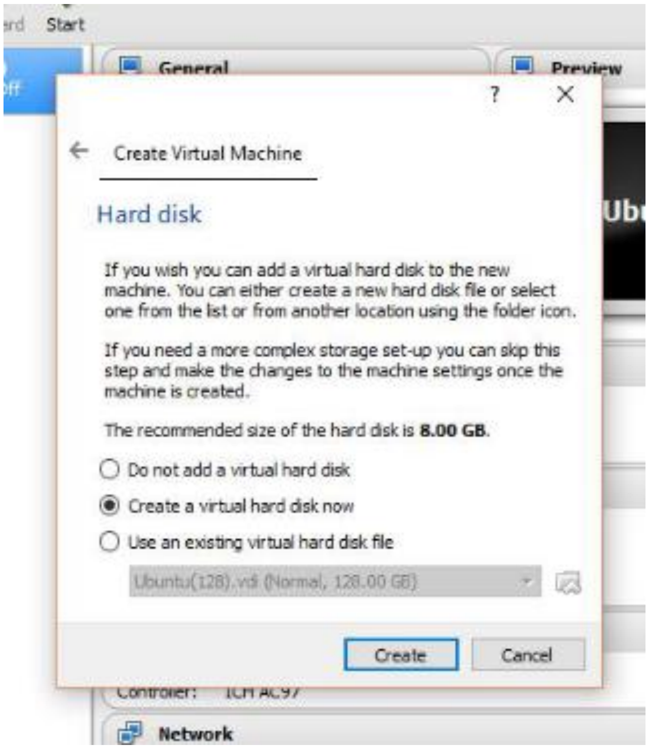
Create Virtual Machine

Name and operating system

Please choose a descriptive name for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name: CSCI572-Linux
Type: Linux
Version: Ubuntu (64-bit)

Select the “Create a virtual hard drive now” option: Note: your grayed area may say Empty


Create Virtual Machine

Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard disk is **8.00 GB**.

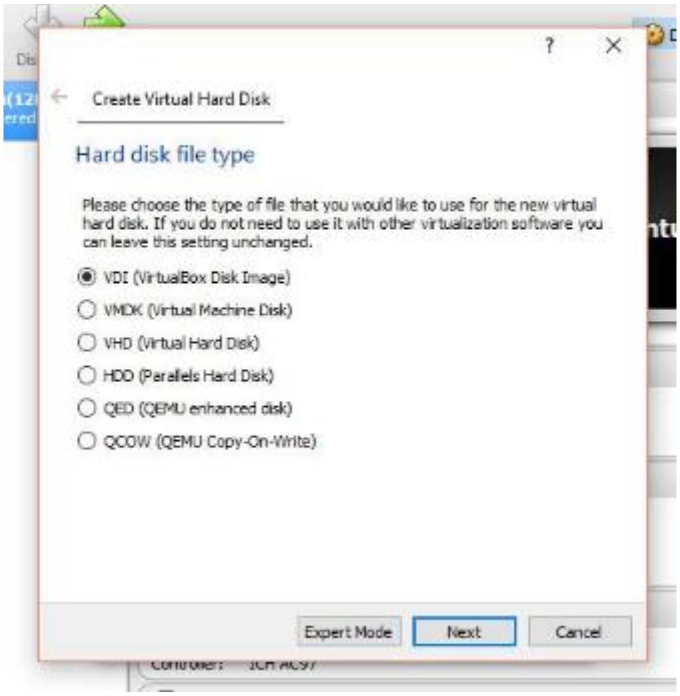
○ Do not add a virtual hard disk

● Create a virtual hard disk now

○ Use an existing virtual hard disk file

Ubuntu(128).vdi (Normal, 128.00 GB)

Select the “VDI (VirtualBox Disk Image)” option for Hard disk File Type


Create Virtual Hard Disk

Hard disk file type

Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

● VDI (VirtualBox Disk Image)

○ VMDK (Virtual Machine Disk)

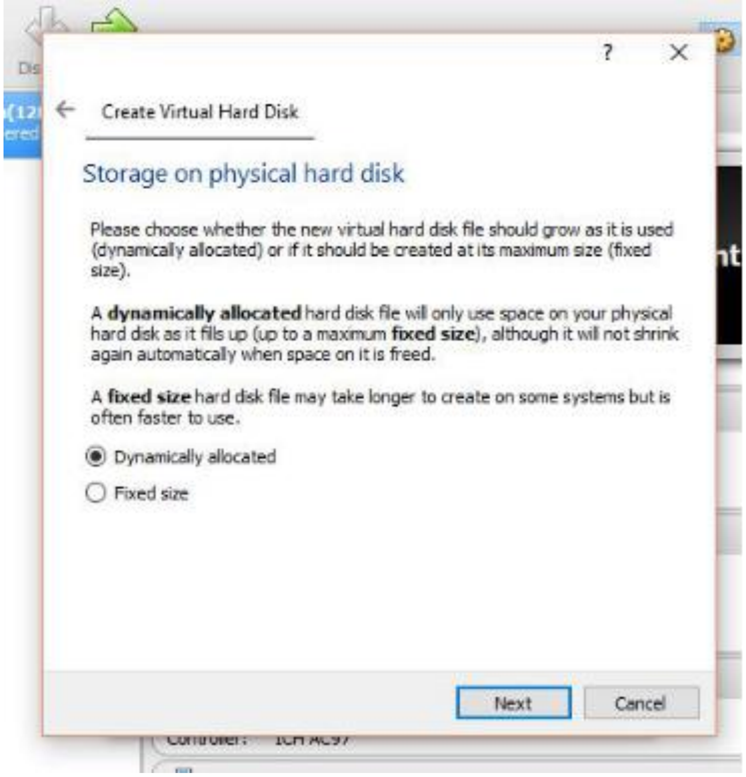
○ VHD (Virtual Hard Disk)

○ HDD (Parallels Hard Disk)

○ QED (QEMU enhanced disk)

○ QCOW (QEMU Copy-On-Write)

- Select “Dynamically allocated” for Storage on physical hard disk



Create Virtual Hard Disk

Storage on physical hard disk

Please choose whether the new virtual hard disk file should grow as it is used (dynamically allocated) or if it should be created at its maximum size (fixed size).

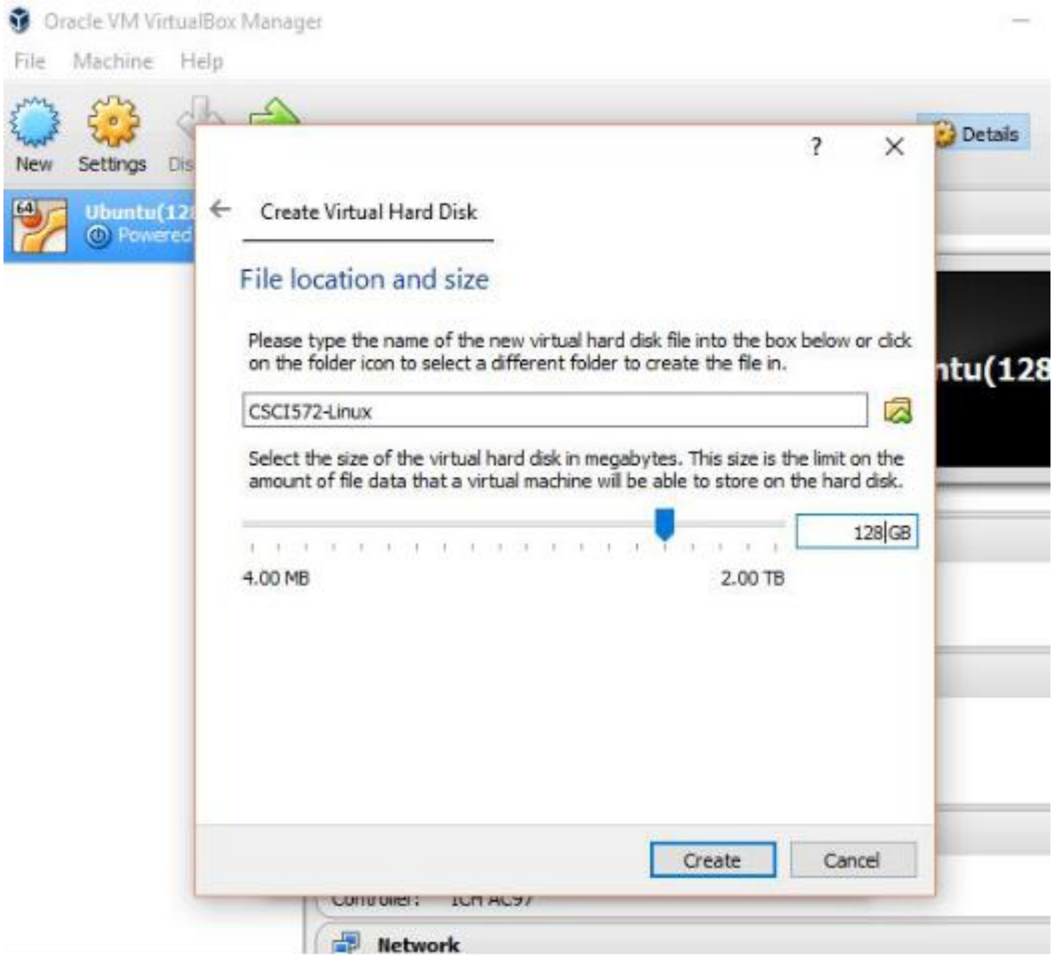
A **dynamically allocated** hard disk file will only use space on your physical hard disk as it fills up (up to a maximum **fixed size**), although it will not shrink again automatically when space on it is freed.

A **fixed size** hard disk file may take longer to create on some systems but is often faster to use.

● Dynamically allocated

○ Fixed size

Select the starting drive size (it is recommended to allot at least 128GB). Note: you may be unable to get exactly 128GB.



Create Virtual Hard Disk

File location and size

Please type the name of the new virtual hard disk file into the box below or click on the folder icon to select a different folder to create the file in.

CSCI572-Linux

Select the size of the virtual hard disk in megabytes. This size is the limit on the amount of file data that a virtual machine will be able to store on the hard disk.

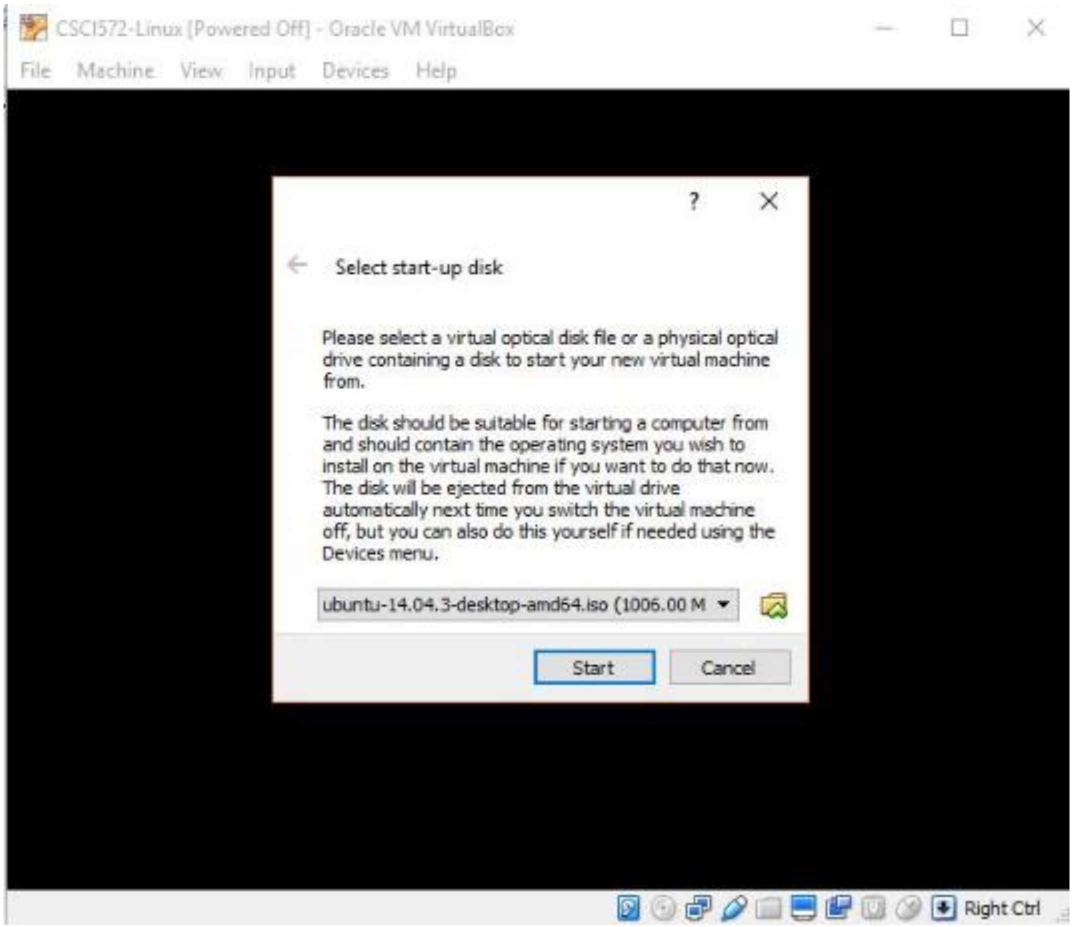
128 GB

4.00 MB

2.00 TB

9. With your new instance selected, select start from the application ribbon.

10. When prompted, select the previously downloaded Ubuntu iso file as the virtual optical disk file by clicking on the folder icon and click on Start


CSCI572-Linux [Powered Off] - Oracle VM VirtualBox
File Machine View Input Devices Help
? ×
Start
Cancel
Right Ctrl

Select start-up disk

Please select a virtual optical disk file or a physical optical drive containing a disk to start your new virtual machine from.

The disk should be suitable for starting a computer from and should contain the operating system you wish to install on the virtual machine if you want to do that now. The disk will be ejected from the virtual drive automatically next time you switch the virtual machine off, but you can also do this yourself if needed using the Devices menu.

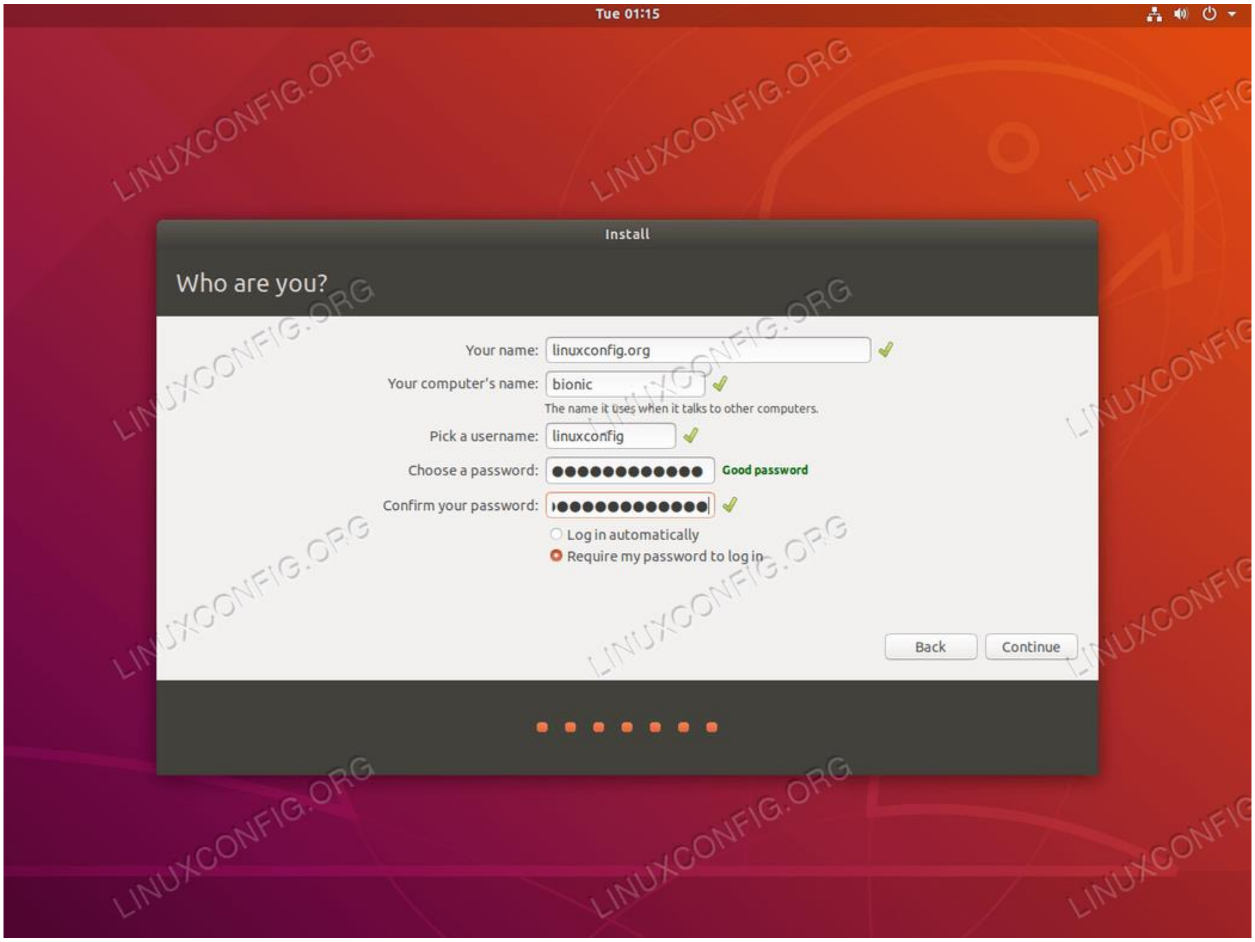
ubuntu-14.04.3-desktop-amd64.iso (1006.00 M

11. Click on Install Ubuntu

Click on Continue and select Erase disk and install Ubuntu and click on Install Now. Messages may appear indicating Auto capture of keyboard and mouse pointer. The messages can be removed

12. Follow the prompts to install Ubuntu. Select Location and Language. Enter your username and

Password for the Ubuntu system. Select either Log in automatically if you want to log in without password when the Ubuntu machine is started from VirtualBox



Who are you?

Your name: linuxconfig.org ✓

Your computer's name: bionic ✓

The name it uses when it talks to other computers.

Pick a username: linuxconfig ✓

Choose a password: ●●●●●●●●●●●● Good password

Confirm your password: ●●●●●●●●●●●●● ✓

○ Log in automatically

● Require my password to log in

Ubuntu Installation will begin after you click on Continue. This will take a while, be patient

LAB 01

Linux Commands

1. ls:-

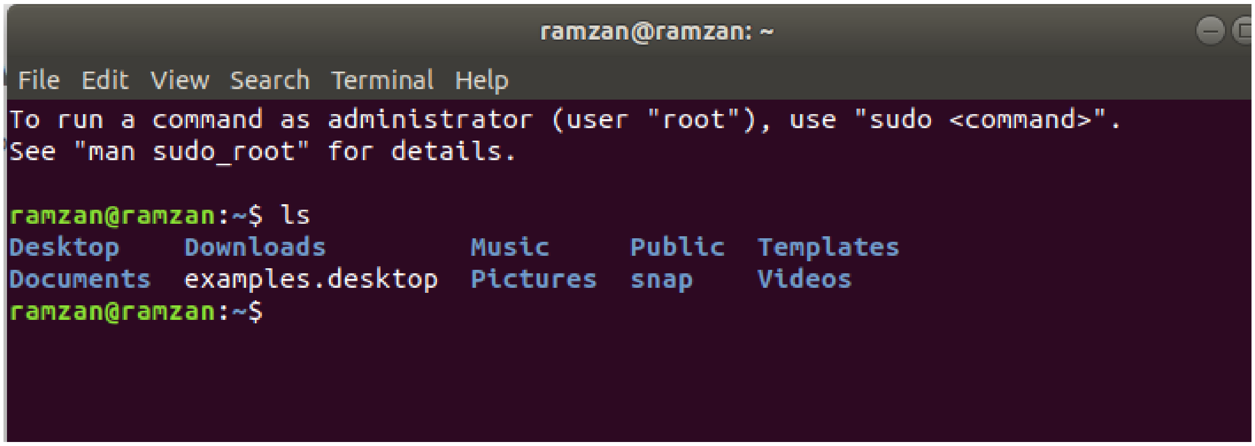
Parameters:-

-a, -all, /

Description:-

Without arguments, lists the files and directories names in the current directory.
$ ls / : Lists the contents of the directory given as an argument.
$ ls –a /home/student : Includes so-called “hidden” files and directories whose names begin with a dot (.).
$ ls [options] [files_or_directories]: Lists the contents of the current directory or a specified directory.
$ ls –all: Lists files and directories with detailed information like permissions, size, owner, etc.

Screenshot:-



```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ramzan@ramzan:~$ ls
Desktop    Downloads         Music     Public   Templates
Documents  examples.desktop  Pictures  snap     Videos
ramzan@ramzan:~$
```

2. date:-

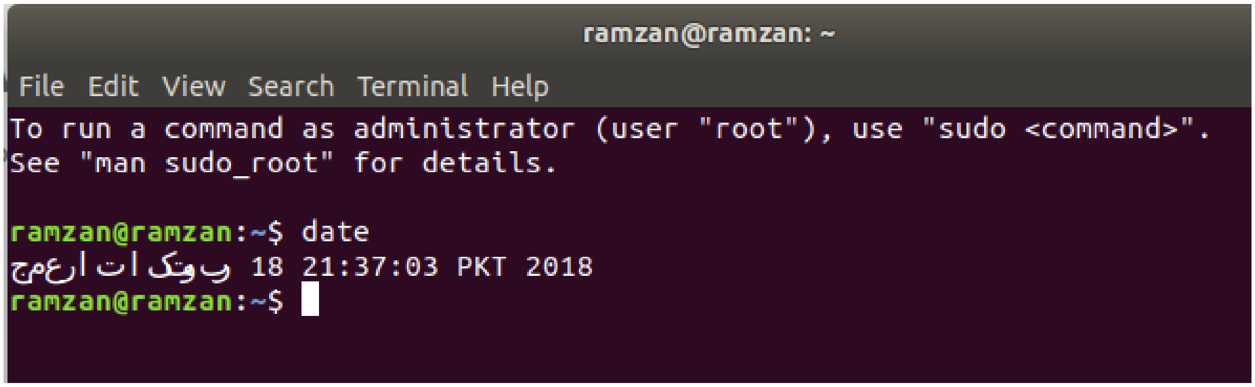
Parameters:-

N.A.

Description:-

Prints the system date and time.

Screenshot:-



```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ramzan@ramzan:~$ date
جمعرات اکتوبر 18 21:37:03 PKT 2018
ramzan@ramzan:~$ 
```

3. *cal:-*

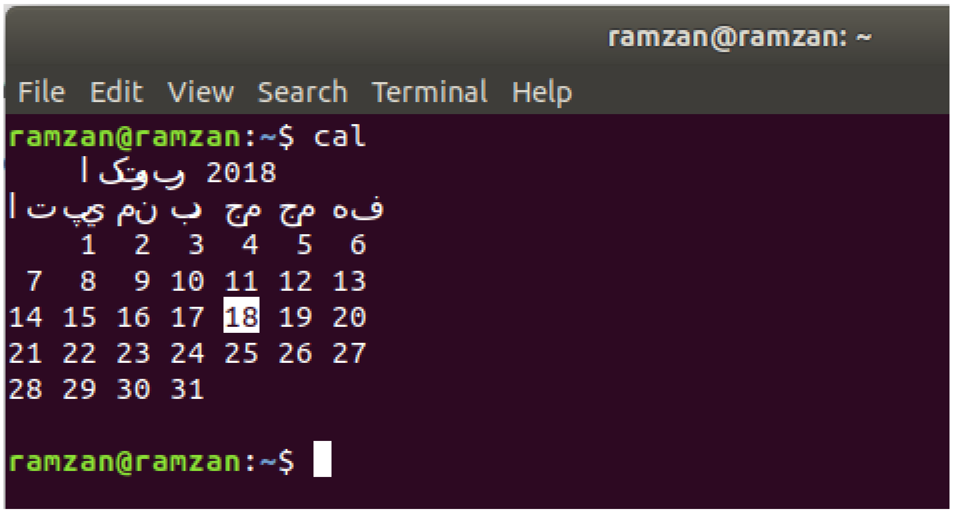
Parameters:-

N.A.

Description:-

Prints the ASCII calendar of the current month.

Screenshot:-



```
ramzan@ramzan:~$ cal
    اکتوبر 2018
ات پي من ب جم جم هف
    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 30 31

ramzan@ramzan:~$ 
```

4. pwd:-

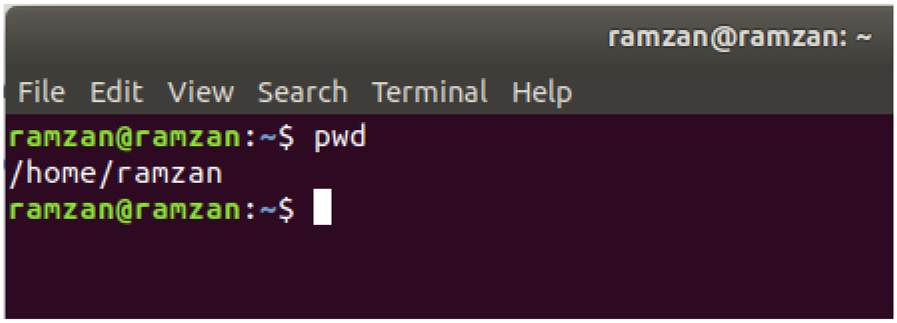
Parameters:-

N.A.

Description:-

Displays the absolute path to the current working directory.

Screenshot:-



```
ramzan@ramzan:~$ pwd
/home/ramzan
ramzan@ramzan:~$ 
```

5. cd:-

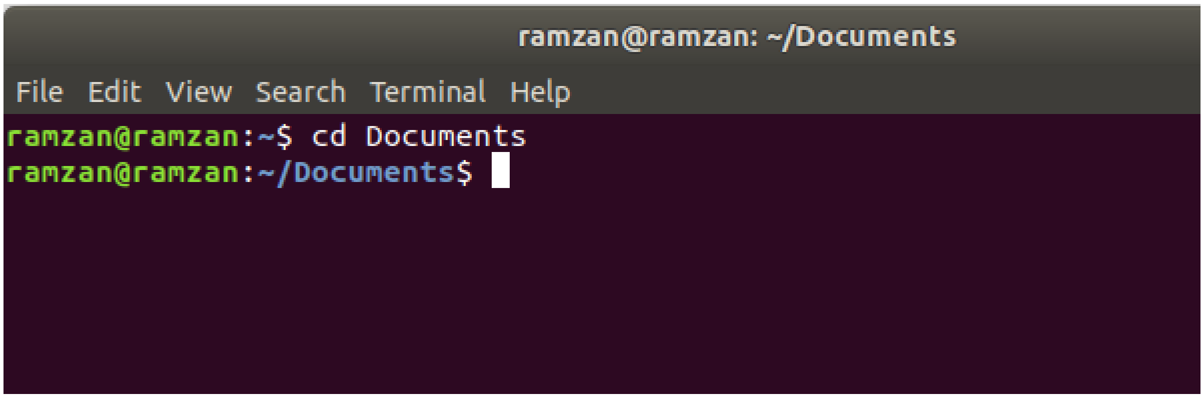
Parameters:-

.., ~, -

Description:-

$ cd: Changes directories.
$ cd /home/uet/cs: To an absolute path.
$ project /docs: To a relative path.
$ cd .. : To a directory one level up.
$ cd ~ : T a directory one level up.
$ cd - : To your previous working directory.

Screenshot:-


ramzan@ramzan: ~/Documents
File Edit View Search Terminal Help

```
ramzan@ramzan:~$ cd Documents
ramzan@ramzan:~/Documents$
```

6. mkdir:-

Parameters:-

N.A.

Description:-

Directories can be created on a Linux operating system using the following command:-

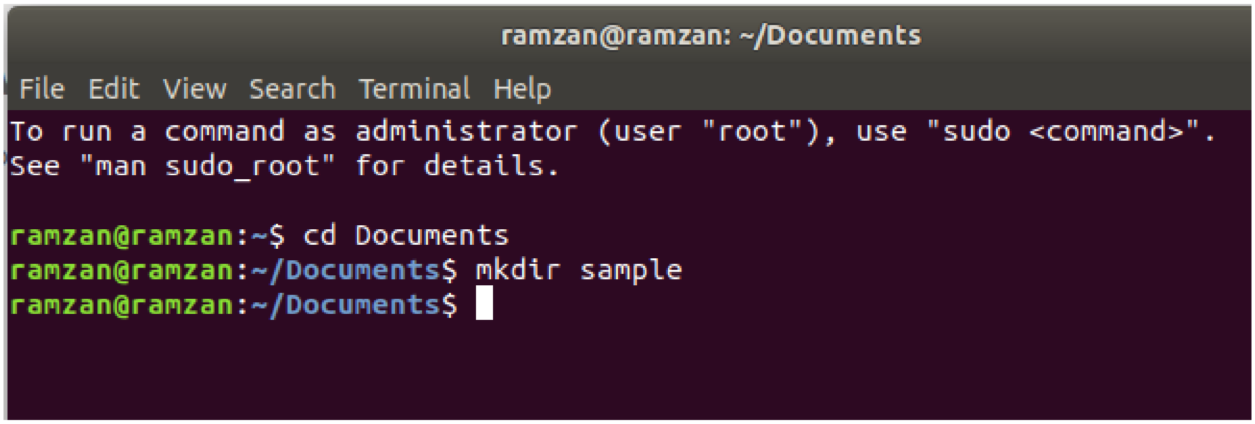
mkdir directoryname

This command will create a subdirectory in your present working directory, which is usually your "Home Directory".

For example,

mkdir mydirectory

Screenshot:-


ramzan@ramzan: ~/Documents
File Edit View Search Terminal Help

```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ramzan@ramzan:~$ cd Documents
ramzan@ramzan:~/Documents$ mkdir sample
ramzan@ramzan:~/Documents$
```

7. rm:-

Parameters:-

N.A.

Description:-

To remove a file, use the command -

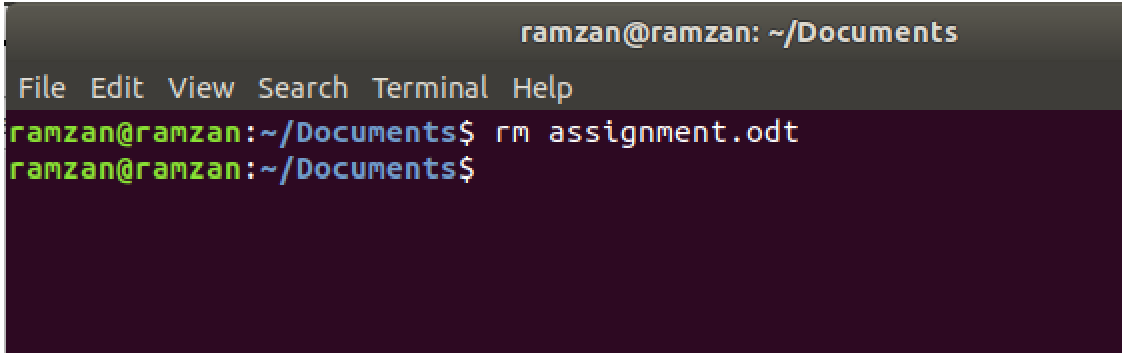
rm filename

Example

Rm assignment.odt

will delete the directory mydirectory

Screenshot:-



ramzan@ramzan: ~/Documents
File Edit View Search Terminal Help

```
ramzan@ramzan:~/Documents$ rm assignment.odt
ramzan@ramzan:~/Documents$
```

8. mv:-

Parameters:-

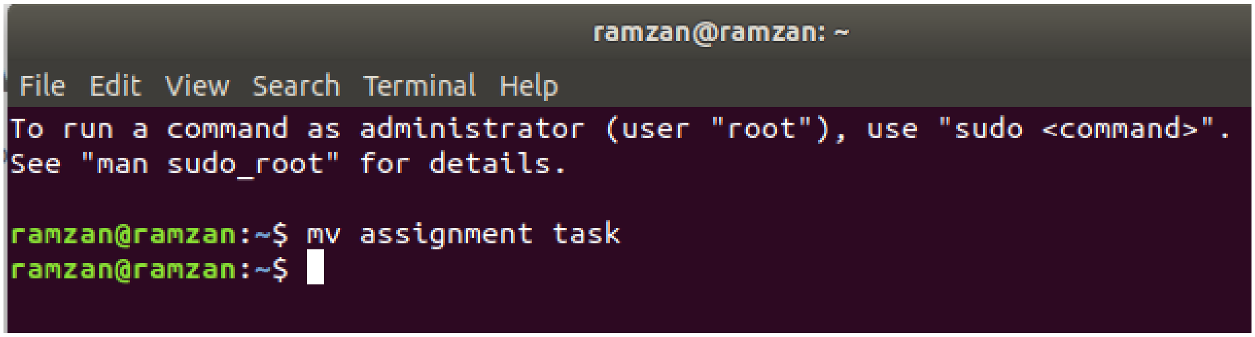
N.A.

Description:-

The 'mv' (move) command can also be used for renaming directories. Use the below-given format:

mv directoryname newdirectoryname

Screenshot:-



```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ramzan@ramzan:~$ mv assignment task
ramzan@ramzan:~$ 
```

9. man:-

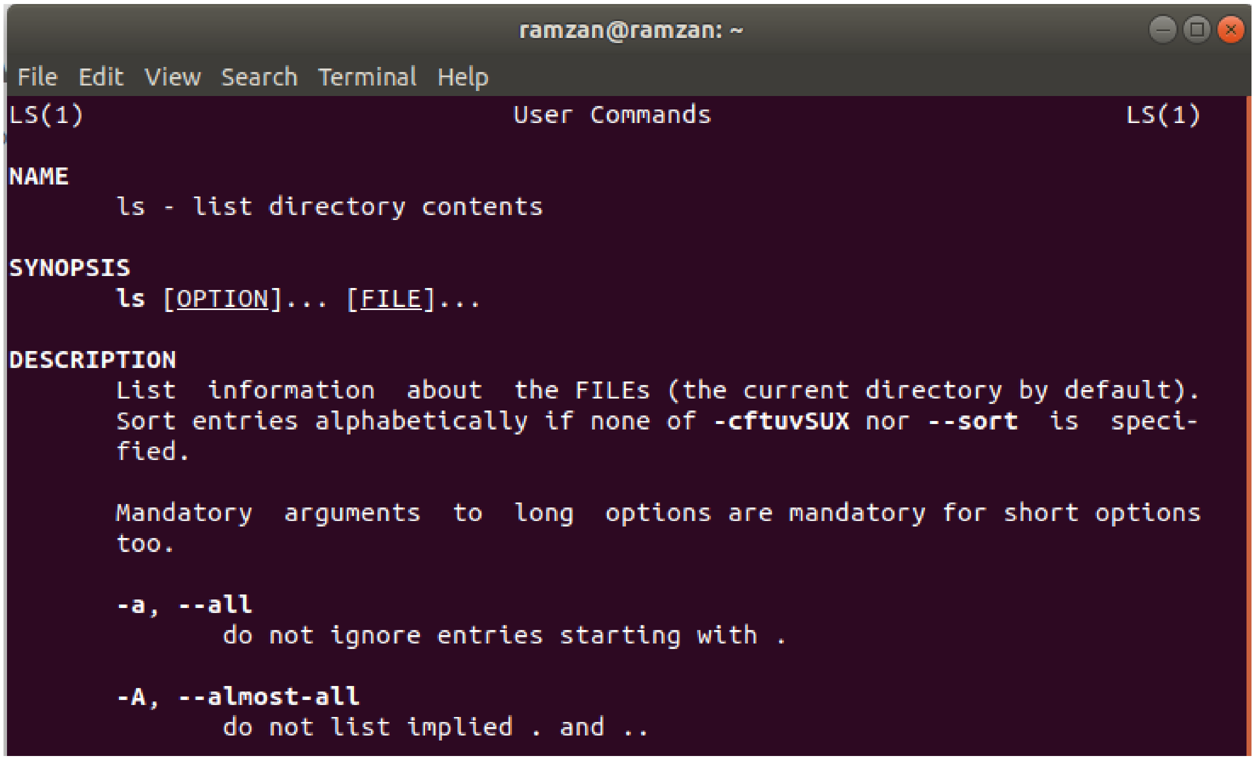
Parameters:-

N.A.

Description:-

Man stands for manual which is a reference book of a Linux operating system. It is similar to HELP file found in popular software.
To get help on any command that you do not understand, you can type
man
The terminal would open the manual page for that command.
For an example, if we type man man and hit enter; terminal would give us information on man command
$ man man

Screenshot:-



```
LS(1)                           User Commands                          LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).
       Sort entries alphabetically if none of -cftuvSUX nor --sort  is  speci-
       fied.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..
```

10. history:-

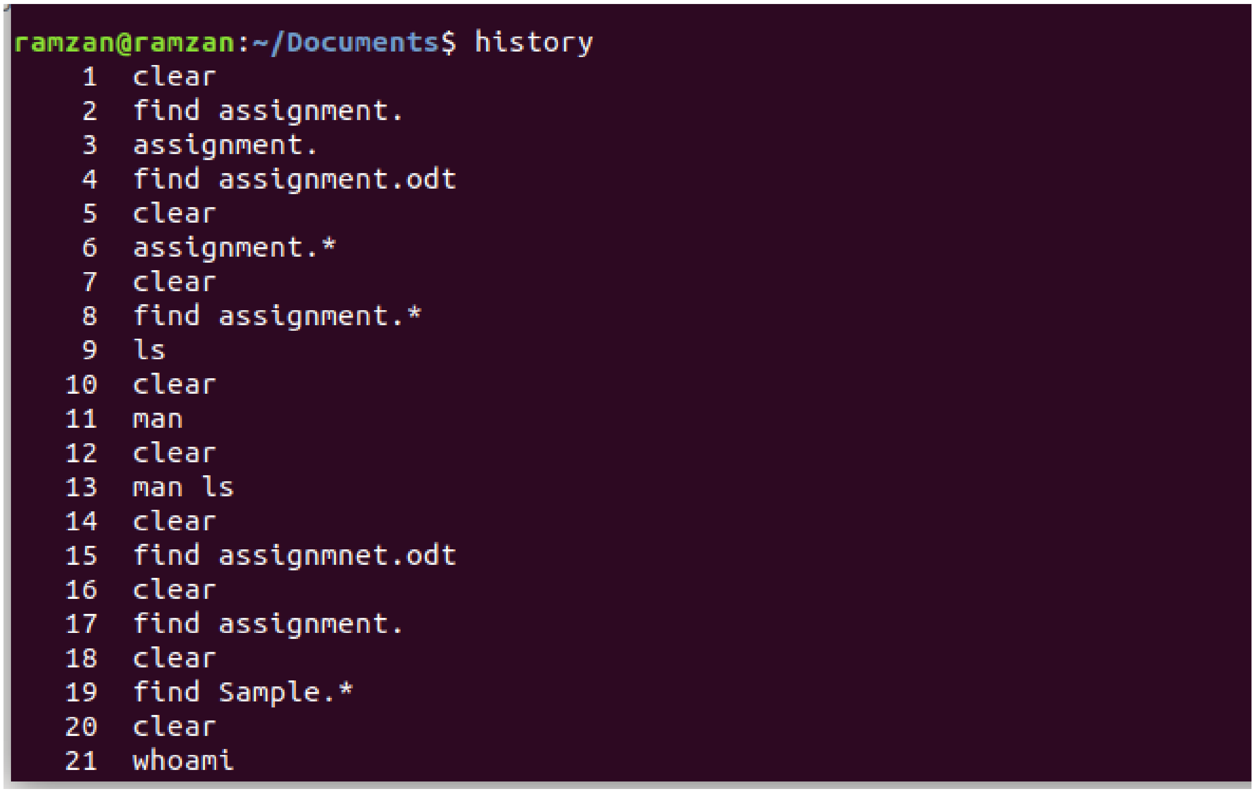
Parameters:-

N.A.

Description:-

History command shows all the commands that you have used in the past for the current terminal session. This can help you refer to the old commands you have entered and re-used them in your operations again.

Screenshot:-



```
ramzan@ramzan:~/Documents$ history
    1  clear
    2  find assignment.
    3  assignment.
    4  find assignment.odt
    5  clear
    6  assignment.*
    7  clear
    8  find assignment.*
    9  ls
   10  clear
   11  man
   12  clear
   13  man ls
   14  clear
   15  find assignmnet.odt
   16  clear
   17  find assignment.
   18  clear
   19  find Sample.*
   20  clear
   21  whoami
```

11. clear:-

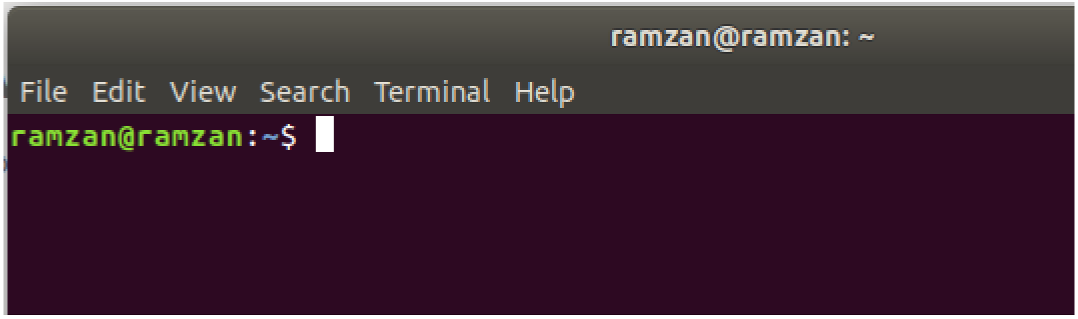
Parameters:-

N.A.

Description:-

This command clears all the clutter on the terminal and gives you a clean window to work on, just like when you launch the terminal.

Screenshot:-



```
ramzan@ramzan:~$ 
```

12. df:-

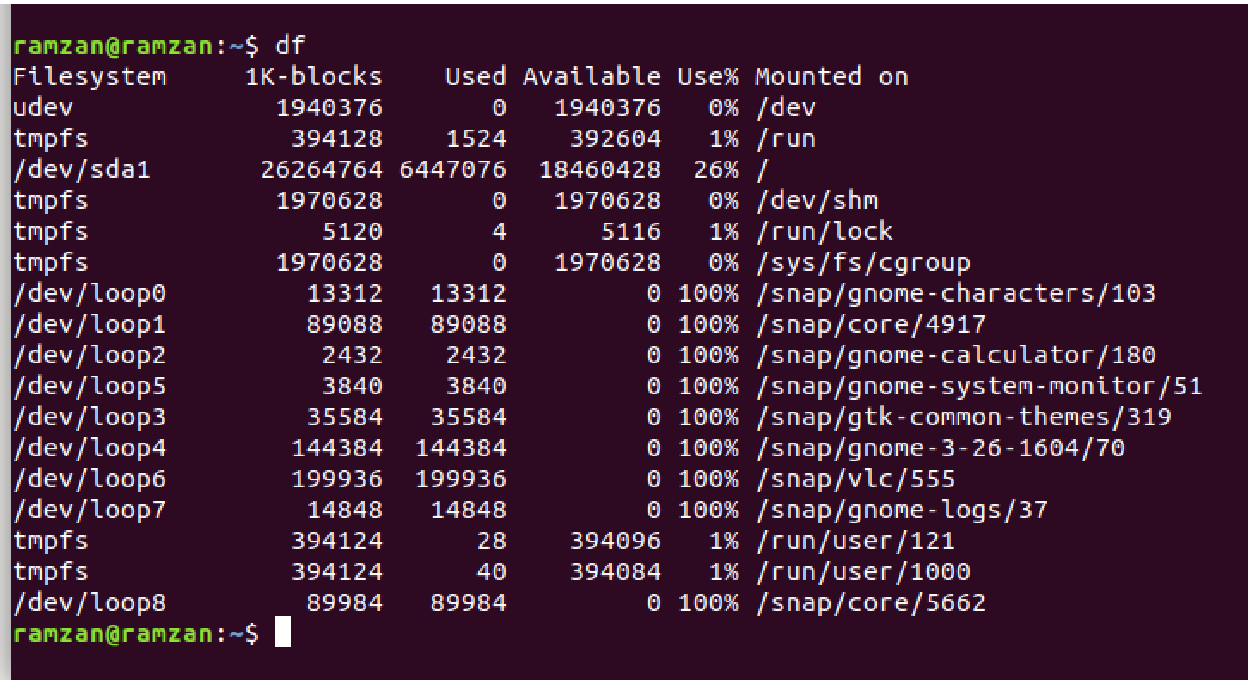
Parameters:-

N.A.

Description:-

Display free disk space.

Screenshot:-



```
ramzan@ramzan:~$ df
Filesystem     1K-blocks    Used Available Use% Mounted on
udev             1940376       0   1940376   0% /dev
tmpfs             394128    1524    392604   1% /run
/dev/sda1       26264764 6447076  18460428  26% /
tmpfs            1970628       0   1970628   0% /dev/shm
tmpfs               5120       4      5116   1% /run/lock
tmpfs            1970628       0   1970628   0% /sys/fs/cgroup
/dev/loop0         13312   13312         0 100% /snap/gnome-characters/103
/dev/loop1         89088   89088         0 100% /snap/core/4917
/dev/loop2          2432    2432         0 100% /snap/gnome-calculator/180
/dev/loop5          3840    3840         0 100% /snap/gnome-system-monitor/51
/dev/loop3         35584   35584         0 100% /snap/gtk-common-themes/319
/dev/loop4        144384  144384         0 100% /snap/gnome-3-26-1604/70
/dev/loop6        199936  199936         0 100% /snap/vlc/555
/dev/loop7         14848   14848         0 100% /snap/gnome-logs/37
tmpfs             394124      28    394096   1% /run/user/121
tmpfs             394124      40    394084   1% /run/user/1000
/dev/loop8         89984   89984         0 100% /snap/core/5662
ramzan@ramzan:~$ 
```

13. echo:-

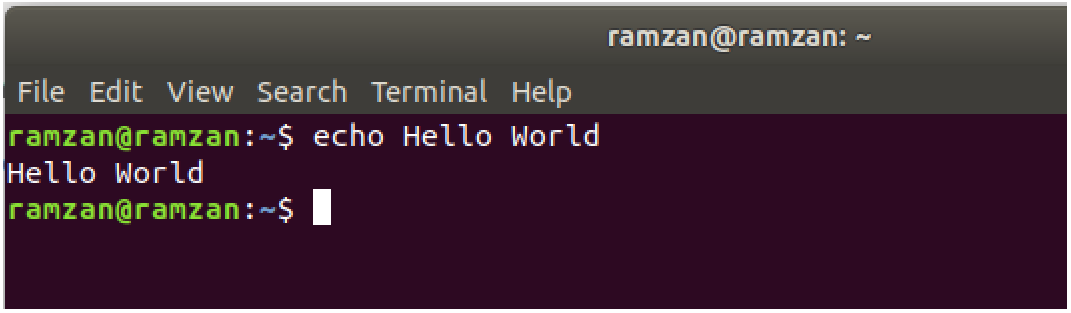
Parameters:-

N.A.

Description:-

Display message on screen.

Screenshot:-



```
ramzan@ramzan:~$ echo Hello World
Hello World
ramzan@ramzan:~$ 
```

14. free:-

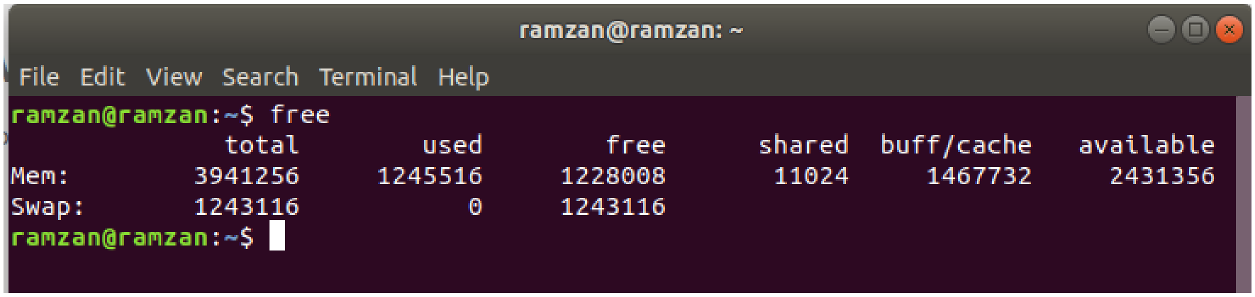
Parameters:-

-h, -m, -g

Description:-

Display memory usage.

Screenshot:-



```
ramzan@ramzan:~$ free
              total        used        free      shared  buff/cache   available
Mem:        3941256     1245516     1228008       11024     1467732     2431356
Swap:       1243116           0     1243116
ramzan@ramzan:~$ 
```

15. logname:-

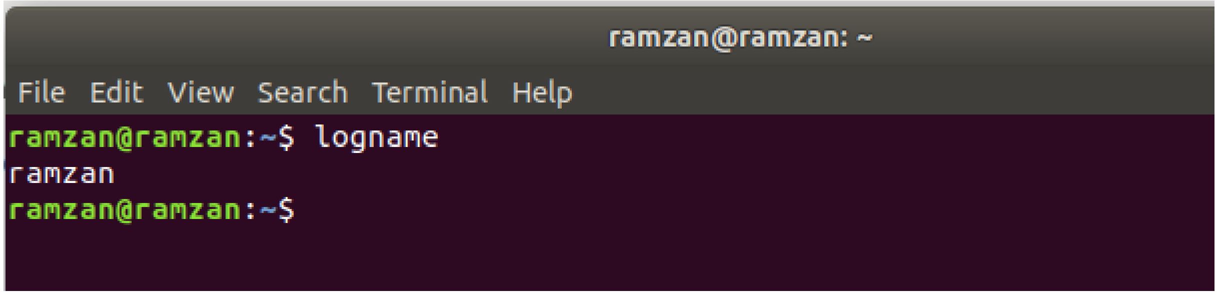
Parameters:-

N.A.

Description:-

Display memory usage.

Screenshot:-



```
ramzan@ramzan:~$ logname
ramzan
ramzan@ramzan:~$
```

16. whoami:-

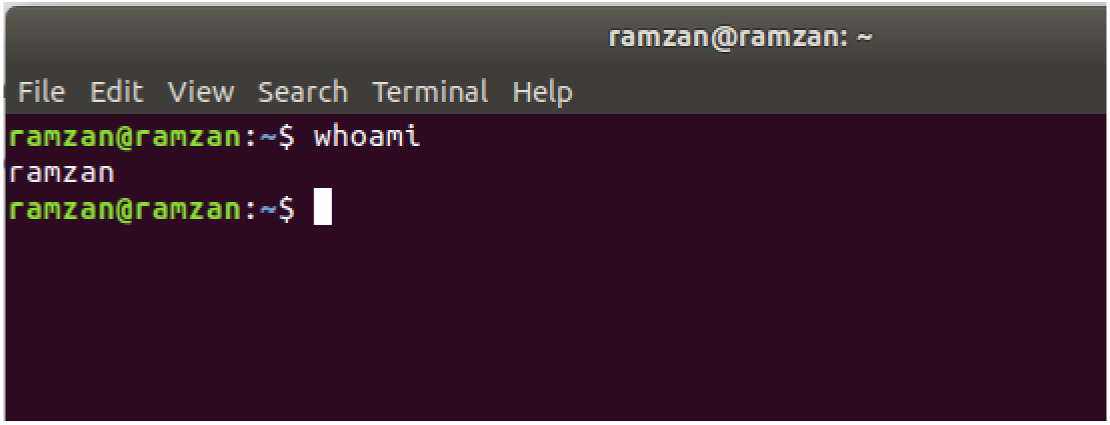
Parameters:-

N.A.

Description:-

Print the current user id and name.

Screenshot:-



```
ramzan@ramzan:~$ whoami
ramzan
ramzan@ramzan:~$
```

17. uname:-

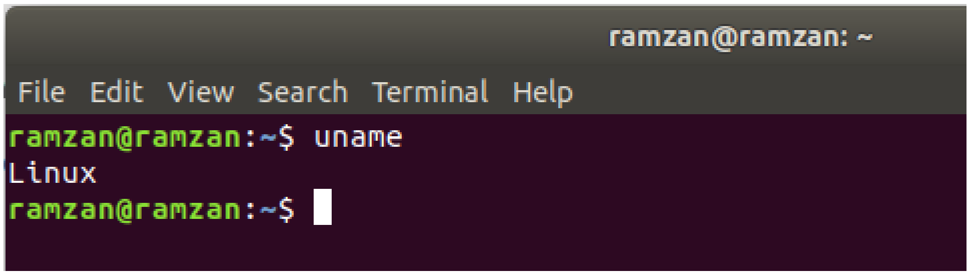
Parameters:-

-a, -r

Description:-

Print system information

Screenshot:-



```
ramzan@ramzan:~$ uname
Linux
ramzan@ramzan:~$ 
```

18. factor:-

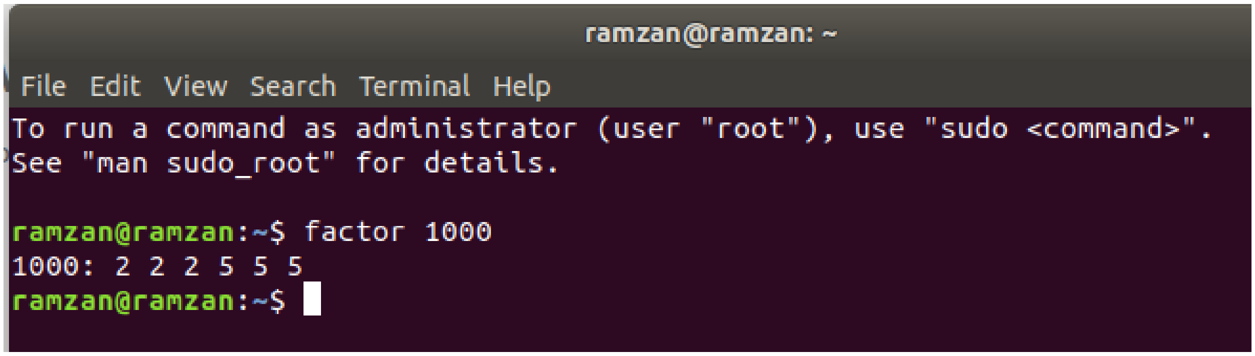
Parameters:-

N.A.

Description:-

Display prime factors of specified integer numbers.

Screenshot:-



```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ramzan@ramzan:~$ factor 1000
1000: 2 2 2 5 5 5
ramzan@ramzan:~$ 
```

19. top:-

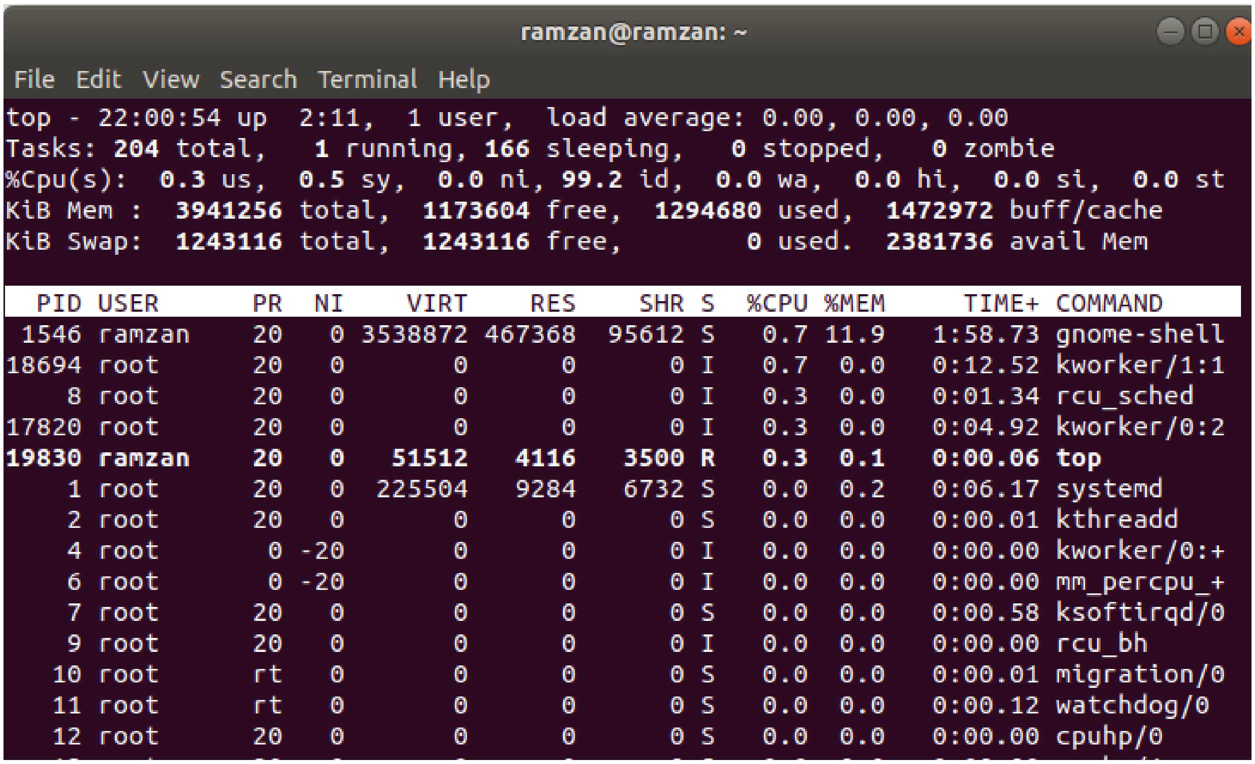
Parameters:-

N.A.

Description:-

Shows top consumers of memory and CPU.

Screenshot:-



```
top - 22:00:54 up  2:11,  1 user,  load average: 0.00, 0.00, 0.00
Tasks: 204 total,   1 running, 166 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  0.5 sy,  0.0 ni, 99.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3941256 total,  1173604 free,  1294680 used,  1472972 buff/cache
KiB Swap:  1243116 total,  1243116 free,        0 used.  2381736 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1546 ramzan    20   0 3538872 467368  95612 S   0.7 11.9   1:58.73 gnome-shell
18694 root      20   0       0      0      0 I   0.7  0.0   0:12.52 kworker/1:1
    8 root      20   0       0      0      0 I   0.3  0.0   0:01.34 rcu_sched
17820 root      20   0       0      0      0 I   0.3  0.0   0:04.92 kworker/0:2
19830 ramzan    20   0   51512   4116   3500 R   0.3  0.1   0:00.06 top
    1 root      20   0  225504   9284   6732 S   0.0  0.2   0:06.17 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.01 kthreadd
    4 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/0:+
    6 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 mm_percpu_+
    7 root      20   0       0      0      0 S   0.0  0.0   0:00.58 ksoftirqd/0
    9 root      20   0       0      0      0 I   0.0  0.0   0:00.00 rcu_bh
   10 root      rt   0       0      0      0 S   0.0  0.0   0:00.01 migration/0
   11 root      rt   0       0      0      0 S   0.0  0.0   0:00.12 watchdog/0
   12 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/0
```

20. ps:-

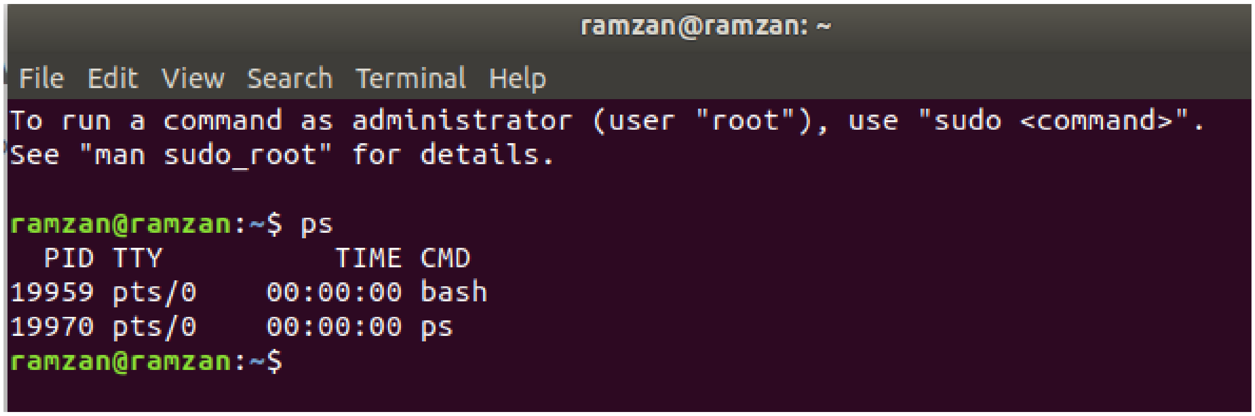
Parameters:-

-ef

Description:-

Shows processes running by user.

Screenshot:-



```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ramzan@ramzan:~$ ps
  PID TTY          TIME CMD
19959 pts/0    00:00:00 bash
19970 pts/0    00:00:00 ps
ramzan@ramzan:~$
```

21. hostname:-

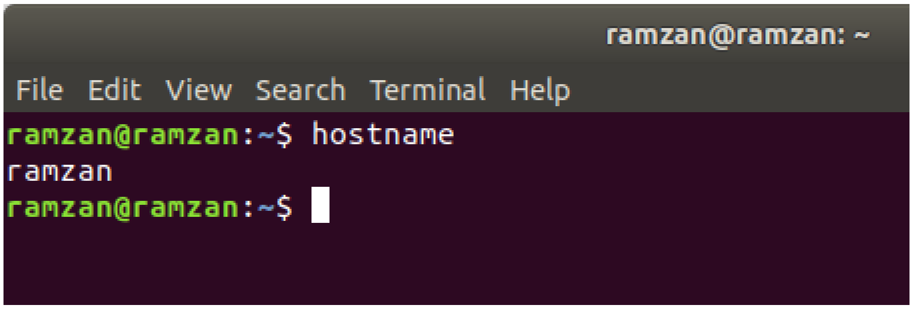
Parameters:-

-I

Description:-

Use hostname to know your name in your host or network. Basically, it displays your hostname and IP address. Just typing “hostname” gives the output. Typing in “hostname -I” gives you your IP address in your network.

Screenshot:-



```
ramzan@ramzan:~$ hostname
ramzan
ramzan@ramzan:~$
```

22. arch:-

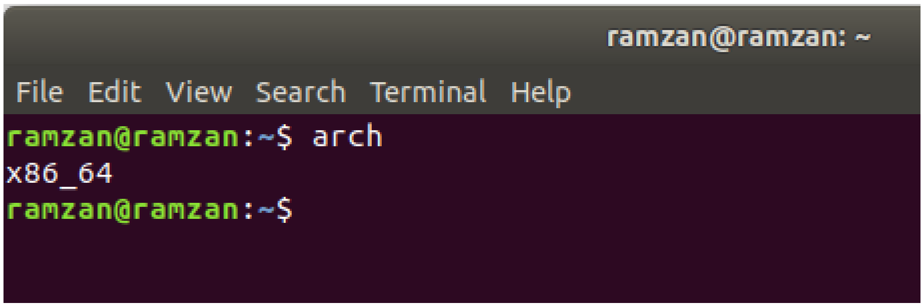
Parameters:-

N.A.

Description:-

The arch command is used to print the machine’s architecture.

Screenshot:-



```
ramzan@ramzan:~$ arch
x86_64
ramzan@ramzan:~$
```

23. uptime:-

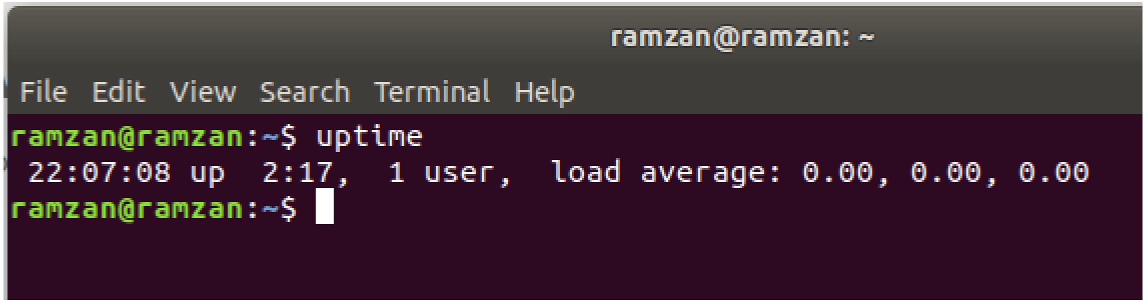
Parameters:-

N.A.

Description:-

Shows how long the system has been running + load.

Screenshot:-



```
ramzan@ramzan:~$ uptime
 22:07:08 up  2:17,  1 user,  load average: 0.00, 0.00, 0.00
ramzan@ramzan:~$
```

24. cat:-

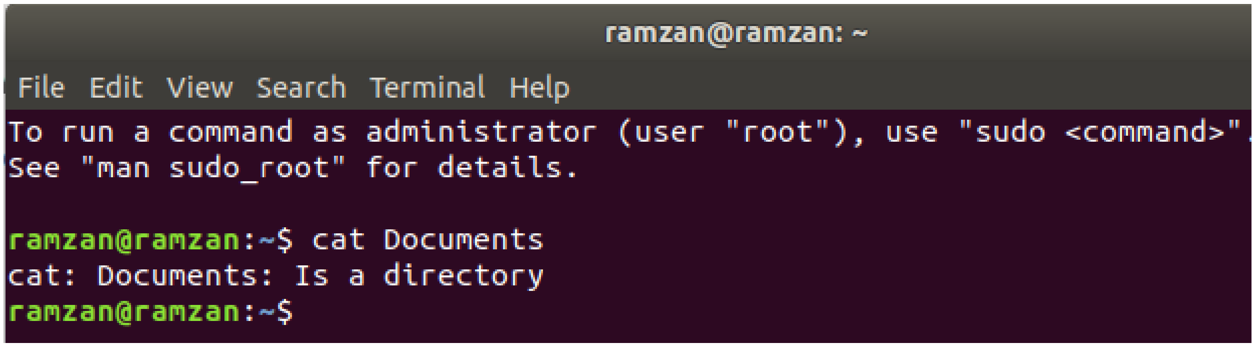
Parameters:-

N.A.

Description:-

View the contents of file.

Screenshot:-



```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ramzan@ramzan:~$ cat Documents
cat: Documents: Is a directory
ramzan@ramzan:~$
```

25. tty:-

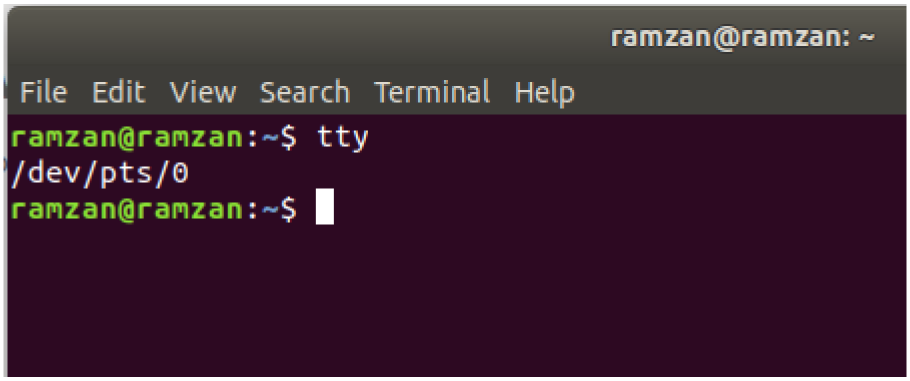
Parameters:-

N.A.

Description:-

Displays current terminal.

Screenshot:-



```
ramzan@ramzan:~$ tty
/dev/pts/0
ramzan@ramzan:~$
```

26. touch:-

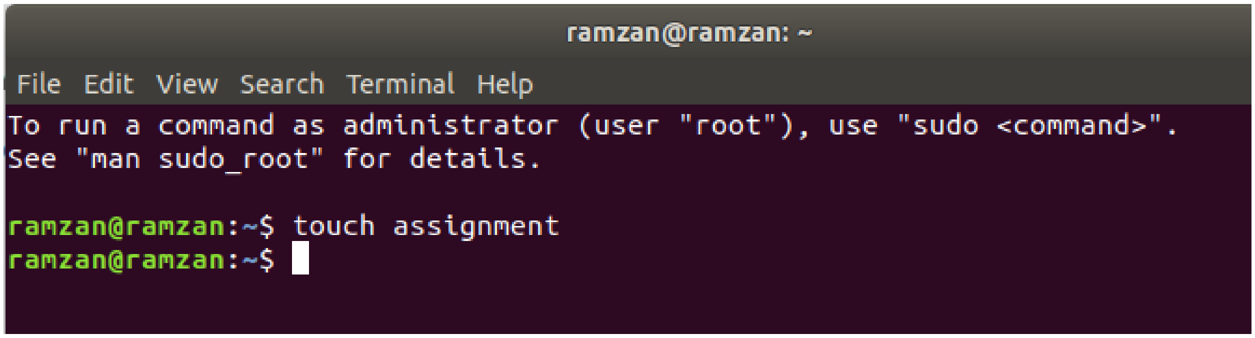
Parameters:-

N.A.

Description:-

Create an empty file.

Screenshot:-



```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ramzan@ramzan:~$ touch assignment
ramzan@ramzan:~$ 
```

27. find:-

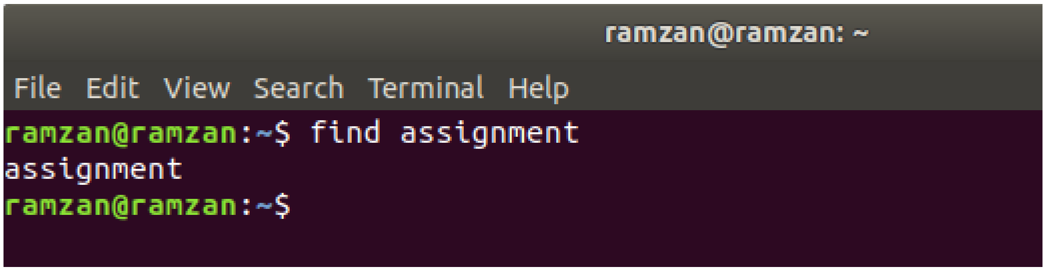
Parameters:-

-name, -iname

Description:-

To find a file by name.

Screenshot:-



```
ramzan@ramzan:~$ find assignment
assignment
ramzan@ramzan:~$
```

28. cp:-

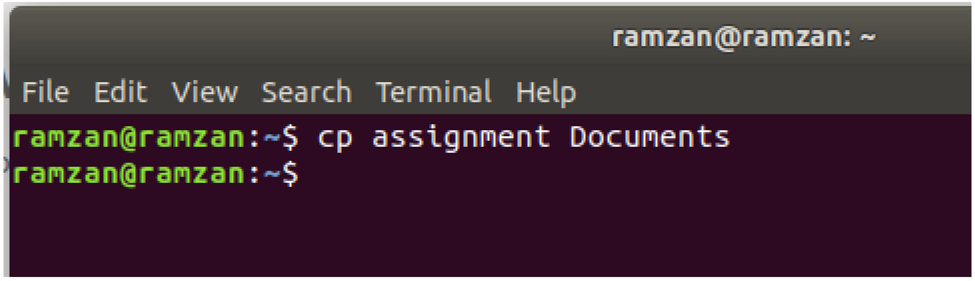
Parameters:-

-a, -f, -i

Description:-

To copy a file.

Screenshot:-



```
ramzan@ramzan:~$ cp assignment Documents
ramzan@ramzan:~$
```

29. dir:-

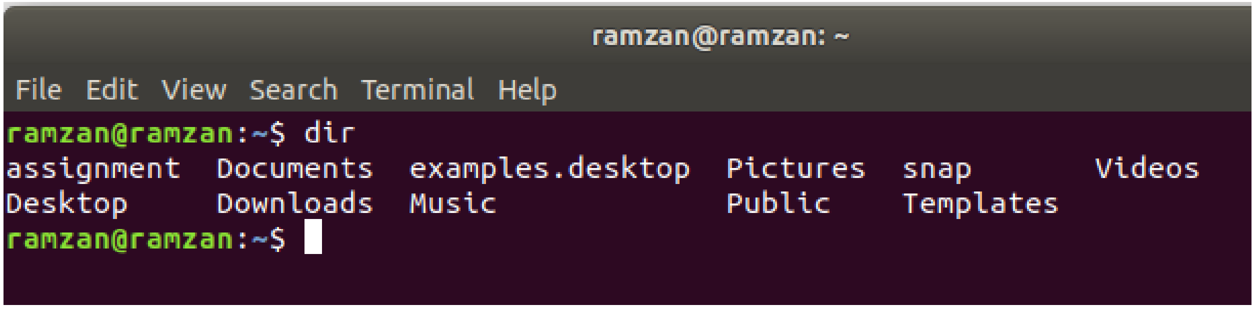
Parameters:-

-a, -all, -l

Description:-

To get a list of all the files and folders in the current directory, use the dir command.

Screenshot:-



```
ramzan@ramzan:~$ dir
assignment  Documents  examples.desktop  Pictures  snap       Videos
Desktop     Downloads  Music             Public    Templates
ramzan@ramzan:~$
```

30. info:-

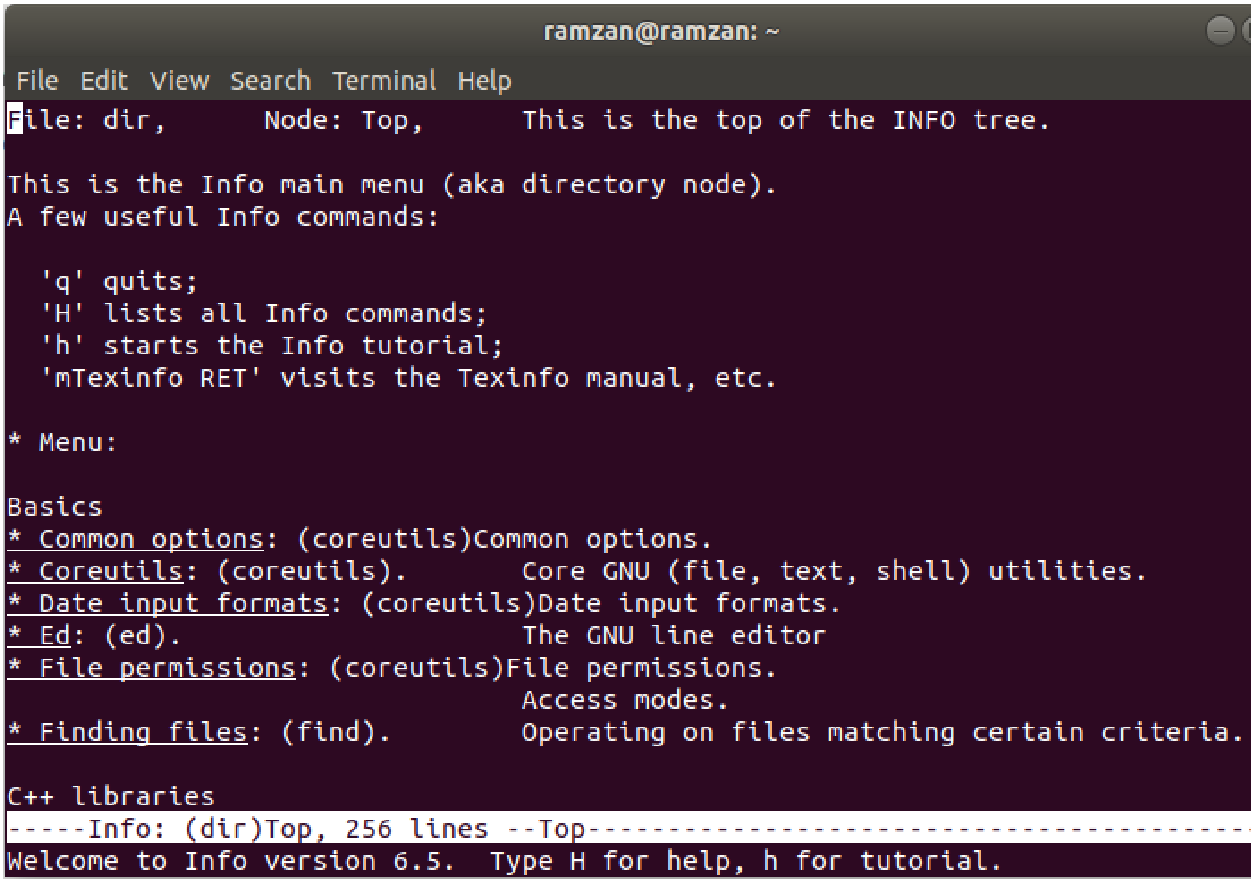
Parameters:-

N.A.

Description:-

Info gives more details about a specific command than by using the man command.

Screenshot:-



```
File: dir,      Node: Top,      This is the top of the INFO tree.

This is the Info main menu (aka directory node).
A few useful Info commands:

  'q' quits;
  'H' lists all Info commands;
  'h' starts the Info tutorial;
  'mTexinfo RET' visits the Texinfo manual, etc.

* Menu:

Basics
* Common options: (coreutils)Common options.
* Coreutils: (coreutils).       Core GNU (file, text, shell) utilities.
* Date input formats: (coreutils)Date input formats.
* Ed: (ed).                     The GNU line editor
* File permissions: (coreutils)File permissions.
                                Access modes.
* Finding files: (find).        Operating on files matching certain criteria.

C++ libraries
-----Info: (dir)Top, 256 lines --Top-------------------------------------
Welcome to Info version 6.5.  Type H for help, h for tutorial.
```

31. nano:-

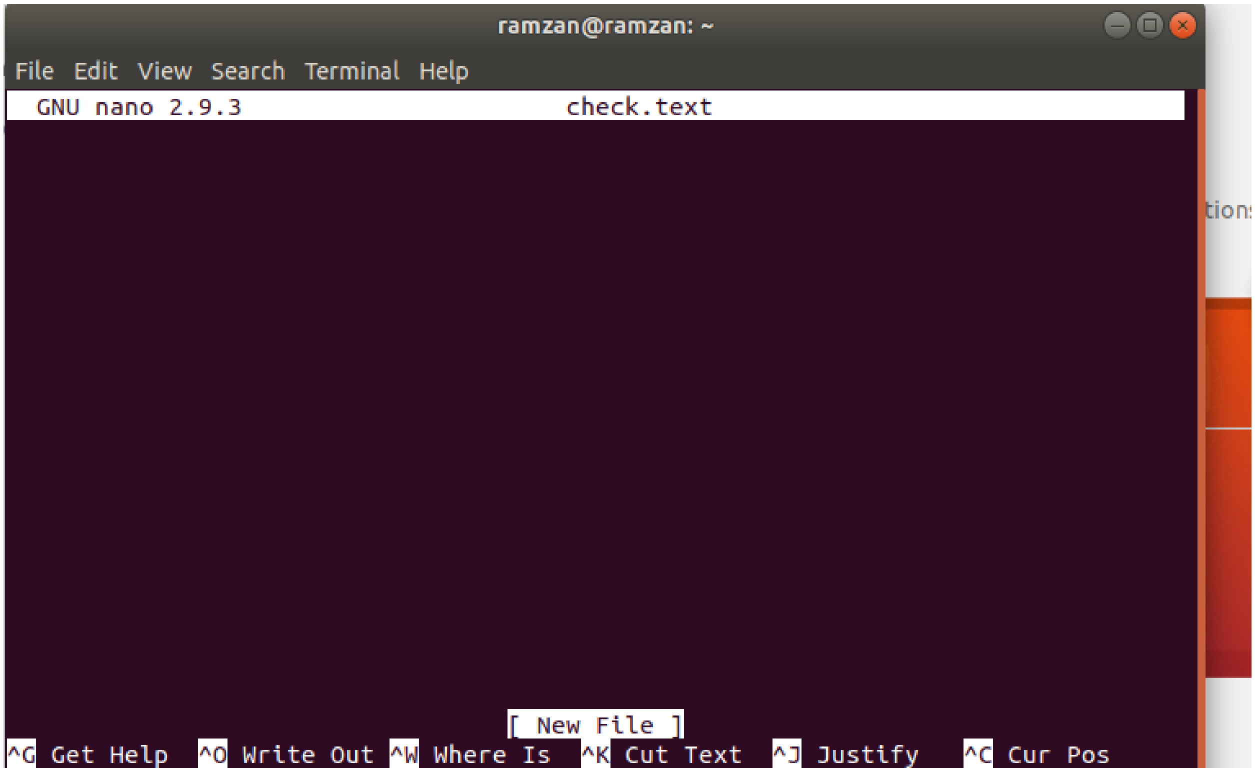
Parameters:-

N.A.

Description:-

nano is already installed text editor in the linux command line. The nano command is a good test editor that denotes keywords with color and can recognize most languages. You can create a new or modify a file using this editor. For example if you need to make a new file named “check.txt”, you can create it by using the command “nano check.txt”. You can save your files after editing by using the sequence Ctrl + X, then Y (or N for no).

Screenshot:-



```
  GNU nano 2.9.3                    check.text

                                [ New File ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
```

32. bzip2:-

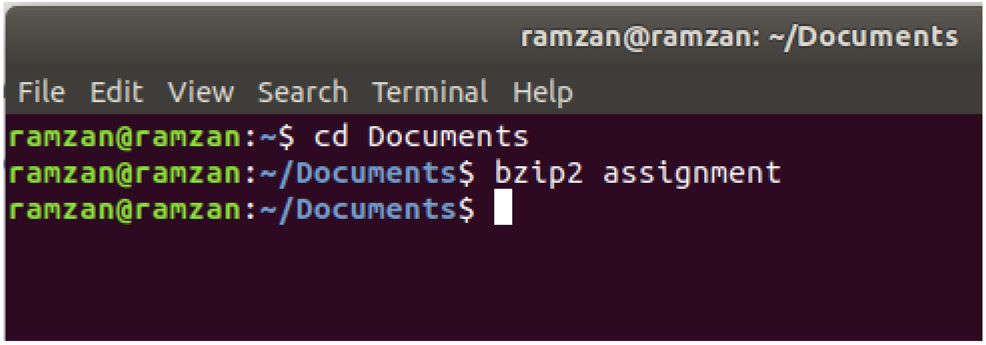
Parameters:-

N.A.

Description:-

A portable, fast, open source program that compresses and decompresses files at a high rate, but that does not archive them.

Screenshot:-


ramzan@ramzan: ~/Documents
File Edit View Search Terminal Help

```
ramzan@ramzan:~$ cd Documents
ramzan@ramzan:~/Documents$ bzip2 assignment
ramzan@ramzan:~/Documents$ 
```

33. service:-

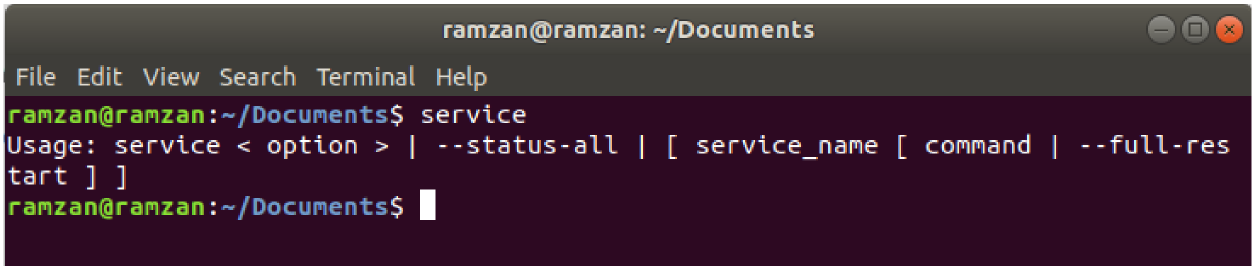
Parameters:-

N.A.

Description:-

This command is the quickest way to start or stop a service, such as networking.

Screenshot:-


ramzan@ramzan: ~/Documents
File Edit View Search Terminal Help

```
ramzan@ramzan:~/Documents$ service
Usage: service < option > | --status-all | [ service_name [ command | --full-res
tart ] ]
ramzan@ramzan:~/Documents$ 
```

34. vi:-

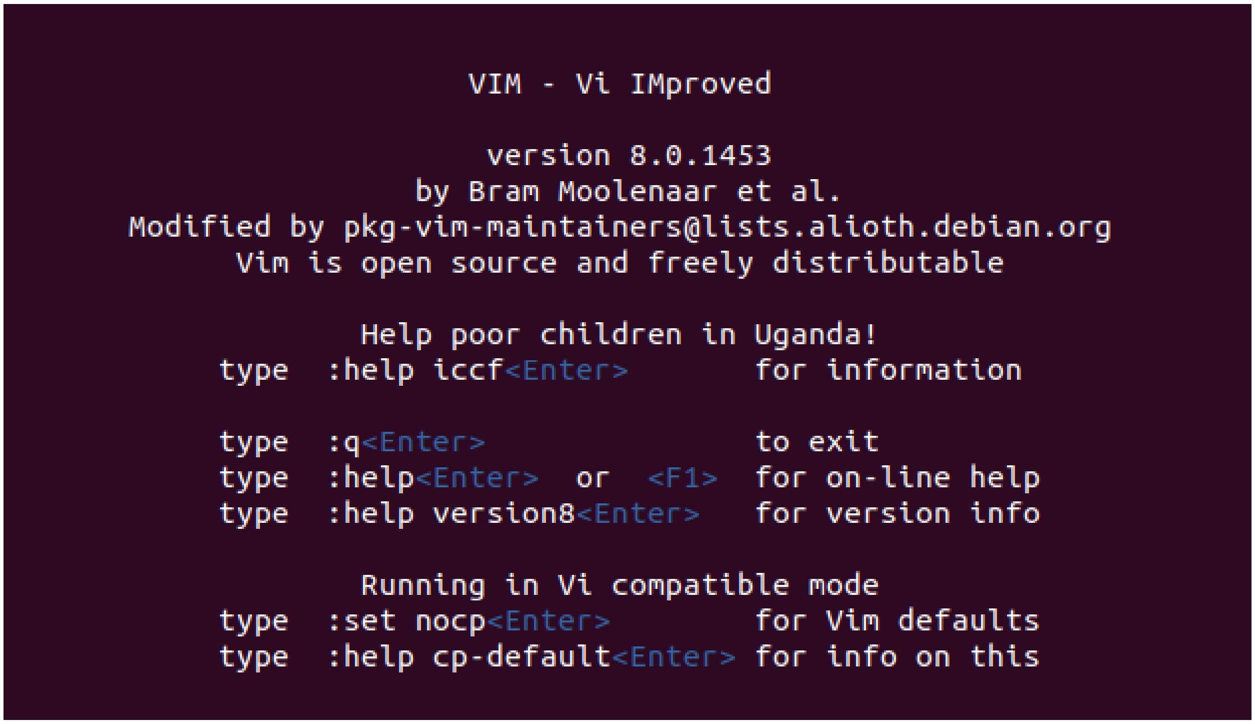
Parameters:-

N.A.

Description:-

The vi environment is a text editor that allows a user to control the system with just the keyboard instead of both mouse selections and keystrokes.

Screenshot:-



```
                         VIM - Vi IMproved

                          version 8.0.1453
                      by Bram Moolenaar et al.
        Modified by pkg-vim-maintainers@lists.alioth.debian.org
              Vim is open source and freely distributable

                     Help poor children in Uganda!
             type  :help iccf<Enter>       for information

             type  :q<Enter>               to exit
             type  :help<Enter>  or  <F1>  for on-line help
             type  :help version8<Enter>   for version info

                     Running in Vi compatible mode
             type  :set nocp<Enter>        for Vim defaults
             type  :help cp-default<Enter> for info on this
```

35. vmstat:-

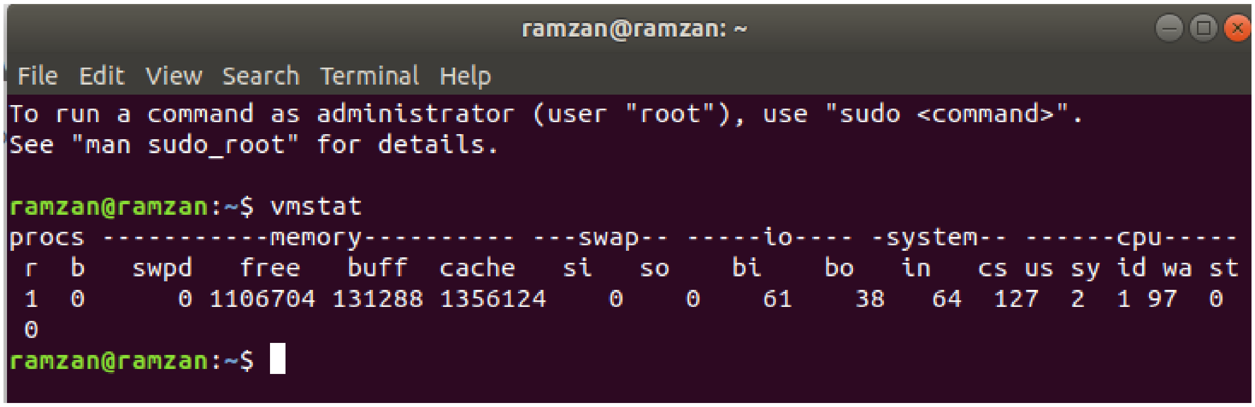
Parameters:-

N.A.

Description:-

The vmstat command snapshots everything in a system and reports information on such items as processes, memory, paging and CPU activity. This is a good method for admins to use to determine where issues/slowdown may occur in a system.

Screenshot:-



```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ramzan@ramzan:~$ vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 1106704 131288 1356124    0    0    61    38   64  127  2  1 97  0
 0
ramzan@ramzan:~$ 
```

36. *ssh:-*

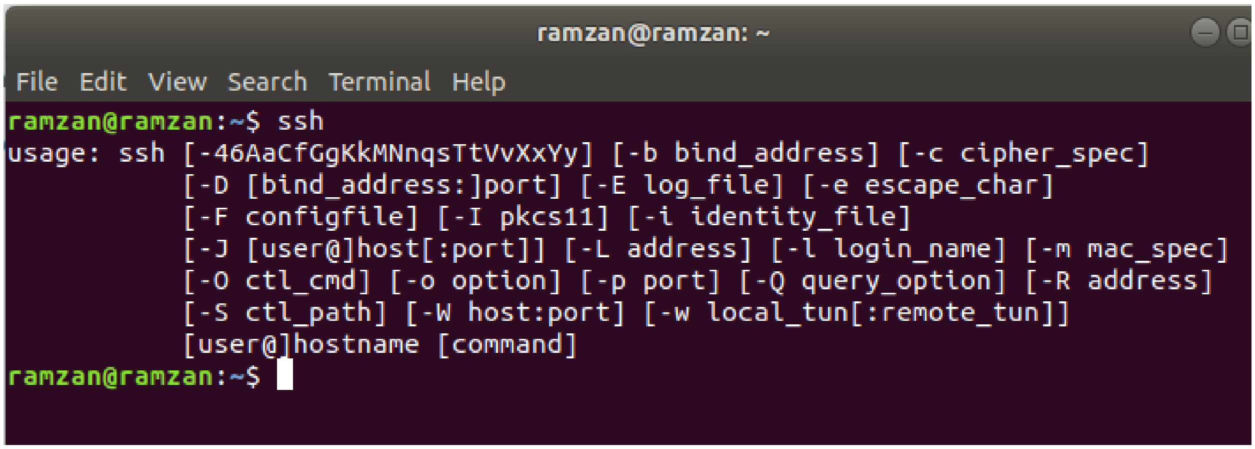
Parameters:-

N.A.

Description:-

SSH is a command interface for secure remote computer access and is used by network admins to remotely control servers.

Screenshot:-



```
ramzan@ramzan:~$ ssh
usage: ssh [-46AaCfGgKkMNnqsTtVvXxYy] [-b bind_address] [-c cipher_spec]
           [-D [bind_address:]port] [-E log_file] [-e escape_char]
           [-F configfile] [-I pkcs11] [-i identity_file]
           [-J [user@]host[:port]] [-L address] [-l login_name] [-m mac_spec]
           [-O ctl_cmd] [-o option] [-p port] [-Q query_option] [-R address]
           [-S ctl_path] [-W host:port] [-w local_tun[:remote_tun]]
           [user@]hostname [command]
ramzan@ramzan:~$ 
```

37. exit:-

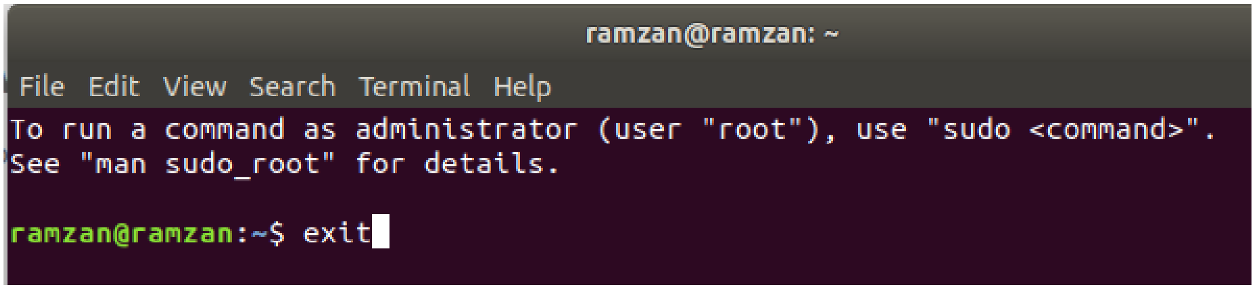
Parameters:-

N.A.

Description:-

exit command is used to exit a shell like so.

Screenshot:-



```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ramzan@ramzan:~$ exit
```

38. expr:-

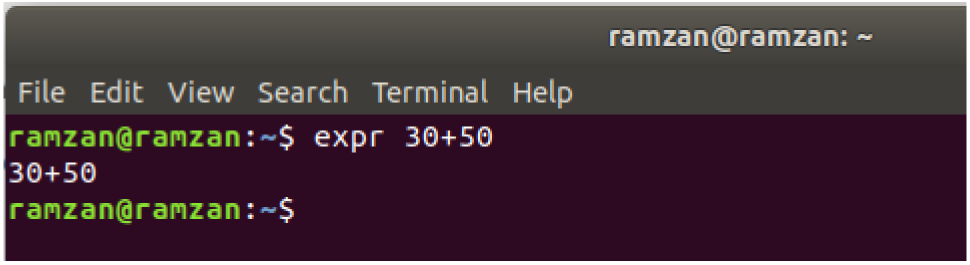
Parameters:-

N.A.

Description:-

expr command is used calculate an expression.

Screenshot:-



```
ramzan@ramzan:~$ expr 30+50
30+50
ramzan@ramzan:~$
```

39. kmod:-

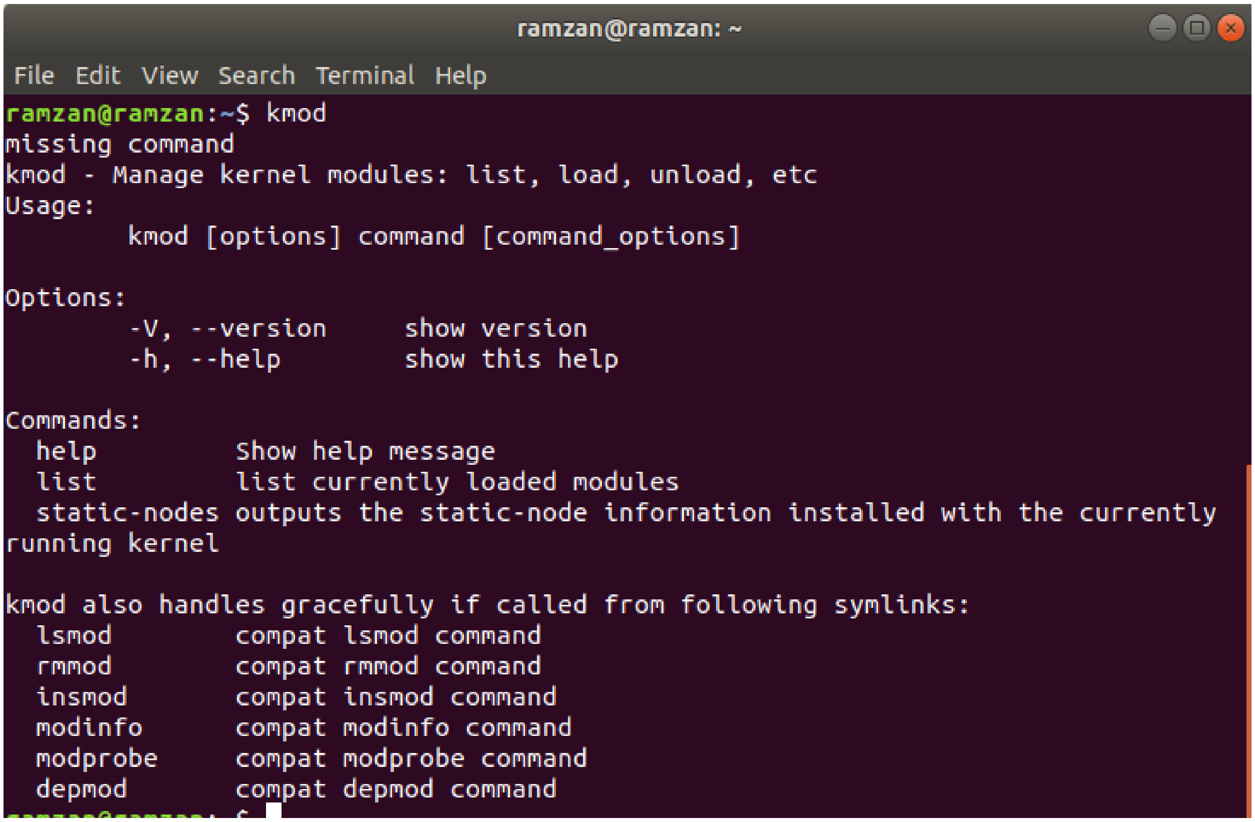
Parameters:-

N.A.

Description:-

kmod command is used to manage linux kernel modules and list all currently loaded modules.

Screenshot:-



```
ramzan@ramzan:~$ kmod
missing command
kmod - Manage kernel modules: list, load, unload, etc
Usage:
        kmod [options] command [command_options]

Options:
        -V, --version     show version
        -h, --help        show this help

Commands:
  help         Show help message
  list         list currently loaded modules
  static-nodes outputs the static-node information installed with the currently
running kernel

kmod also handles gracefully if called from following symlinks:
  lsmod        compat lsmod command
  rmmod        compat rmmod command
  insmod       compat insmod command
  modinfo      compat modinfo command
  modprobe     compat modprobe command
  depmod       compat depmod command
ramzan@ramzan:~$
```

40. lscpu:-

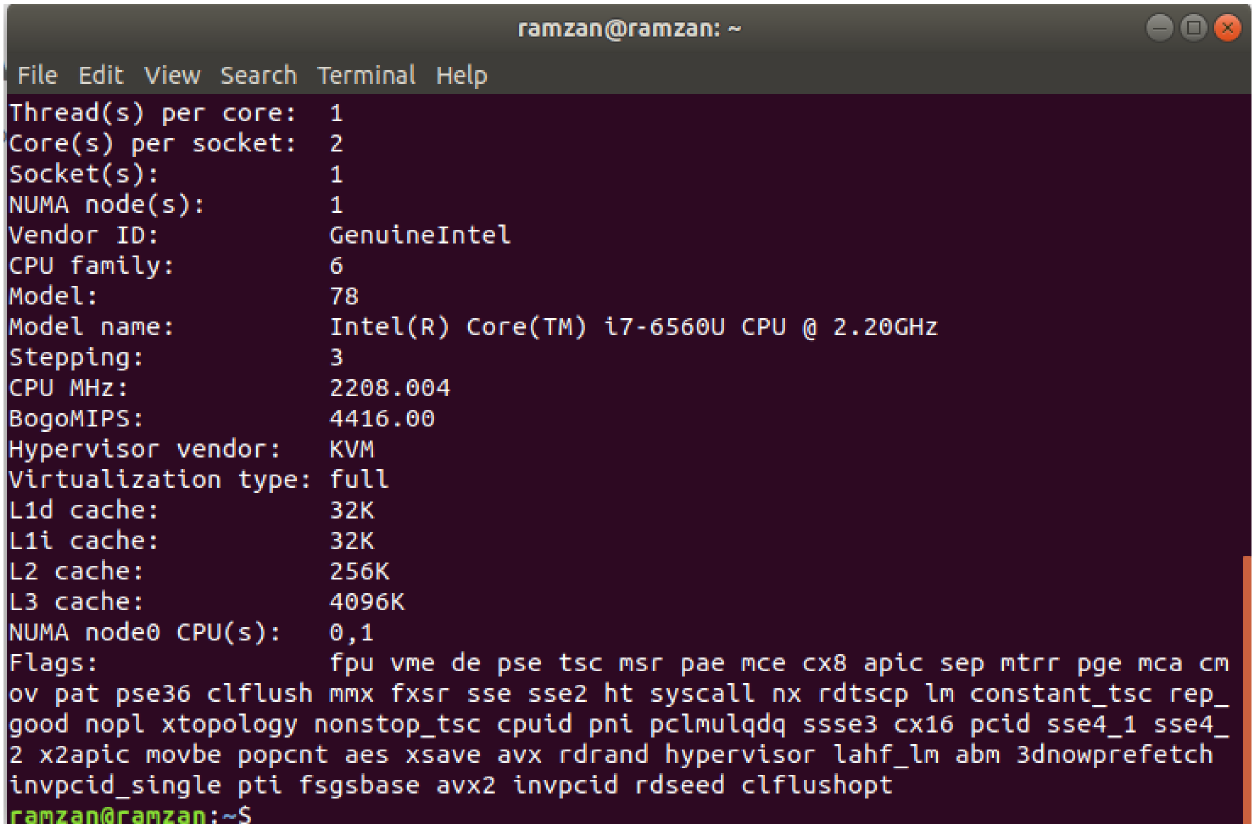
Parameters:-

N.A.

Description:-

lscpu command displays system's CPU architecture information (such as number of CPUs, threads, cores, sockets, and more).

Screenshot:-



```
Thread(s) per core:  1
Core(s) per socket:  2
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               78
Model name:          Intel(R) Core(TM) i7-6560U CPU @ 2.20GHz
Stepping:            3
CPU MHz:             2208.004
BogoMIPS:            4416.00
Hypervisor vendor:   KVM
Virtualization type: full
L1d cache:           32K
L1i cache:           32K
L2 cache:            256K
L3 cache:            4096K
NUMA node0 CPU(s):   0,1
Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cm
ov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_tsc rep_
good nopl xtopology nonstop_tsc cpuid pni pclmulqdq ssse3 cx16 pcid sse4_1 sse4_
2 x2apic movbe popcnt aes xsave avx rdrand hypervisor lahf_lm abm 3dnowprefetch
invpcid_single pti fsgsbase avx2 invpcid rdseed clflushopt
ramzan@ramzan:~$
```

41. nproc:-

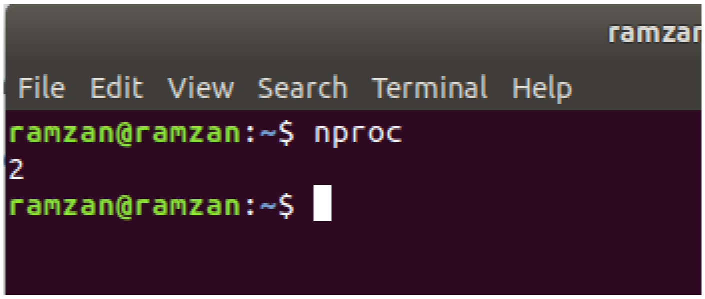
Parameters:-

N.A.

Description:-

nproc command shows the number of processing units present to the current process. It's output may be less than the number of online processors on a system

Screenshot:-



```
ramzan@ramzan:~$ nproc
2
ramzan@ramzan:~$ 
```

42. stat:-

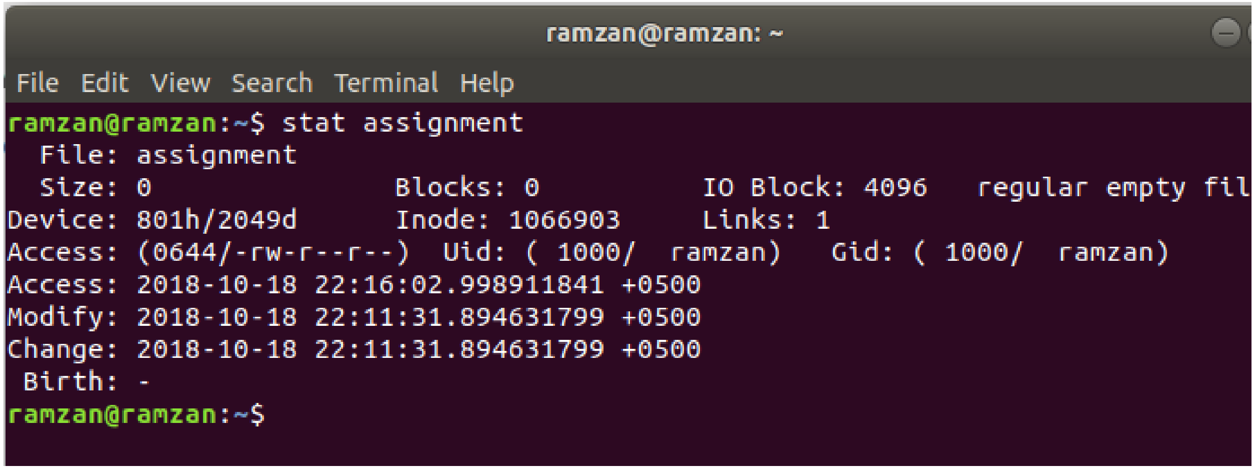
Parameters:-

N.A.

Description:-

stat command is used to show the statistics of a file.

Screenshot:-



```
ramzan@ramzan:~$ stat assignment
  File: assignment
  Size: 0               Blocks: 0          IO Block: 4096   regular empty fil
Device: 801h/2049d      Inode: 1066903     Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/  ramzan)   Gid: ( 1000/  ramzan)
Access: 2018-10-18 22:16:02.998911841 +0500
Modify: 2018-10-18 22:11:31.894631799 +0500
Change: 2018-10-18 22:11:31.894631799 +0500
 Birth: -
ramzan@ramzan:~$
```

43. w:-

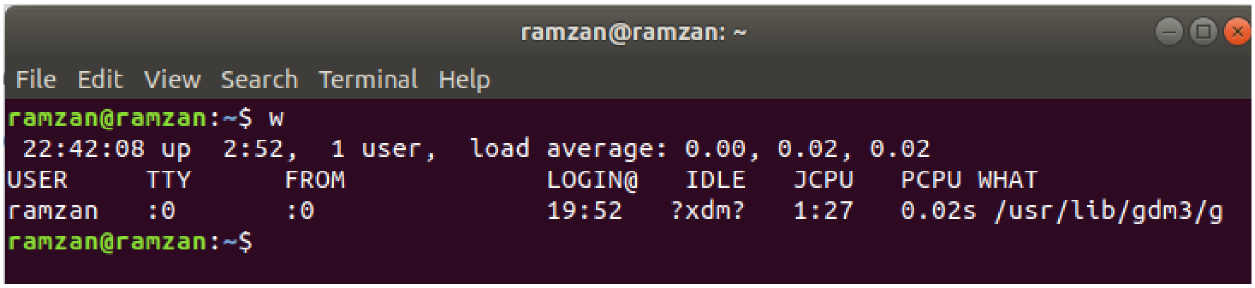
Parameters:-

N.A.

Description:-

w command displays system uptime, load averages and information about the users currently on the machine, and what they are doing (their processes).

Screenshot:-



```
ramzan@ramzan:~$ w
 22:42:08 up  2:52,  1 user,  load average: 0.00, 0.02, 0.02
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
ramzan   :0       :0               19:52   ?xdm?   1:27   0.02s /usr/lib/gdm3/g
ramzan@ramzan:~$
```

44. wc:-

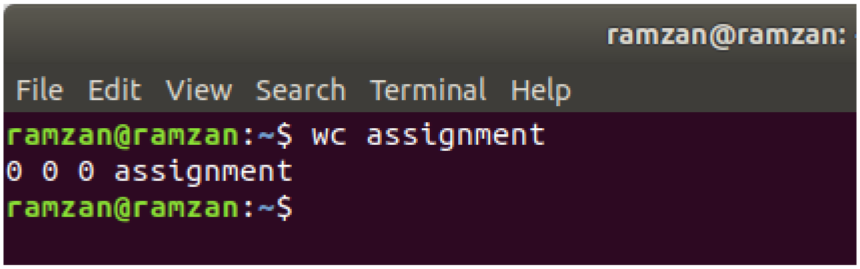
Parameters:-

N.A.

Description:-

wc command is used to display newline, word, and byte counts for each file specified, and a total for many files.

Screenshot:-



```
ramzan@ramzan:~$ wc assignment
0 0 0 assignment
ramzan@ramzan:~$
```

45. yes:-

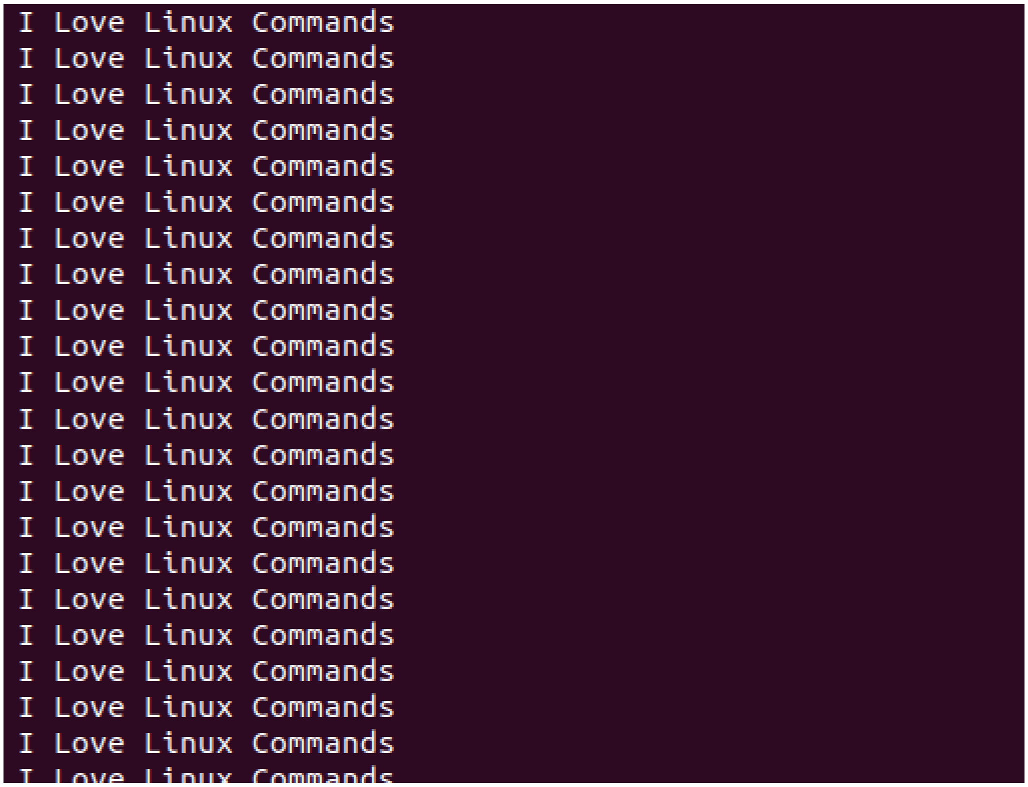
Parameters:-

N.A.

Description:-

yes command is used to display a string repeatedly until when terminated or killed using Ctrl + C.

Screenshot:-



```
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
I Love Linux Commands
```

46. du:-

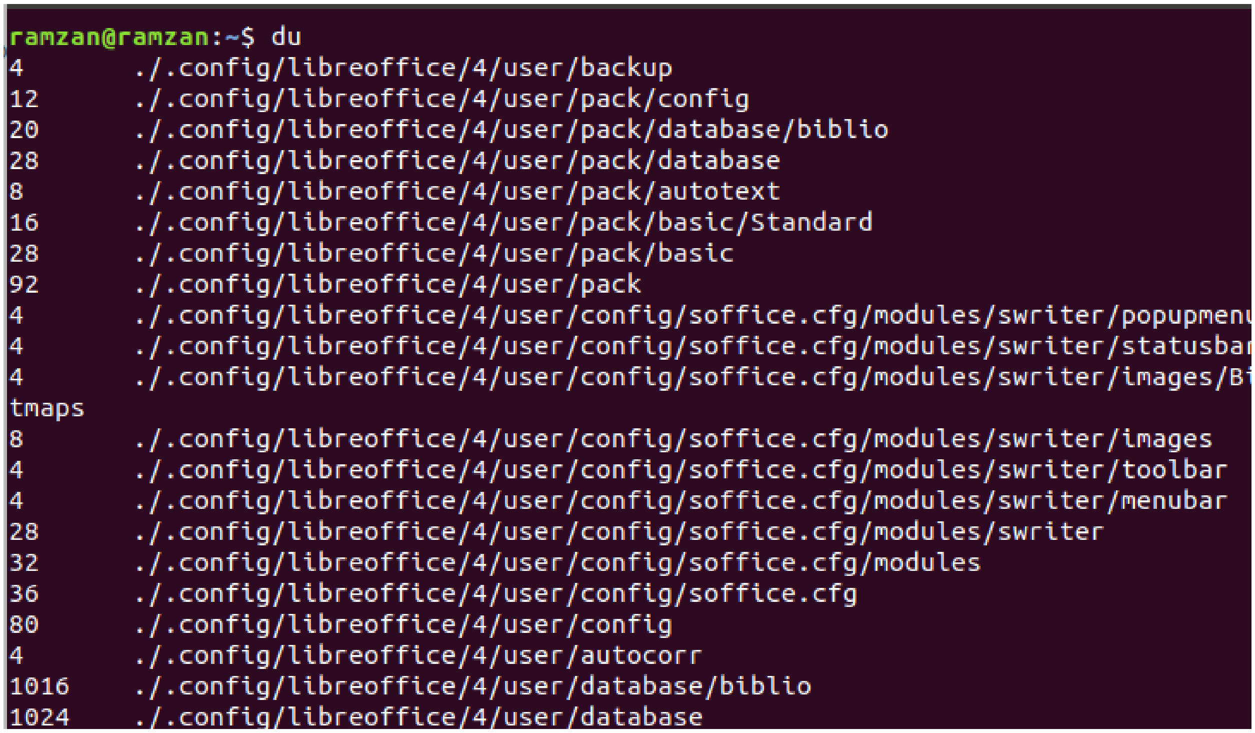
Parameters:-

-h, -s

Description:-

To retrieve more detailed information about which files use the disk space in a directory, you can use the du command

Screenshot:-



```
ramzan@ramzan:~$ du
4       ./.config/libreoffice/4/user/backup
12      ./.config/libreoffice/4/user/pack/config
20      ./.config/libreoffice/4/user/pack/database/biblio
28      ./.config/libreoffice/4/user/pack/database
8       ./.config/libreoffice/4/user/pack/autotext
16      ./.config/libreoffice/4/user/pack/basic/Standard
28      ./.config/libreoffice/4/user/pack/basic
92      ./.config/libreoffice/4/user/pack
4       ./.config/libreoffice/4/user/config/soffice.cfg/modules/swriter/popupmenu
4       ./.config/libreoffice/4/user/config/soffice.cfg/modules/swriter/statusbar
4       ./.config/libreoffice/4/user/config/soffice.cfg/modules/swriter/images/Bi
tmaps
8       ./.config/libreoffice/4/user/config/soffice.cfg/modules/swriter/images
4       ./.config/libreoffice/4/user/config/soffice.cfg/modules/swriter/toolbar
4       ./.config/libreoffice/4/user/config/soffice.cfg/modules/swriter/menubar
28      ./.config/libreoffice/4/user/config/soffice.cfg/modules/swriter
32      ./.config/libreoffice/4/user/config/soffice.cfg/modules
36      ./.config/libreoffice/4/user/config/soffice.cfg
80      ./.config/libreoffice/4/user/config
4       ./.config/libreoffice/4/user/autocorr
1016    ./.config/libreoffice/4/user/database/biblio
1024    ./.config/libreoffice/4/user/database
```

47. pmap:-

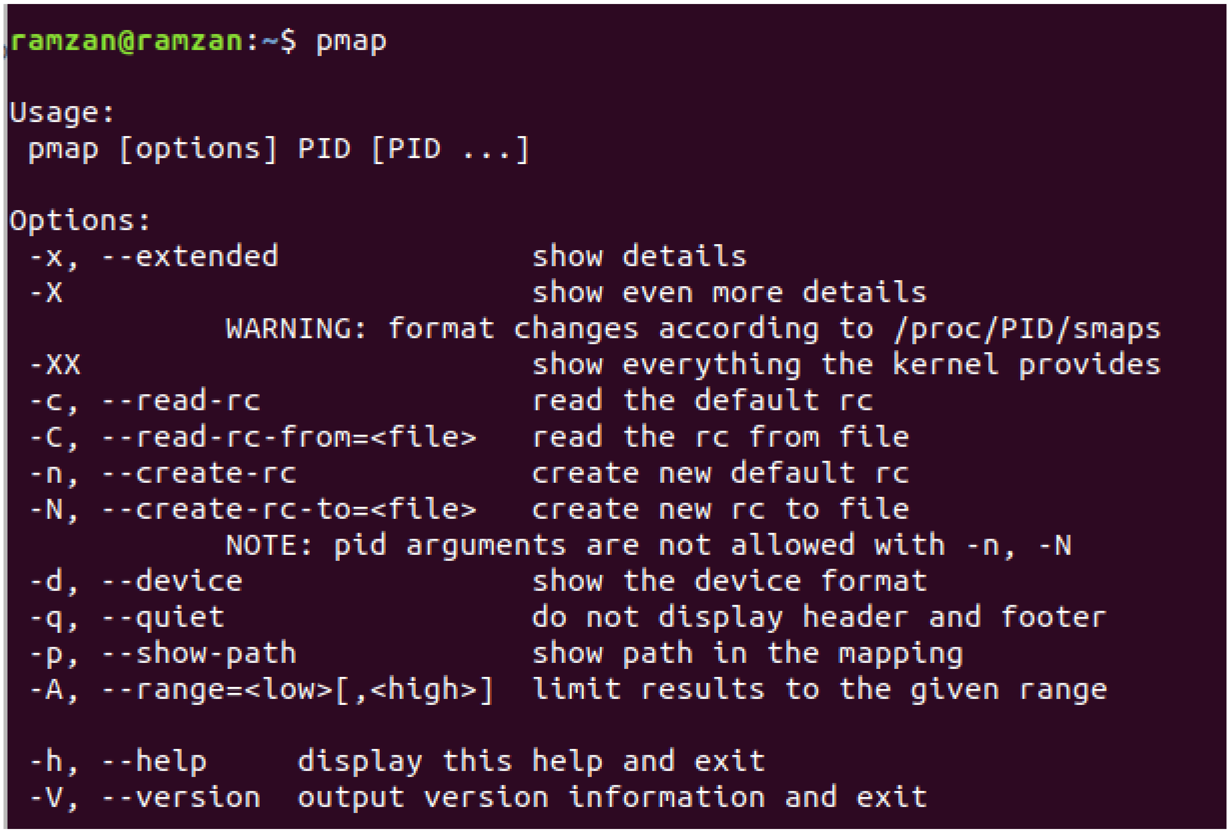
Parameters:-

N.A.

Description:-

Display Memory map of process.

Screenshot:-



```
ramzan@ramzan:~$ pmap

Usage:
 pmap [options] PID [PID ...]

Options:
 -x, --extended              show details
 -X                          show even more details
            WARNING: format changes according to /proc/PID/smaps
 -XX                         show everything the kernel provides
 -c, --read-rc               read the default rc
 -C, --read-rc-from=<file>   read the rc from file
 -n, --create-rc             create new default rc
 -N, --create-rc-to=<file>   create new rc to file
            NOTE: pid arguments are not allowed with -n, -N
 -d, --device                show the device format
 -q, --quiet                 do not display header and footer
 -p, --show-path             show path in the mapping
 -A, --range=<low>[,<high>]  limit results to the given range

 -h, --help     display this help and exit
 -V, --version  output version information and exit
```

48. shutdown:-

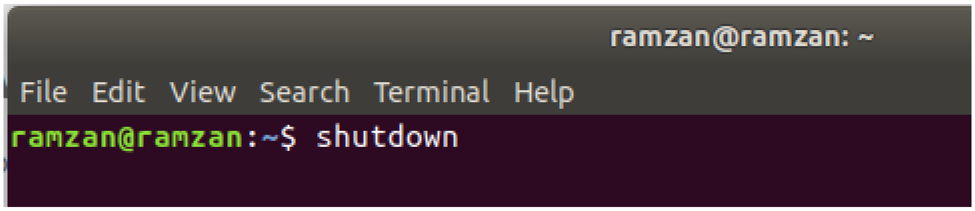
Parameters:-

-h, -r

Description:-

The shutdown command turns off the computer and can be combined with variables such as -h for halt after shutdown or -r for reboot after shutdown.

Screenshot:-



```
ramzan@ramzan:~$ shutdown
```

49. reboot:-

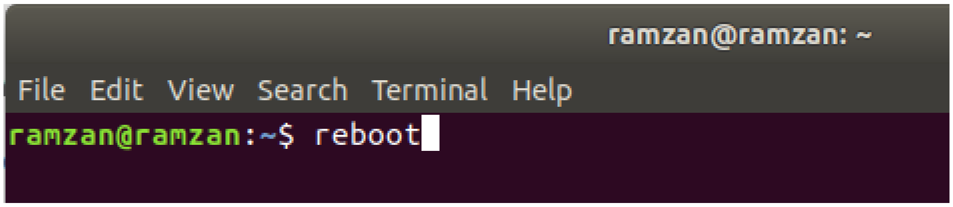
Parameters:-

N.A.

Description:-

Restart the system.

Screenshot:-



```
ramzan@ramzan:~$ reboot
```

50. pstree:-

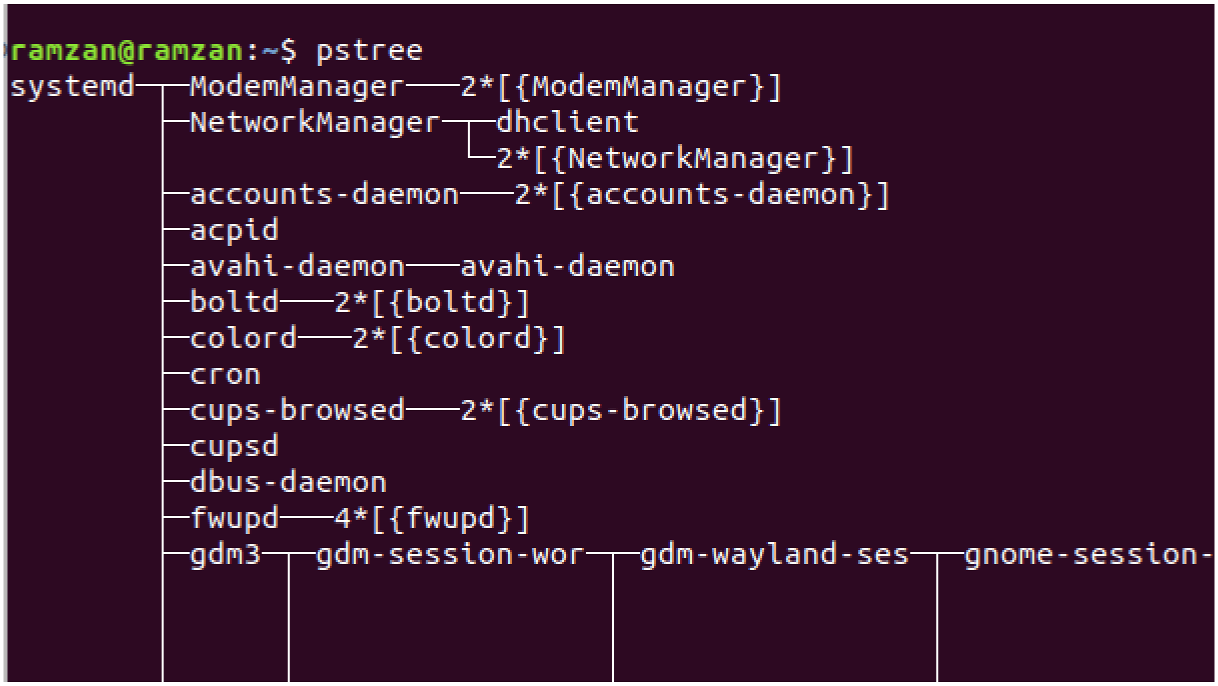
Parameters:-

N.A.

Description:-

This commands shows all the processes running currently along with associated child process, in a tree like format similar to ‘tree‘ command output.

Screenshot:-



```
ramzan@ramzan:~$ pstree
systemd─┬─ModemManager───2*[{ModemManager}]
        ├─NetworkManager─┬─dhclient
        │                └─2*[{NetworkManager}]
        ├─accounts-daemon───2*[{accounts-daemon}]
        ├─acpid
        ├─avahi-daemon───avahi-daemon
        ├─boltd───2*[{boltd}]
        ├─colord───2*[{colord}]
        ├─cron
        ├─cups-browsed───2*[{cups-browsed}]
        ├─cupsd
        ├─dbus-daemon
        ├─fwupd───4*[{fwupd}]
        ├─gdm3─┬─gdm-session-wor─┬─gdm-wayland-ses─┬─gnome-session-
        │      │                 │                 │
```

LAB 02

System Calls

3. IO SYSTEM CALLS

AIM:

To write a ‘c’ program for I/O system calls.

ALGORITHM:

1. Start the programs
2. open a file for O_RDWR for R/W,O_CREATE for creating a file , O_TRUNC for truncate a file
3. Using getchar(), read the character and stored in the string[] array
4. The string [] array is write into a file close it.
5. Then the first is opened for read only mode and read the characters and displayed It and close the file
6. Stop the program

Write a program to take id name and CGPA and write in a file using IO system calls. Your program also read id name and CGPA from file and print on console.

```
#include <fcntl.h>
#include <stdio.h>
#include <zconf.h>
main( )
{
   char id[20], name[50],CGPA[5];
   char Rid[20], Rname[50],RCGPA[4];
   int fp = open("file",O_RDWR|O_CREAT);
   if(fp != -1)
   {
      printf("Enter ID : ");
      fgets(id, sizeof(id),stdin);
      printf("Enter Name : ");
      fgets(name, sizeof(name),stdin);
      printf("Enter CGPA : ");
      fgets(CGPA, sizeof(CGPA),stdin);
      printf("All records write using write() System Calls\n");
      write(fp,id, sizeof(id));
```

```
        write(fp,name, sizeof(name));
        write(fp,CGPA, sizeof(CGPA));
        lseek(fp,0,0);
        printf("All records Read using Read() System Calls\n");
        read(fp,Rid, sizeof(id));
        read(fp,Rname, sizeof(name));
        read(fp,RCGPA, sizeof(CGPA));
        printf("ID : %s\nName : %s\nCGPA : %s\n",Rid,Rname,RCGPA);
        close(fp);
    }
    else
    {
        printf("Ops Error");
    }
    return 0;
}
```

output

```
Enter ID : Your ID
Enter Name : Your Name
Enter CGPA : 3.72
All records write using write() System Calls
All records Read using Read() System Calls
ID : Your ID
Name :  Your Name
CGPA : 3.72
Process finished with exit code 0
```

4. PROCESS SYSTEM CALLS

AIM:

To write c program to implement the Process system calls.

ALGORITHM:

1. Start the program.
2. Declare the pid and get the pid by using the getpid() method.
3. Create a child process by calling the fork() system call
4. Check if(pid==0) then print the child process id and then print the parent process value. Otherwise print
5. Stop the program

Write a program to create a child process using fork system call. And print id of child and parent process.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include <sys/wait.h>
void main(int argc,char *arg[])
{
   int pid;
   pid=fork();
   if(pid<0)
   {
      printf("fork failed");
      exit(1);
   }
   else if(pid==0)
   {
      printf("Child Process id is -%d\n",getpid());
      exit(0);
   }
   else
   {
        printf("Parent Process id is -%d\n",getpid());
        exit(0);
    }
}
```

output

```
Parent Process id is -3496
Child Process id is -3497

Process finished with exit code 0
```

Write a program to create a child process using fork system call. Your program execute child process before parent process. Hint: use wait system call

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include <sys/wait.h>
void main(int argc,char *arg[])
{
    int pid;
    pid=fork();
    if(pid<0)
    {
        printf("fork failed");
        exit(1);
    }
    else if(pid==0)
    {
        printf("Child Process id is -%d\n",getpid());
        exit(0);
    }
    else
    {
        wait(NULL);
        printf("Parent Process id is -%d\n",getpid());
        exit(0);
    }
}
```

output

```
Child Process id is -3609
Parent Process id is -3607

Process finished with exit code 0
```

Write a program to create a child process using fork system call. Your program execute child process before parent process and print system name using ececlp system call. Also print parent and child process ID.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include <sys/wait.h>
void main(int argc,char *arg[])
{
    int pid;
    pid=fork();
    if(pid<0)
    {
        printf("fork failed");
        exit(1);
    }
    else if(pid==0)
    {
        printf("Child Process id is -%d\n",getpid());
        execlp("whoami","ls",NULL);
        // my system name is usman
```

```
                //that is why print usman.
        // here user host/system name print
        exit(0);
    }
    else
    {
        wait(NULL);
        printf("Parent Process id is -%d\n",getpid());
        exit(0);
    }
}
```

output

```
Child Process id is -3661
usman
Parent Process id is -3659
Process finished with exit code 0
```

Write a program to print Id name and CGP passed by another program using execl() System call.

```
//save program as a.c
#include<stdio.h>
int main(int argc,char const *argv[])
{
    for(int i=0; i<argc; i++)
    {
        printf("%s\t",argv[i]);
    }
    printf("\n");
}
```

Run program as gcc a.c -o a

```
//save program as b.c
#include<stdio.h>
#include<unistd.h>
int main(int argc,char const *argv[])
{
   char id[20],name[100],cgpa[4];
   puts("Enter Id : ");
   fgets(id,20,stdin);
   puts("Enter Name : ");
   fgets(name,100,stdin);
   puts("Enter Cgpa : ");
   fgets(cgpa,4,stdin);
   execl("/home/usman/Desktop/OS Lab solution/mid
solution/a","a",id,name,cgpa,NULL);
}
```

Run as gcc b.c -o b
./b

output

```
Enter Id :
F2016065065
Enter Name :
Your Name
Enter Cgpa :
3.72
a       F2016065065
        Your Name
        3.72

Process finished with exit code 0
```

Write a program to make child orphan and print parent and child process ID.

```
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main()
{
    pid_t pid = fork();
    if (pid > 0)
    {
        printf("Parent process Section\n");
    }
    else
    {
        sleep(10);
        printf("hy i am child process\n");
        exit(0);
    }

    return 0;
}
```

output

```
usman@usman-HP-Notebook:~/Desktop/OS Lab solution/New Solution$ gcc program1.c -o program1
usman@usman-HP-Notebook:~/Desktop/OS Lab solution/New Solution$ ./program1
Parent process Section
usman@usman-HP-Notebook:~/Desktop/OS Lab solution/New Solution$ hy i am child process
```

Write a program to execute zombie (a process called zombie if their child dies) process. Use sleep wait and fork system calls to perform that task.

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

int main()
{
    // Fork returns process id
    // in parent process
    pid_t pid = fork();

    // Parent process
    if (pid > 0)
    {
        wait(NULL);
        sleep(30);
        printf("Parent process Section\n");
    }
    else  if(pid ==0)
    {
        printf("hy i am child process\n");
        exit(0);
    }
    else
    {
        printf("Fork call failed");
    }

    return 0;
}
```

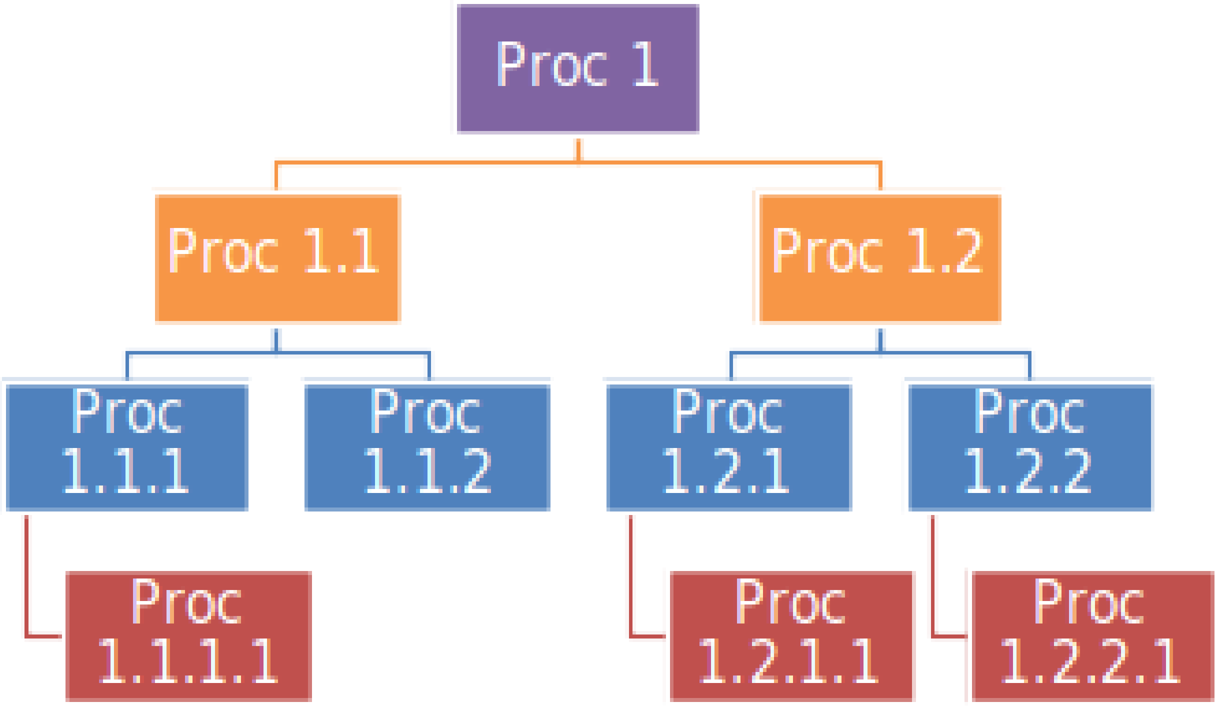
output

```
hy i am child process
Parent process Section

Process finished with exit code 0
```

Question: **The purpose of this assignment is to use linux system calls like fork (), wait ( ) ,vfork( ),clone ( )and exit ( ) etc. Use fork system call to generate the following tree. After creating the above tree solve following the questions:**

- **What is the Process ID of *Proc 1.1.1.1*?**
- **Kill *Proc 1.1?***
- **What is the state of *Proc 1.1.1* and *Proc 1.1.2* and what happened to *Proc 1.1.1.1*?**
- **Who is the parent of Proc 1.1.1 and Proc 1.1.2?**
- **What is the Parent ID of *Proc 1.2.1*?**
- **Create the same tree using vfork and clone command and analyze the difference between trees.**
- **Block the process *Proc 1.2* from termination until all of its child process terminated**.

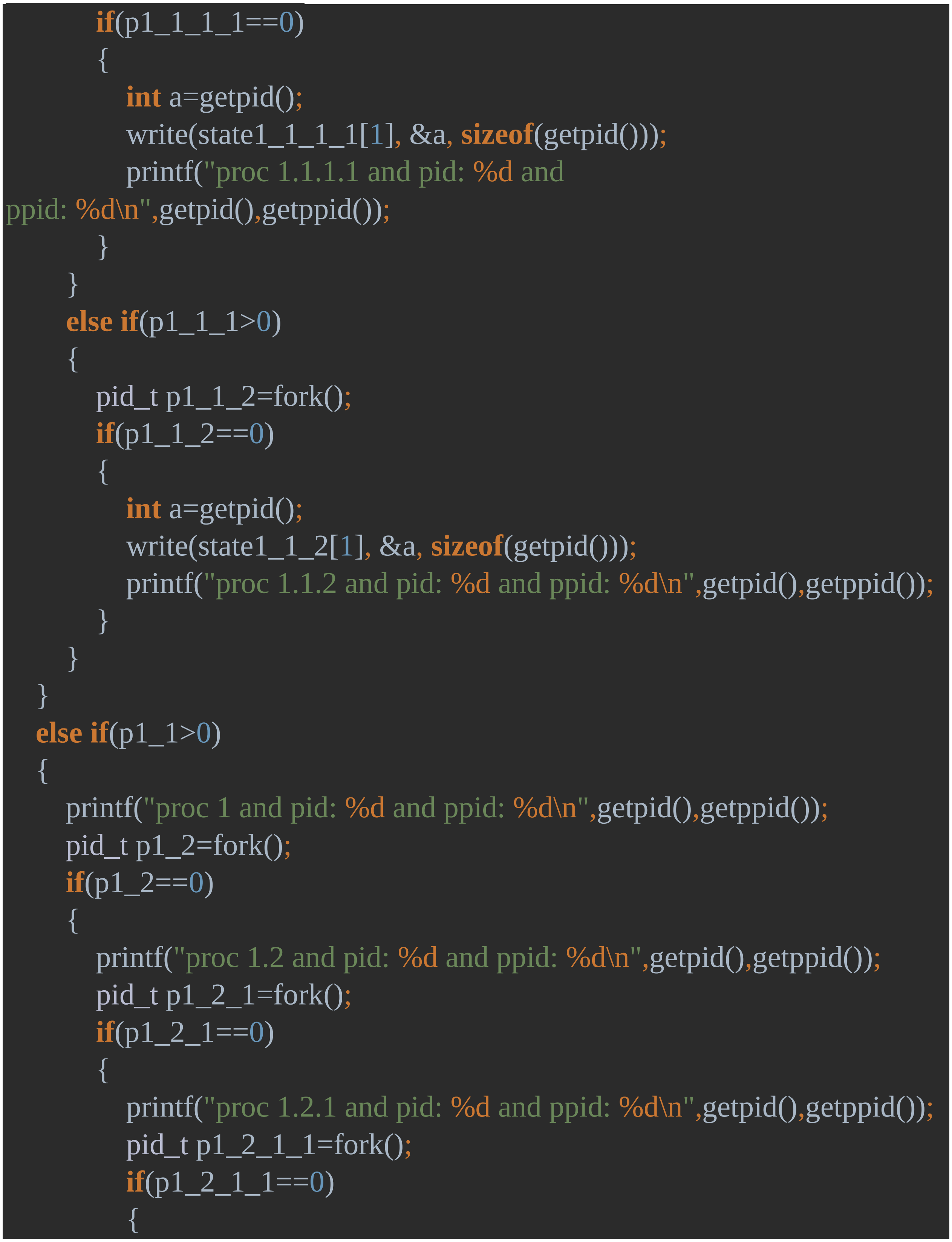


```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<signal.h>
#include<string.h>


int savearr[2];
int state1_1_1[2];
int state1_1_2[2];
int state1_1_1_1[2];
int t=0;
int main()
{
    pipe(savearr);
    pipe(state1_1_1);
    pipe(state1_1_2);
    pipe(state1_1_1_1);

    pid_t p1_1=fork();
    if(p1_1==0)
    {

        printf("proc 1.1 and pid: %d and ppid: %d\n",getpid(),getppid());
        int a=getpid();
        write(savearr[1], &a, sizeof(getpid()));
        pid_t p1_1_1=fork();
        if(p1_1_1==0)
        {
            int a=getpid();
            write(state1_1_1[1], &a, sizeof(getpid()));
            printf("proc 1.1.1 and pid: %d and ppid: %d\n",getpid(),getppid());
            pid_t p1_1_1_1=fork();
```



```
            if(p1_1_1_1==0)
            {
                int a=getpid();
                write(state1_1_1_1[1], &a, sizeof(getpid()));
                printf("proc 1.1.1.1 and pid: %d and
ppid: %d\n",getpid(),getppid());
            }
        }
        else if(p1_1_1>0)
        {
            pid_t p1_1_2=fork();
            if(p1_1_2==0)
            {
                int a=getpid();
                write(state1_1_2[1], &a, sizeof(getpid()));
                printf("proc 1.1.2 and pid: %d and ppid: %d\n",getpid(),getppid());
            }
        }
    }
    else if(p1_1>0)
    {
        printf("proc 1 and pid: %d and ppid: %d\n",getpid(),getppid());
        pid_t p1_2=fork();
        if(p1_2==0)
        {
            printf("proc 1.2 and pid: %d and ppid: %d\n",getpid(),getppid());
            pid_t p1_2_1=fork();
            if(p1_2_1==0)
            {
                printf("proc 1.2.1 and pid: %d and ppid: %d\n",getpid(),getppid());
                pid_t p1_2_1_1=fork();
                if(p1_2_1_1==0)
                {
                    printf("proc 1.2.1.1 and pid: %d and 
ppid: %d\n",getpid(),getppid());
                }
            }
            if(p1_2_1>0)
            {
                pid_t p1_2_2=fork();
                if(p1_2_2==0)
                {
                    printf("proc 1.2.2 and pid: %d and 
ppid: %d\n",getpid(),getppid());
                    pid_t p1_2_2_1=fork();

                    if(p1_2_2_1==0)
                    {
                        printf("proc 1.2.2.1 and pid: %d and 
ppid: %d\n",getpid(),getppid());
                    }
                }
            }
        }
        read(state1_1_1[0],&t,sizeof(int));
        char command[20];
        sprintf(command,"ps ");
        system("command");
        printf("\nfinal process 1.1.1 id: %d\n",t);
        read(state1_1_2[0],&t,sizeof(int));
        printf("\nfinal process 1.1.2 id: %d\n",t);
        read(state1_1_1_1[0],&t,sizeof(int));
        printf("\nfinal process 1.1.1.1 id: %d\n",t);


        read(savearr[0],&t,sizeof(int));
        kill(t,1);          //killllllllllllllllllllllllllllllllllllllll
        printf("\nfinal process 1.1 Killed id: %d\n",t);
    }


//sleep(50);

}
```

Output

```
usman@usman-hp-notebook:~/Desktop/OS Lab solution/asgn2_a  Process/q1$ gcc fork1.c
usman@usman-hp-notebook:~/Desktop/OS Lab solution/asgn2_a  Process/q1$ ./a.out
proc 1 and pid: 3769 and ppid: 3686
proc 1.1 and pid: 3770 and ppid: 3769
proc 1.2 and pid: 3771 and ppid: 3769
proc 1.1.1 and pid: 3772 and ppid: 3770
proc 1.1.2 and pid: 3773 and ppid: 3770
proc 1.2.1 and pid: 3774 and ppid: 3771
proc 1.1.1.1 and pid: 3777 and ppid: 3772
proc 1.2.2 and pid: 3775 and ppid: 3771
proc 1.2.1.1 and pid: 3778 and ppid: 3774
proc 1.2.2.1 and pid: 3779 and ppid: 3775

final process 1.1.1 id: 3772

final process 1.1.2 id: 3773

final process 1.1.1.1 id: 3777

final process 1.1 Killed id: 3770
```

Vfork( )

```
#include<stdio.h>
#include<unistd.h>
```

```
#include<stdlib.h>
#include<signal.h>
#include<string.h>


int main()
{
   pid_t p1_1=vfork();
   if(p1_1==0)
   {

      printf("c     proc 1.1 and pid: %d and ppid: %d\n",getpid(),getppid());
      exit(0);
      pid_t p1_1_1=vfork();
      if(p1_1_1==0)
      {
         int a=getpid();

         printf("c     proc 1.1.1 and pid: %d and ppid: %d\n",getpid(),getppid());
         exit(0);
         pid_t p1_1_1_1=vfork();
         if(p1_1_1_1==0)
         {

            printf("c     proc 1.1.1.1 and pid: %d and
ppid: %d\n",getpid(),getppid());
            exit(0);
         }
      }
      else if(p1_1_1>0)
      {
         pid_t p1_1_2=vfork();
         if(p1_1_2==0)
```

```
            {

                printf("c     proc 1.1.2 and pid: %d and
ppid: %d\n",getpid(),getppid());
                exit(0);
            }
        }
        exit(0);
    }
    else if(p1_1>0)
    {
        printf("p     proc 1 and pid: %d and ppid: %d\n",getpid(),getppid());
        pid_t p1_2=vfork();
        if(p1_2==0)
        {
            printf("c     proc 1.2 and pid: %d and ppid: %d\n",getpid(),getppid());
            exit(0);
            pid_t p1_2_1=vfork();
            if(p1_2_1==0)
            {
                printf("c     proc 1.2.1 and pid: %d and
ppid: %d\n",getpid(),getppid());
                pid_t p1_2_1_1=vfork();
                if(p1_2_1_1==0)
                {
                    printf("c     proc 1.2.1.1 and pid: %d and
ppid: %d\n",getpid(),getppid());
                    exit(0);
                }
            }
            if(p1_2_1>0)
            {
                pid_t p1_2_2=vfork();
                if(p1_2_2==0)
                {
                    printf("c     proc 1.2.2 and pid: %d and 
ppid: %d\n",getpid(),getppid());
                    exit(0);
                    pid_t p1_2_2_1=vfork();

                    if(p1_2_2_1==0)
                    {
                        printf("c     proc 1.2.2.1 and pid: %d and 
ppid: %d\n",getpid(),getppid());
                        exit(0);
                    }
                }
            }
        }
    }
    sleep(50);
}
```

Output

```
c     proc 1.1 and pid: 3504 and ppid: 3502
p     proc 1 and pid: 3502 and ppid: 2411
c     proc 1.2 and pid: 3505 and ppid: 3502
```

Write a program and Executes as a parent process, which occurs naturally. The parent process must output the following statement: "Parent process is running and about to fork to a child process.

```
#include<stdio.h>
#include<time.h>
int main()
{
    time_t mytime;
    mytime = time(NULL);
    printf("Outsider program is running. Time now is ");
    printf(ctime(&mytime));
}
```

Output

```
Outsider program is running. Time now is Fri Nov 30 16:25:52 2018
```

Exercise

1.

LAB 03

PIPE PROCESSING

5. PIPE PROCESSING

AIM:

To write a program for create a pope processing

ALGORITHM:

1. Start the program.
2. Declare the variables.
3. Read the choice.
4. Create a piping processing using IPC.
5. Assign the variable lengths
6. “strcpy” the message lengths.
7. To join the operation using IPC .
8. Stop the program

Write a program where process send and receive message using pipes and print the message in same order as send.

```
#include <stdio.h>
#include <unistd.h>
# define SIZE 10
int main()
{
   char message1[] = "message 1";
   char message2[] = "message 2";
   char message3[] = "message 3";
   char buffer[SIZE];
   int pip[2];
   if(pipe(pip)>=0)
   {
      write(pip[1],message1,SIZE);
      write(pip[1],message2,SIZE);
      write(pip[1],message3,SIZE);
      for(int i=0; i<3; i++)
      {
         read(pip[0],buffer,SIZE);
         printf("%s \n",buffer);
```

```
        }
        return 0;
    }

}
```

<u>Output</u>

```
message 1
message 2
message 3
```

Write a program to send message from parent to child using pipes.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

# define SIZE 10
int main()
{
    char message1[] = "message 1";
    char message2[] = "message 2";
    char message3[] = "message 3";
    char buffer[SIZE];
    int pip[2];
    printf("\n");
    if(pipe(pip)<0)
        exit(0);
    int pid = fork();
    if(pid > 0)
    {
        // write by the parent process
        write(pip[1],message1,SIZE);
        write(pip[1],message2,SIZE);
        write(pip[1],message3,SIZE);
    }
    else if(pid==0)
    {
        // read by the child process
        for(int i=0; i<3; i++)
        {
            read(pip[0],buffer,SIZE);
            printf("%s\n",buffer);
        }
    }
    else
    {
        printf("Ops Error in Fork");
    }

}
```

Output

```
message 1
message 2
message 3
```

write a program where you copy [‘Linux World!!’,’Understanding’,’ Concepts of’,’ Piping ’] in a character array using pipes.

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#define MSG_LEN 64
int main(){
```

```
int result;
int fd[2];
char message[MSG_LEN];
char recvd_msg[MSG_LEN];
result = pipe (fd);
//Creating a pipe//fd[0] is for reading and fd[1] is for writing
if (result < 0)
{
    perror("pipe ");
    exit(1);
}
strncpy(message,"Linux World!! ",MSG_LEN);
result=write(fd[1],message,strlen(message));
if (result < 0)
{
    perror("write");
    exit(2);
}
strncpy(message,"Understanding ",MSG_LEN);
result=write(fd[1],message,strlen(message));
if (result < 0)
{
    perror("write");
    exit(2);
}
strncpy(message,"Concepts of ",MSG_LEN);
result=write(fd[1],message,strlen(message));
if (result < 0)
{
    perror("write");
    exit(2);
}
strncpy(message,"Piping ", MSG_LEN);
```

```
    result=write(fd[1],message,strlen(message));
    if (result < 0)
    {
        perror("write");
        exit(2);
    }
    result=read (fd[0],recvd_msg,MSG_LEN);
    if (result < 0)
    {
        perror("read");
        exit(3);
    }
    printf("%s\n",recvd_msg);
    return 0;
}
```

Output

```
Linux World!! Understanding Concepts of Piping
```

Exercise

LAB 04

Threading creation and execution

6. Threading creation and execution

AIM:

To write a ‘c’ program for create and execute thread

Program1

```
#include <stdio.h>
#include <zconf.h>
#include <pthread.h>
void *kidfunc(void *p) {

    printf ("Kid ID is ---> %d\n", getpid( ));
}
int main ( ) {
    pthread_t kid ;
    pthread_create(&kid, NULL, kidfunc, NULL) ;
    printf ("Parent ID is ---> %d\n", getpid( )) ;
    pthread_join(kid, NULL) ;
    printf ("No more kid!\n") ;
}
```

<u>OUTPUT</u>

```
Parent ID is ---> 7250
Kid ID is ---> 7250
No more kid!
```

Program2

```
#include <stdio.h>
#include <pthread.h>

int glob_data = 5 ;

void *kidfunc(void *p) {

    printf ("Kid here. Global data was %d.\n", glob_data) ;
    glob_data = 15 ;
    printf ("Kid Again. Global data was now %d.\n", glob_data) ;
}
int main ( ) {
    pthread_t kid ;
    pthread_create (&kid, NULL, kidfunc, NULL) ;
    printf ("Parent here. Global data = %d\n", glob_data) ;
    glob_data = 10 ;

    pthread_join (kid, NULL) ;
    printf ("End of program. Global data = %d\n", glob_data) ;

}
```

OUTPUT

```
Parent here. Global data = 5
Kid here. Global data was 5.
Kid Again. Global data was now 15.
End of program. Global data = 15
```

Program3

```
/* Multithreaded C Program Using the Pthread API */

#include<pthread.h>

#include<stdio.h>
#include <stdlib.h>

int sum; /*This data is shared by the thread(s) */
void *runner(void *param); /* the thread */

```

```
int main(int argc, char *argv[]) {

   pthread_t tid; /* the thread identifier */
   pthread_attr_t attr; /* set of thread attributes */
   if(argc != 2)
   {

      fprintf(stderr,"usage: a.out <integer value>\n");
      exit(0);
   }

   if(atoi(argv[1]) < 0)

   {
      fprintf(stderr, "%d must be >= 0 \n", atoi(argv[1]));
      exit(0);
   }

/* get the default attributes */
   pthread_attr_init(&attr);

/*create the thread */
   pthread_create(&tid,&attr,runner,argv[1]);

/* Now wait for the thread to exit */
   pthread_join(tid,NULL);
   printf("sum = %d\n",sum);

}

/*The thread will begin control in this function */
void *runner(void *param)
```

```
{
   int upper = atoi(param);
   int i;

   sum=0;
   if(upper > 0)
   {
      for(i=1; i <= upper;i++)
         sum += i;
   }
   pthread_exit(0);
}
```

OUTPUT

```
usman@usman-hp-notebook:~/Desktop/C programming/thread$ ./main
usage: a.out <integer value>
usman@usman-hp-notebook:~/Desktop/C programming/thread$ ./main 10
sum = 55
```

Program4

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NUM_THREADS 5

void *PrintHello(void *threadid)
{
   printf("\n %p: Hello World!\n", threadid);
   pthread_exit(NULL);
}

int main( )
```

```
{
   pthread_t threads [NUM_THREADS];
   int rc, t;
   for(t=0; t < NUM_THREADS; t++)
   {
      printf ("Creating thread %d\n", t);
      rc = pthread_create (&threads[t], NULL, PrintHello, (void *) t );
      if (rc) {
         printf("ERROR; return code from pthread_create() is %d\n", rc);
         exit(-1);
      }
   }
   pthread_exit(NULL);
}
```

OUTPUT:

```
Creating thread 0
Creating thread 1

 (nil): Hello World!
Creating thread 2

 0x1: Hello World!
Creating thread 3

 0x2: Hello World!
Creating thread 4

 0x3: Hello World!

 0x4: Hello World!
```

Program5

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/wait.h>
#include <wait.h>
#include <stdlib.h>

int this_is_global;

void thread_func( void *ptr );

int main( ) {

    int local_main; int pid, status;

    pthread_t  thread1, thread2;

    printf("First, we create two threads to see better what context they share...\n");
    this_is_global=1000;

    printf("Set this_is_global=%d\n",this_is_global);
    pthread_create( &thread1, NULL, (void*)&thread_func, (void*) NULL);
    pthread_create(&thread2, NULL, (void*)&thread_func, (void*) NULL);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    printf("After threads, this_is_global=%d\n",this_is_global);
    printf("\n");
    printf("Now that the threads are done, let's call fork..\n");
    local_main=17;
    this_is_global=17;
```

```
  printf("Before fork(), local_main=%d, this_is_global=%d\n",local_main,
this_is_global);
  pid=fork();
  if (pid == 0) { /* this is the child */
    printf("In child, pid %d: &global: %x, &local: %x\n", getpid(),
&this_is_global, &local_main);
    local_main=13; this_is_global=23;
    printf("Child set local main=%d, this_is_global=%d\n",local_main,
this_is_global);
    exit(0);
  }
  else { /* this is parent */
    printf("In parent, pid %d: &global: %x, &local: %x\n", getpid(),
&this_is_global, &local_main);
    wait(&status);

    printf("In parent, local_main=%d, this_is_global=%d\n",local_main,
this_is_global);
  }
  exit(0);
}

void thread_func(void *dummy) {

  int local_thread;

  printf("Thread %d, pid %d, addresses: &global: %x, &local: %x\n",
pthread_self(),getpid(),&this_is_global, &local_thread); this_is_global++;

  printf("In Thread %d, incremented this_is_global=%d\n", pthread_self(),
this_is_global); pthread_exit(0);
}
```

OUTPUT

```
First, we create two threads to see better what context they share...
Set this_is_global=1000
Thread -1825200384, pid 8012, addresses: &global: 611e1014, &local: 93359ed4
In Thread -1825200384, incremented this_is_global=1001
Thread -1833593088, pid 8012, addresses: &global: 611e1014, &local: 92b58ed4
In Thread -1833593088, incremented this_is_global=1002
After threads, this_is_global=1002

Now that the threads are done, let's call fork..
Before fork(), local_main=17, this_is_global=17
In parent, pid 8012: &global: 611e1014, &local: c1f36aec
In child, pid 8016: &global: 611e1014, &local: c1f36aec
Child set local main=13, this_is_global=23
In parent, local_main=17, this_is_global=17
```

Program6

```
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>

int tot_items = 0 ;

struct kidrec { int data ; pthread_t id ;

} ;

#define NKIDS 50

void *kidfunc(void *p)
{

   int *ip = (int *)p ; int tmp, n ;
   tmp = tot_items ; for (n = 50000; n--; )
      tot_items = tmp + *ip ;
}

int main ( )

{
   struct kidrec kids[NKIDS] ;

   int m ;

   for (m=0; m<NKIDS; ++m)

   {
      kids[m].data = m+1 ;
      pthread_create (&kids[m].id, NULL, kidfunc, &kids[m].data) ;
   }
   for (m=0; m<NKIDS; ++m) pthread_join (kids[m].id, NULL) ;

   printf ("End of Program. Grand Total = %d\n", tot_items) ;

}
```

OUTPUT:

```
End of Program. Grand Total = 1120
```

Program7

```
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
#include <zconf.h>

#define NUM_THREADS 7
```

```
char *messages[NUM_THREADS];

void *PrintHello(void *threadid)

{
    int *id_ptr, taskid;

    sleep(1);

    id_ptr = (int *) threadid; taskid = *id_ptr;
    printf("\n %s from thread %d \n\n", messages[taskid], taskid);
    pthread_exit(NULL);
}

int main( )

{
    pthread_t threads[NUM_THREADS];
    int *taskids[NUM_THREADS];
    int rc, t;

    messages[0] = "English: Hello World!";
    messages[1] = "French: Bonjour, le monde!";
    messages[2] = "Spanish: Hola al mundo";
    messages[3] = "Klingon: Nuq neH!";
    messages[4] = "German: Guten Tag, Welt!";
    messages[5] = "Russian: Zdravstvytye, mir!";
    messages[6] = "Japan: Sekai e konnichiwa!";
    messages[7] = "Latin: Orbis, te saluto!";

    for(t=0;t<NUM_THREADS;t++)

    {
        taskids[t] = (int *) malloc(sizeof(int)); *taskids[t] = t;
        printf("Creating thread %d\n", t);
        rc = pthread_create(&threads[t], NULL, PrintHello, (void *) taskids[t]);
        if (rc)
        {
            printf("ERROR; return code from pthread_create() is %d\n", rc);
exit(-1);
        }
    }

    pthread_exit(NULL);

}
```

OUTPUT:

```
Creating thread 0
Creating thread 1
Creating thread 2
Creating thread 3
Creating thread 4
Creating thread 5
Creating thread 6
 French: Bonjour, le monde! from thread 1
 English: Hello World! from thread 0
 Klingon: Nuq neH! from thread 3
 German: Guten Tag, Welt! from thread 4
 Russian: Zdravstvytye, mir! from thread 5
 Japan: Sekai e konnichiwa! from thread 6
 Spanish: Hola al mundo from thread 2
```

Exercise

1. Write A C program to demonstrate use of pthread basic functions.
2. Write a C program to show multiple threads with global and static variables.

LAB 05

PRODUCER-CONSUMER PROBLEM USING SEMOPHERES

7. PRODUCER-CONSUMER PROBLEM USING SEMOPHERES

AIM:

To implement producer/consumer problem using semaphore.

ALGORITHM:

1. Declare variable for producer & consumer as pthread-t-tid produce tid consume.
2. Declare a structure to add items, semaphore variable set as struct.
3. Read number the items to be produced and consumed.
4. Declare and define semaphore function for creation and destroy.
5. Define producer function.
6. Define consumer function.
7. Call producer and consumer.
8. Stop the execution.

PROGRAM: (PRODUCER-CONSUMER PROBLEM)

```
#include<stdio.h>
void main()
{
    int buffer[10], bufsize, in, out, produce, consume, choice=0;
    in = 0;
    out = 0;
    bufsize = 10;
    while(choice !=3)
    {
        printf("\n1. Produce\t 2. Consume \t3. Exit");
        printf("\nEnter your choice: =");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1: {
                if ((in + 1) % bufsize == out)
                    printf("\nBuffer is Full");
                else {
                    printf("\nEnter the value: ");
                    scanf("%d", &produce);
                    buffer[in] = produce;
                    in = (in + 1) % bufsize;
                }
                break;
            }
            case 2: {
                if (in == out)
                    printf("\nBuffer is Empty");
                else {
                    consume = buffer[out];
                    printf("\nThe consumed value is %d", consume);
                    out = (out + 1) % bufsize;
                }
                break;
            }
        }
    }
}
```

Output

```
1. Produce       2. Consume    3. Exit
Enter your choice: =1

Enter the value: 5

1. Produce       2. Consume    3. Exit
Enter your choice: =2

The consumed value is 5
1. Produce       2. Consume    3. Exit
```

```
Enter your choice: =2

Buffer is Empty
1. Produce      2. Consume    3. Exit
Enter your choice: =1

Enter the value: 2

1. Produce      2. Consume    3. Exit
Enter your choice: =1

Enter the value: 4

1. Produce      2. Consume    3. Exit
Enter your choice: =2

The consumed value is 2
1. Produce      2. Consume    3. Exit
Enter your choice: =2

The consumed value is 4
1. Produce      2. Consume    3. Exit
Enter your choice: =2

Buffer is Empty
1. Produce      2. Consume    3. Exit
Enter your choice: =3

Process finished with exit code 0
```

LAB 06

Scheduling Algorithms

8. FIRST COME FIRST SERVE SCHEDULING

AIM:

To write the program to implement CPU & scheduling algorithm for first come first serve scheduling.

ALGORITHM:

1. Start the program.
2. Get the number of processes and their burst time.
3. Initialize the waiting time for process 1 and 0.
4. Process for(i=2;i<=n;i++),wt.p[i]=p[i-1]+bt.p[i-1].
5. The waiting time of all the processes is summed then average value time is calculated.
6. The waiting time of each process and average times are displayed
7. Stop the program

PROGRAM : (FIRST COME FIRST SERVE SCHEDULING)

```
#include <stdio.h>
struct process
{
   int pid;
   int bt;
   int wt;
   int tt;
}p[10];
int main()
{
   int i,n,totwt,tottt,avg1,avg2;
   printf("Enter the no of process : ");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
   {
      p[i].pid=i;
      printf("Enter the burst time of %d process : ",i);
      scanf("%d",&p[i].bt);
   }
   p[1].wt=0;
```

```
    p[1].tt=p[1].bt+p[1].wt;
    i=2;
    while(i<=n)
    {
        p[i].wt=p[i-1].bt+p[i-1].wt; p[i].tt=p[i].bt+p[i].wt;
        i ++;
    }
    i=1;
    totwt=tottt=0;
    printf("\nprocess id \t brust time\twait time\tterminate time\n");
    while(i<=n){
        printf("\t%d   \t\t%d  \t\t    %d\t\t\t  %d\n",p[i].pid,p[i].bt,p[i].wt,p[i].tt);
        totwt=p[i].wt+totwt;
        tottt=p[i].tt+tottt;
        i++;
    }
    avg1=totwt/n;
    avg2=tottt/n;
    printf("\naverage time W. R. T. waiting time=%d\naverage time W. R. T. terminate time=%d\n",avg1,avg2);
    return 0;
}
```

Output

```
Enter the no of process : 5
Enter the burst time of 1 process : 7
Enter the burst time of 2 process : 5
Enter the burst time of 3 process : 3
Enter the burst time of 4 process : 6
Enter the burst time of 5 process : 4

process id          burst time       wait time          terminate time
        1               7               0                   7
        2               5               7                   12
        3               3               12                  15
        4          6            15               21
        5          4            21               25

average time W. R. T. waiting time=11
average time W. R. T. terminate time=16

Process finished with exit code 0
```

9. SHORTEST JOB FIRST SCHEDULING

AIM:

To write a program to implement CPU scheduling algorithm for shortest job first scheduling.

ALGORITHM:

1. Start the program. Get the number of processes and their burst time.
2. Initialize the waiting time for process 1 as 0.
3. The processes are stored according to their burst time.
4. The waiting time for the processes are calculated a follows:

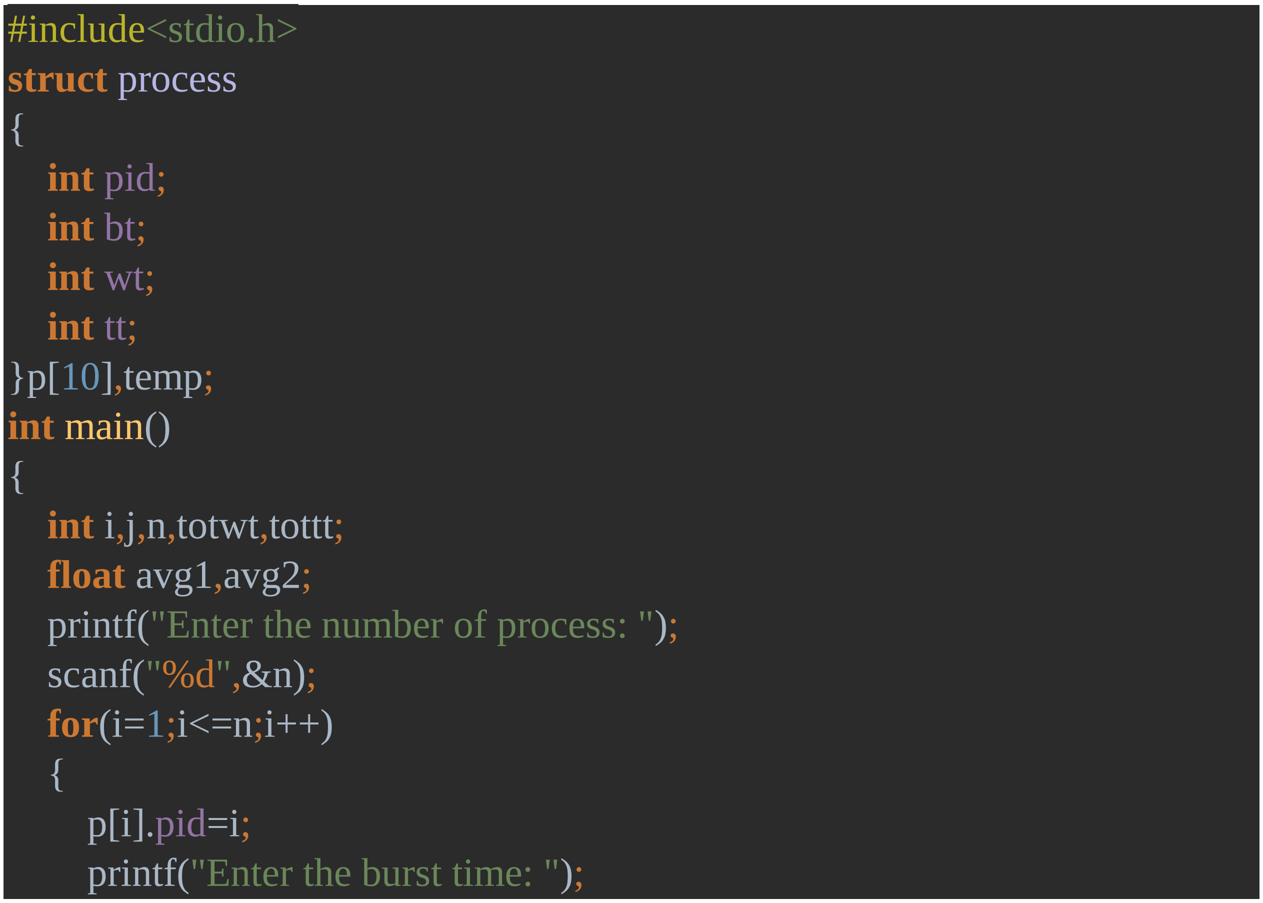
for(i=2;i<=n;i++).wt.p[i]=p[i=1]+bt.p[i-1].

5. The waiting time of all the processes summed and then the average time is calculate

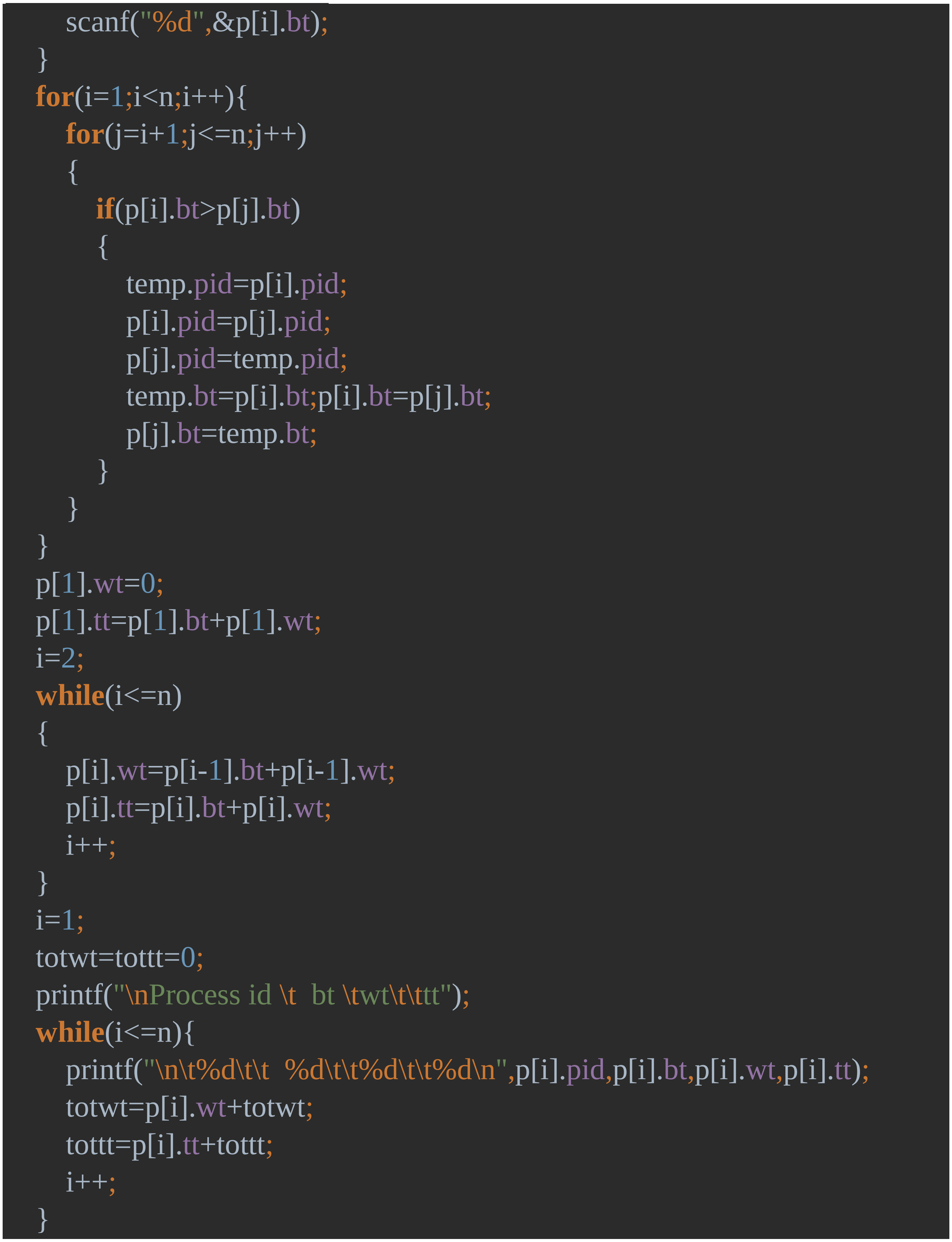
6. The waiting time of each processes and average time are displayed.

Stop the program.

PROGRAM: (SHORTEST JOB FIRST SCHEDULING)



```
#include<stdio.h>
struct process
{
   int pid;
   int bt;
   int wt;
   int tt;
}p[10],temp;
int main()
{
   int i,j,n,totwt,tottt;
   float avg1,avg2;
   printf("Enter the number of process: ");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
   {
      p[i].pid=i;
      printf("Enter the burst time: ");
```



```
    scanf("%d",&p[i].bt);
}
for(i=1;i<n;i++){
    for(j=i+1;j<=n;j++)
    {
        if(p[i].bt>p[j].bt)
        {
            temp.pid=p[i].pid;
            p[i].pid=p[j].pid;
            p[j].pid=temp.pid;
            temp.bt=p[i].bt;p[i].bt=p[j].bt;
            p[j].bt=temp.bt;
        }
    }
}
p[1].wt=0;
p[1].tt=p[1].bt+p[1].wt;
i=2;
while(i<=n)
{
    p[i].wt=p[i-1].bt+p[i-1].wt;
    p[i].tt=p[i].bt+p[i].wt;
    i++;
}
i=1;
totwt=tottt=0;
printf("\nProcess id \t  bt \twt\t\ttt");
while(i<=n){
    printf("\n\t%d\t\t  %d\t\t%d\t\t%d\n",p[i].pid,p[i].bt,p[i].wt,p[i].tt);
    totwt=p[i].wt+totwt;
    tottt=p[i].tt+tottt;
    i++;
}
```

```
    avg1=totwt/n;
    avg2=tottt/n;
    printf("Average time W. R. T. waiting time=%f\nAverage time W. R. T. 
terminate time=%f\n",avg1,avg2);
    return 0;
}
```

Output

```
Enter the number of process: 5
Enter the burst time: 7
Enter the burst time: 4
Enter the burst time: 2
Enter the burst time: 3
Enter the burst time: 5
Process id          bt            wt            tt
       3              2           0             2
       4              3           2             5
       2              4           5             9
       5              5           9             14
       1              7           14            21
Average time W. R. T. waiting time=6.000000
Average time W. R. T. terminate time=10.000000

Process finished with exit code 0
```

10. PRIORITY SCHEDULING

***AIM*:**

To write a ‘C’ program to perform priority scheduling.

***ALGORITHM*:**

11. Start the program.
12. Read burst time, waiting time, turn the around time and priority.
13. Initialize the waiting time for process 1 and 0.
14. Based up on the priority process are arranged
15. The waiting time of all the processes is summed and then the average waiting time
16. The waiting time of each process and average waiting time are displayed based on the priority.
17. Stop the program.

PROGRAM: (PRIORITY SCHEDULING)

```
#include<stdio.h>
struct process
{
   int pid;
   int bt;
   int wt;
   int tt;
   int prior;
}p[10],temp;

int main()
{
   int i,j,n;
   float totwt,tottt,arg1,arg2;
   printf("Enter the number of process : ");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
   {
      p[i].pid=i;
      printf("Enter the burst time : ");
```

```
    scanf("%d",&p[i].bt);
    printf("Enter the priority : ");
    scanf("%d",&p[i].prior);
}
for(i=1;i<n;i++)
{
    for(j=i+1;j<=n;j++)
    {
        if(p[i].prior>p[j].prior)
        {
            temp.pid=p[i].pid;
            p[i].pid=p[j].pid;
            p[j].pid=temp.pid;
            temp.bt=p[i].bt;
            p[i].bt=p[j].bt;
            p[j].bt=temp.bt;
            temp.prior=p[i].prior;
            p[i].prior=p[j].prior;
            p[j].prior=temp.prior;
        }
    }
}
p[i].wt=0;
p[1].tt=p[1].bt+p[1].wt;
i=2;
while(i<=n)
{
    p[i].wt=p[i-1].bt+p[i-1].wt;
    p[i].tt=p[i].bt+p[i].wt;
    i++;
}
i=1;
totwt=tottt=0;
```

```
    printf("\nProcess ID \t bt \t wt \t tt\n");
    while(i<=n)
    {
        printf("     %d      \t %d \t\t %d \t\t %d\t\n",p[i].pid,p[i].bt,p[i].wt,p[i].tt);
        totwt=p[i].wt+totwt;
        tottt=p[i].tt+tottt;
        i++;
    }
    arg1=totwt/n;
    arg2=tottt/n;
    printf("Average time W. R. T. waiting time=%f\nAverage time W. R. T. terminate time=%f\n",arg1,arg2);
    return 0;
}
```

Output

```
Enter the number of process : 5
Enter the burst time : 2
Enter the priority : 4
Enter the burst time : 7
Enter the priority : 0
Enter the burst time : 5
Enter the priority : 1
Enter the burst time : 3
Enter the priority : 2
Enter the burst time : 6
Enter the priority : 3
Process ID      bt              wt              tt
    2           7               0               7
    3           5               7               12
    4           3               12              15
    5           6               15              21
    1           2               21              23
Average time W. R. T. waiting time=11.000000
Average time W. R. T. terminate time=15.60000
```

11.ROUND ROBIN SCHEDULING

AIM:

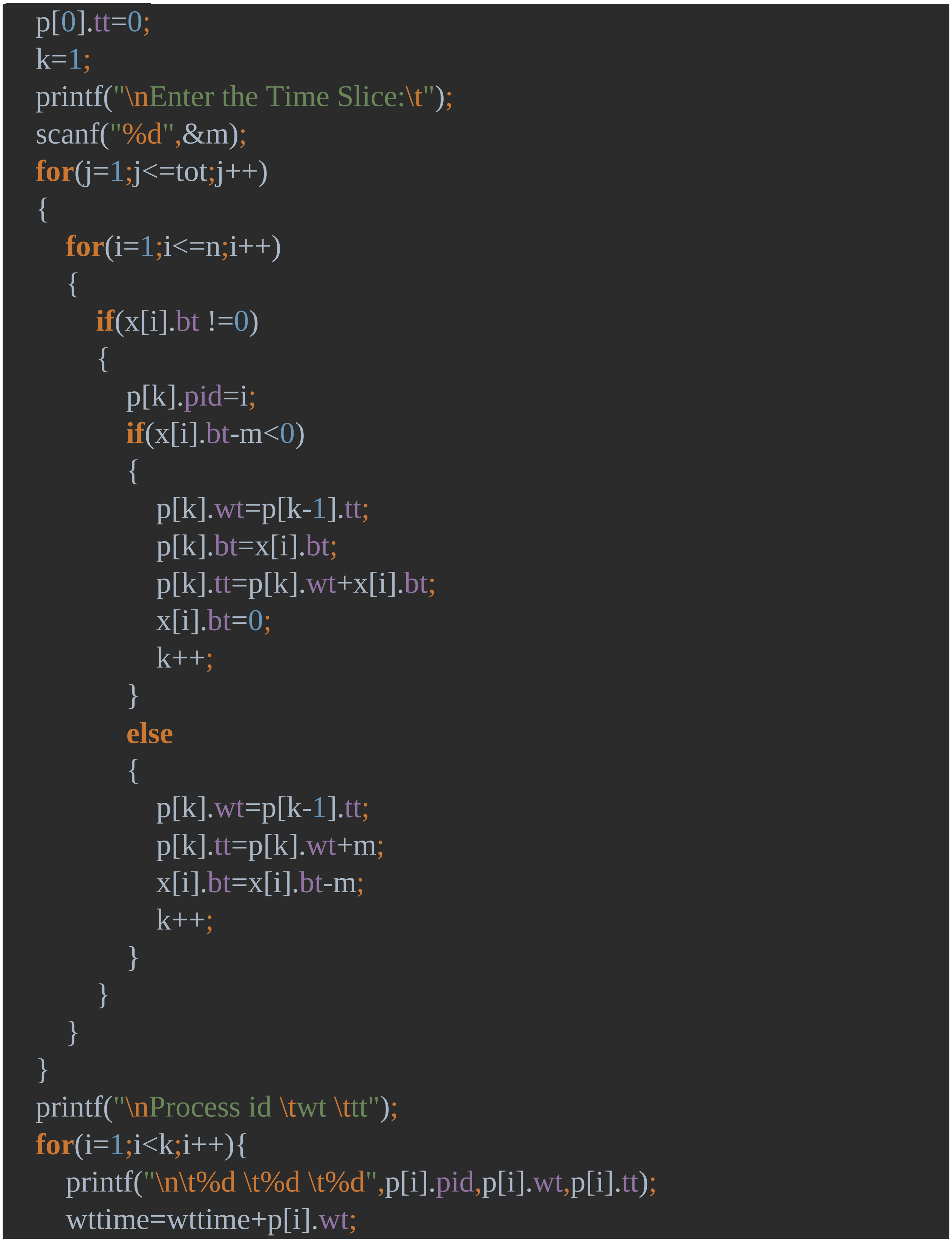
To write a program to implement cpu scheduling for Round Robin Scheduling.

ALGORITHM:

1. Get the number of process and their burst time.
2. Initialize the array for Round Robin circular queue as ‘0’.
3. The burst time of each process is divided and the quotients are stored on the round Robin array.
4. According to the array value the waiting time for each process and the average time are calculated as line the other scheduling.
5. The waiting time for each process and average times are displayed.
6. Stop the program.

PROGRAM : (ROUND ROBIN SCHEDULING)

```
#include<stdio.h>
struct process
{
    int pid,bt,tt,wt;
};
int main()
{
    struct process x[10],p[30];
    int i,j,k,tot=0,m,n;
    float wttime=0.0,tottime=0.0,a1,a2;
    printf("\nEnter the number of process:\t");
    scanf("%d",&n);
    for(i=1;i<=n;i++){
        x[i].pid=i;
        printf("\nEnter the Burst Time:\t");
        scanf("%d",&x[i].bt);
        tot=tot+x[i].bt;
    }
    printf("\nTotal Burst Time:\t%d",tot);
```



```
p[0].tt=0;
k=1;
printf("\nEnter the Time Slice:\t");
scanf("%d",&m);
for(j=1;j<=tot;j++)
{
    for(i=1;i<=n;i++)
    {
        if(x[i].bt !=0)
        {
            p[k].pid=i;
            if(x[i].bt-m<0)
            {
                p[k].wt=p[k-1].tt;
                p[k].bt=x[i].bt;
                p[k].tt=p[k].wt+x[i].bt;
                x[i].bt=0;
                k++;
            }
            else
            {
                p[k].wt=p[k-1].tt;
                p[k].tt=p[k].wt+m;
                x[i].bt=x[i].bt-m;
                k++;
            }
        }
    }
}
printf("\nProcess id \twt \ttt");
for(i=1;i<k;i++){
    printf("\n\t%d \t%d \t%d",p[i].pid,p[i].wt,p[i].tt);
    wttime=wttime+p[i].wt;
```

```
        tottime=tottime+p[i].tt;
        a1=wttime/n;
        a2=tottime/n;
    }
    printf("\n\nAverage Waiting Time:\t%f",a1);
    printf("\n\nAverage TurnAround Time:\t%f",a2);
    return 0;
}
```

Output

```
Enter the number of process:  5
Enter the Burst Time: 7
Enter the Burst Time: 3
Enter the Burst Time: 5
Enter the Burst Time: 2
Enter the Burst Time: 8
Total Burst Time:       25
Enter the Time Slice:  3
Process id       wt        tt
          1                0         3
          2                3         6
          3                6         9
          4                9         11
          5                11        14
          1                14        17
          3                17        19
          5                19        22
          1                22        23
          5                23        25

Average Waiting Time:           24.799999
Average TurnAround Time:   29.799999
```

LAB 07

FIRST FIT MEMORY MANAGEMENT

12.FIRST FIT MEMORY MANAGEMENT

AIM:

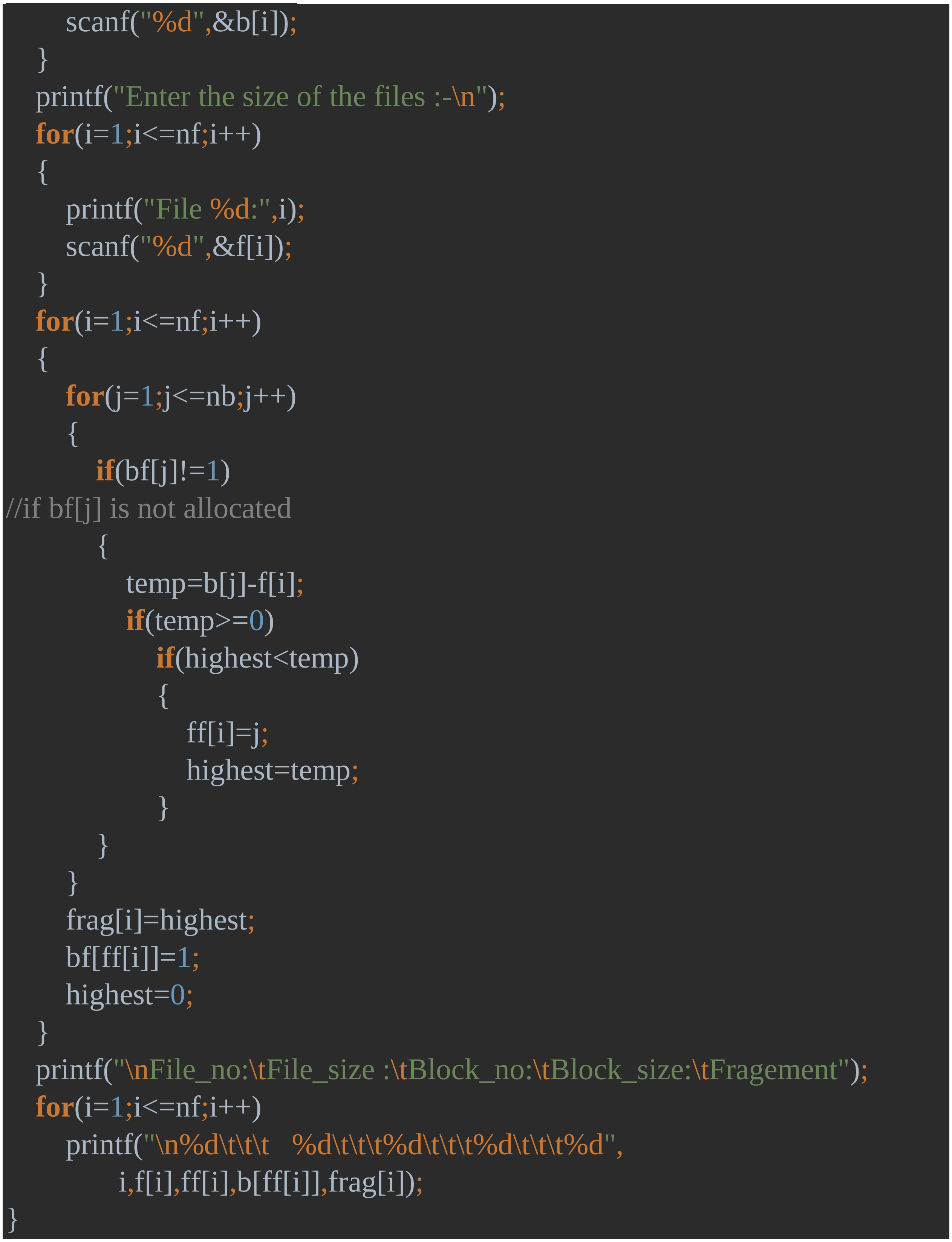
To implement first fit, best fit algorithm for memory management.

ALGORITHM:

1. Start the program.
2. Get the segment size, number of process to be allocated and their corresponding size.
3. Get the options. If the option is ‘2’ call first fit function.
4. If the option is ‘1’ call best fit function. Otherwise exit.
5. For first fit, allocate the process to first possible segment which is free and set the personnel slap as ‘1’. So that none of process to be allocated to segment which is already allocated and vice versa.
6. For best fit, do the following steps,.
7. Sorts the segments according to their sizes.
8. Allocate the process to the segment which is equal to or slightly greater than the process size and set the flag as the ‘1’ .So that none of the process to be allocated to the segment which is already allocated and vice versa.
9. Stop the program

PROGRAM: (FIRST FIT MEMORY MANAGEMENT)

```
#include<stdio.h>
#define max 25
void main()
{
   int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
   static int bf[max],ff[max];
   printf("\n\tMemory Management Scheme - First Fit");
   printf("\nEnter the number of blocks:");
   scanf("%d",&nb);
   printf("Enter the number of files:");
   scanf("%d",&nf);
   printf("\nEnter the size of the blocks:-\n");
   for(i=1;i<=nb;i++)
   {
      printf("Block %d:",i);
```



```
        scanf("%d",&b[i]);
    }
    printf("Enter the size of the files :-\n");
    for(i=1;i<=nf;i++)
    {
        printf("File %d:",i);
        scanf("%d",&f[i]);
    }
    for(i=1;i<=nf;i++)
    {
        for(j=1;j<=nb;j++)
        {
            if(bf[j]!=1)
//if bf[j] is not allocated
            {
                temp=b[j]-f[i];
                if(temp>=0)
                    if(highest<temp)
                    {
                        ff[i]=j;
                        highest=temp;
                    }
            }
        }
        frag[i]=highest;
        bf[ff[i]]=1;
        highest=0;
    }
    printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
    for(i=1;i<=nf;i++)
        printf("\n%d\t\t\t   %d\t\t\t%d\t\t\t%d\t\t\t%d",
               i,f[i],ff[i],b[ff[i]],frag[i]);
}
```

Output

```
          Memory Management Scheme - First Fit
Enter the number of blocks:3
Enter the number of files:2

Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:1
File 2:4

File_no:      File_size :        Block_no:          Block_size:         Fragement
1                      1                  3                    7                    6
2                      4                  1                    5                    1

Process finished with exit code 0
```

LAB 08

FILE MANIPULATION

13.FILE MANIPULATION - I

AIM:

To write a program for file manipulation for displays the file and directory in memory

ALGORITHM:

1. Start the program.
2. Use the pre defined function list out the files in directory..
3. Main function is used to check the file present in the directory or not.
4. Using the file pointer in the file to that the argument is less than a times means
5. print
6. By using if loop check in file, open two means print error
7. Stop the program.

PROGRAM: (FILE MANIPULATION - I)

```
#include <dirent.h>
#include <stdio.h>
int main(void)
{
   DIR *d;
   struct dirent *dir;
   d = opendir(".");
   if (d)
   {
      while ((dir = readdir(d)) != NULL)
      {
         printf("%s\n", dir->d_name);
      }
      closedir(d);
   }
   return(0);
}
```

Output

untitled

```
.
CMakeFiles
CMakeCache.txt
file
cmake_install.cmake
stdfile
STDFILE
student
..
untitled.cbp
Makefile

Process finished with exit code 0
```

14.FILE MANIPULATION-II

AIM:

To write a program performs file manipulation.

ALGORITHM:

1. Start the program.
2. Declare the arguments for file open and file create.
3. print the file in directory otherwise display the error message error in creation
4. if check the files in directory
5. close the files and directory
6. Stop the program.

PROGRAM : (FILE MANIPULATION-II)

```
#include<stdio.h>
#include<sys/stat.h>
#include<time.h>
#include <zconf.h>
#include <fcntl.h>

int main(int ag,char*arg[])
{
    char buf[100];
    struct stat s;
    int fd1,fd2,n;
    fd1=open(arg[1],0);
    fd2=creat(arg[2],0777);
    stat(arg[2],&s);
    if(fd2==-1)
        printf("ERROR IN CREATION");
    while((n=read(fd1,buf,sizeof(buf)))>0)
    {
        if(write(fd2,buf,n)!=n)
        {
            close(fd1);
            close(fd2);
        }
    }
    printf("\t\n UID FOR FILE.......>%d "
          "\n FILE ACCESSTIME.....>%s "
          "\n FILE MODIFIED TIME........>%s "
          "\n FILE I-NODENUMBER......>%d "
          "\n PERMISSION FORFILE.....>%o\n\n"
          "",s.st_uid,ctime(&s.st_atime),
          ctime(&s.st_mtime),s.st_mode);
    close(fd1);
    close(fd2);
}
```

output

```
UID FOR FILE.......>1000
FILE ACCESSTIME.....>Mon Nov 12 22:14:38 2018

FILE MODIFIED TIME........>Mon Nov 12 22:14:38 2018

FILE I-NODENUMBER......>33261
PERMISSION FORFILE.....>0
```

LAB 09

SIMULATE PAGE REPLACEMENT ALGORITHMS

15.SIMULATE PAGE REPLACEMENT ALGORITHMS FIFO

AIM:

To Simulate FIFO page replacement algorithms.

ALGORITHM:

1. Start the program
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames in First in first out order.
7. Display the number of page faults.
8. Stop the program

PROGRAM:(SIMULATE PAGE REPLACEMENT ALGORITHMS FIFO)

```
#include<stdio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int main()
{
   printf("FIFI PAGE REPLACEMENT ALGORITHM\n");
   printf("Enter no.of frames : ");
   scanf("%d",&nof);
   printf("Enter number of Pages : ");
   scanf("%d",&nor);
   for(i=0;i<nor;i++)
   {
      printf("Enter the Page No : ");
      scanf("%d",&ref[i]);
   }

   printf("\nThe given Pages are:");
```

```
    for(i=0;i<nor;i++)
        printf("%4d",ref[i]);
    for(i=1;i<=nof;i++)
        frm[i]=-1;
    printf("\n");
    for(i=0;i<nor;i++)
    {
        flag=0;
        printf("\n\t page no %d->\t",ref[i]);
        for(j=0;j<nof;j++)
        {
            if(frm[j]==ref[i])
            {flag=1;
                break;
            }}
        if(flag==0)
        {
            pf++;
            victim++;
            victim=victim%nof;
            frm[victim]=ref[i];
            for(j=0;j<nof;j++)
                printf("%4d",frm[j]);
        }
    }
    printf("\n\nNo.of pages faults...%d",pf);
    return 1;
}
```

Output

```
FIFI PAGE REPLACEMENT ALGORITHM
Enter no.of frames : 4
Enter number of Pages : 9
Enter the Page No : 6
Enter the Page No : 4
Enter the Page No : 2
Enter the Page No : 1
Enter the Page No : 3
Enter the Page No : 5
Enter the Page No : 7
Enter the Page No : 9
Enter the Page No : 8

The given Pages are:   6   4   2   1   3   5   7   9   8

        page no 6->   6  -1  -1  -1
        page no 4->   6   4  -1  -1
        page no 2->   6   4   2  -1
        page no 1->   6   4   2   1
        page no 3->   3   4   2   1
        page no 5->   3   5   2   1
        page no 7->   3   5   7   1
        page no 9->   3   5   7   9
        page no 8->   8   5   7   9

No.of pages faults...9
Process finished with exit code 1
```

16.SIMULATE PAGE REPLACEMENT ALGORITHMS: LRU

AIM:

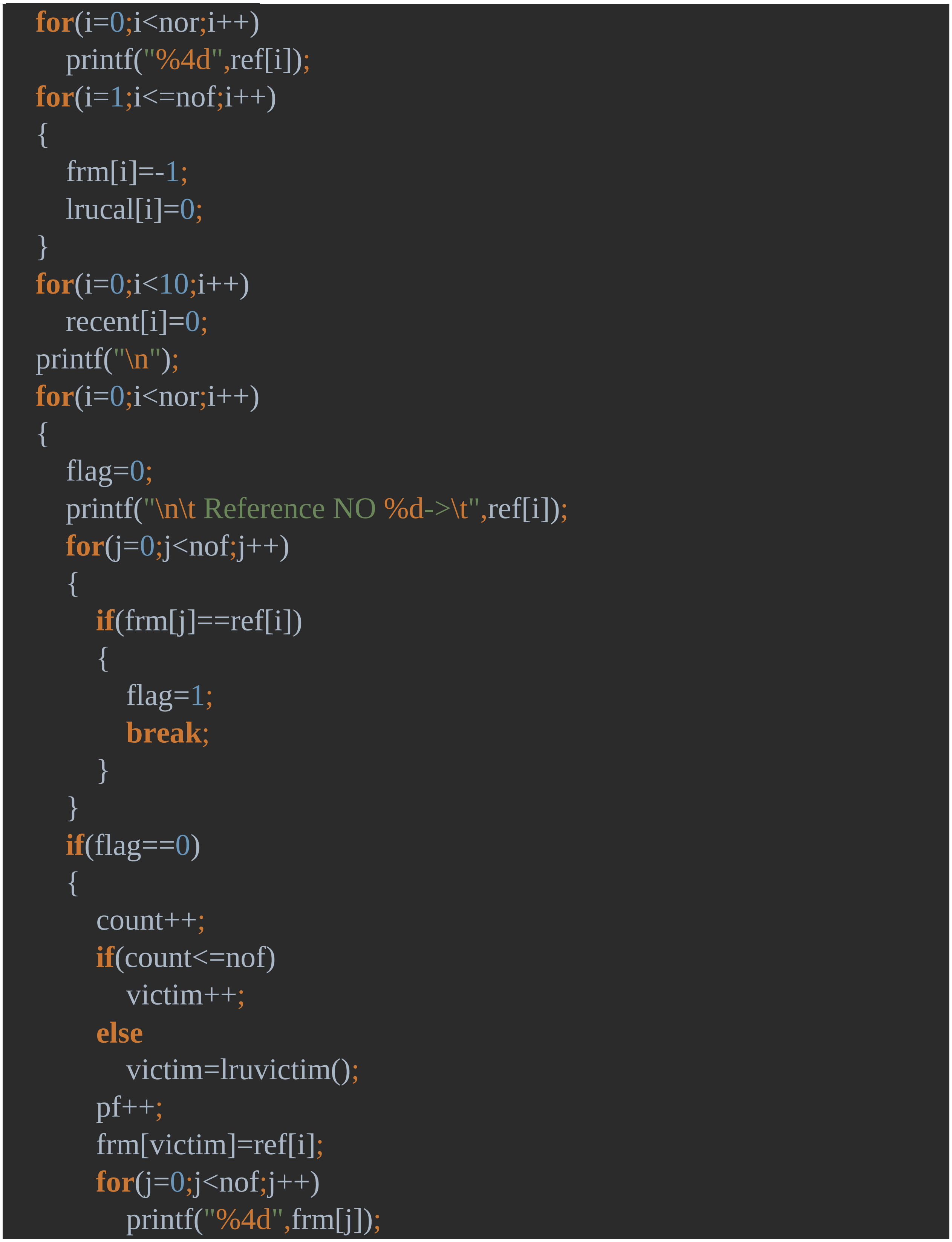
To Simulate LRU page replacement algorithms

ALGORITHM:

1. Start
1. Read the number of frames
2. Read the number of pages
3. Read the page numbers
4. Initialize the values in frames to -1
5. Allocate the pages in to frames by selecting the page that has not been used for the longest
6. period of time.
7. Display the number of page faults.
8. Stop

PROGRAM: (SIMULATE PAGE REPLACEMENT ALGORITHMS: LRU)

```
#include<stdio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],lrucal[50],count=0;
int lruvictim();
int main()
{
   printf("LRU PAGE REPLACEMENT ALGORITHM\n");
   printf("Enter no.of Frames : ");
   scanf("%d",&nof);
   printf("Enter number of Pages : ");
   scanf("%d",&nor);
   for(i=0;i<nor;i++)
   {
      printf("Enter the Page No : ");
      scanf("%d",&ref[i]);
   }
   printf("\nThe given Pages are:");
```



```
for(i=0;i<nor;i++)
    printf("%4d",ref[i]);
for(i=1;i<=nof;i++)
{
    frm[i]=-1;
    lrucal[i]=0;
}
for(i=0;i<10;i++)
    recent[i]=0;
printf("\n");
for(i=0;i<nor;i++)
{
    flag=0;
    printf("\n\t Reference NO %d->\t",ref[i]);
    for(j=0;j<nof;j++)
    {
        if(frm[j]==ref[i])
        {
            flag=1;
            break;
        }
    }
    if(flag==0)
    {
        count++;
        if(count<=nof)
            victim++;
        else
            victim=lruvictim();
        pf++;
        frm[victim]=ref[i];
        for(j=0;j<nof;j++)
            printf("%4d",frm[j]);
        }
        recent[ref[i]]=i;
    }
    printf("\n\nNo.of page faults...%d",pf);
    return 99;
}
int lruvictim()
{
    int i,j,temp1,temp2;
    for(i=0;i<nof;i++)
    {
        temp1=frm[i];
        lrucal[i]=recent[temp1];
    }
    temp2=lrucal[0];
    for(j=1;j<nof;j++)
    {
        if(temp2>lrucal[j])
            temp2=lrucal[j];
    }
    for(i=0;i<nof;i++)
        if(ref[temp2]==frm[i])
            return i;
    return 0;
}
```

Output

```
LRU PAGE REPLACEMENT ALGORITHM
Enter no.of Frames : 4
Enter number of Pages : 9
Enter the Page No : 8
Enter the Page No : 6
Enter the Page No : 4
Enter the Page No : 2
Enter the Page No : 1
Enter the Page No : 3
Enter the Page No : 5
Enter the Page No : 7
Enter the Page No : 9

The given Pages are:   8   6   4   2   1   3   5   7   9

             Reference NO 8->        8  -1  -1  -1
             Reference NO 6->        8   6  -1  -1
             Reference NO 4->        8   6   4  -1
             Reference NO 2->        8   6   4   2
             Reference NO 1->        1   6   4   2
             Reference NO 3->        1   3   4   2
             Reference NO 5->        1   3   5   2
             Reference NO 7->        1   3   5   7
             Reference NO 9->        9   3   5   7

No.of page faults...9
Process finished with exit code 99
```

17.SIMULATE PAGE REPLACEMENT ALGORITHMS: OPTIMAL

AIM:

To create program for optimal page replacement algorithms.

ALGORITHM:

1. Start the program
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames by selecting the page that will not be used for the longest period of time.
7. Display the number of page faults.
8. Stop the program

PROGRAM: (SIMULATE PAGE REPLACEMENT ALGORITHMS: OPTIMAL)

```
#include<stdio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],optcal[50],count=0;
int optvictim(int);
void main()
{
    printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHN\n");
    printf("Enter the no.of frames : ");
    scanf("%d",&nof);
    printf("Enter the no.of Pages : ");
    scanf("%d",&nor);

    for(i=0;i<nor;i++)
    {
        printf("Enter the Page No : ");
        scanf("%d",&ref[i]);
    }
    printf("\nThe given Pages are:");
    for(i=0;i<nor;i++)
        printf("%4d",ref[i]);
    for(i=0;i<nof;i++)
    {
        frm[i]=-1;optcal[i]=0;
    }
    for(i=0;i<10;i++)
        recent[i]=0;
    printf("\n");
    for(i=0;i<nor;i++)
    {
        flag=0;
        printf("\n\tref no %d ->\t",ref[i]);
        for(j=0;j<nof;j++)
        {
            if(frm[j]==ref[i])
            {
                flag=1;
                break;
            }
        }
        if(flag==0)
        {
            count++;
            if(count<=nof)
                victim++;
            else
                victim=optvictim(i);
            pf++;
            frm[victim]=ref[i];
```

```
            for(j=0;j<nof;j++)
                printf("%4d",frm[j]);
        }
    }
    printf("\n Number of page faults: %d",pf);
}
int optvictim(int index)
{
    int i,j,temp,notfound;
    for(i=0;i<nof;i++)
    {
        notfound=1;
        for(j=index;j<nor;j++)
            if(frm[i]==ref[j])
            {
                notfound=0;
                optcal[i]=j;
                break;
            }
        if(notfound==1)
            return i;
    }
    temp=optcal[0];
    for(i=1;i<nof;i++)
        if(temp<optcal[i])
            temp=optcal[i];
    for(i=0;i<nof;i++)
        if(frm[temp]==frm[i])
            return i;
    return 0;
}
```

OUTPUT:

```
OPTIMAL PAGE REPLACEMENT ALGORITHN
Enter the no.of frames : 3
Enter the no.of Pages : 7
Enter the Page No : 7
Enter the Page No : 3
Enter the Page No : 5
Enter the Page No : 1
Enter the Page No : 2
Enter the Page No : 6
Enter the Page No : 4

The given Pages are:   7   3   5   1   2   6   4

        ref no 7 ->       7  -1  -1
        ref no 3 ->       7   3  -1
        ref no 5 ->       7   3   5
        ref no 1 ->       1   3   5
        ref no 2 ->       2   3   5
        ref no 6 ->       6   3   5
        ref no 4 ->       4   3   5
 Number of page faults: 7
Process finished with exit code 26
```

LAB 10

SIMULATE ALGORITHM FOR DEADLOCK PREVENTION

18. SIMULATE ALGORITHM FOR DEADLOCK PREVENTION

AIM :

To Simulate Algorithm for Deadlock prevention

ALGORITHM:

1. Start the program
2. Attacking Mutex condition: never grant exclusive access. But this may not be possible for several resources.
3. Attacking preemption: not something you want to do.
4. Attacking hold and wait condition: make a process hold at the most 1 resource
5. At a time. Make all the requests at the beginning. Nothing policy. If you feel, retry.
6. Attacking circular wait: Order all the resources. Make sure that the requests are issued in the
7. Correct order so that there are no cycles present in the resource graph. Resources numbered 1 ... n.
8. Resources can be requested only in increasing
9. Order. i.e. you cannot request a resource whose no is less than any you may be holding.
10. Stop the program

PROGRAM: (SIMULATE ALGORITHM FOR DEADLOCK PREVENTION)

```
#include<stdio.h>
int max[10][10], alloc[10][10], need[10][10];
int avail[10],i,j,p,r,finish[10]={0},flag=0;
int fun();
int main()
{
    printf("SIMULATION OF DEADLOCK PREVENTION\n");
    printf("Enter no. of processes, resources : \n");
    scanf("%d%d",&p,&r);
    printf("Enter allocation matrix : \n");
    for(i=0;i<p;i++)
        for(j=0;j<r;j++)
            scanf("%d",&alloc[i][j]);
    printf("Enter max matrix : \n");
    for(i=0;i<p;i++) /*reading the maximum matrix and availale matrix*/
```

```
        for(j=0;j<r;j++)
            scanf("%d",&max[i][j]);
    printf("enter available matrix : \n");
    for(i=0;i<r;i++)
        scanf("%d",&avail[i]);
    for(i=0;i<p;i++)
        for(j=0;j<r;j++)
            need[i][j]=max[i][j]-alloc[i][j];
    fun(); /*calling function*/
    if(flag==0)
    {
        if(finish[i]!=1)
        {
            printf("\n\n Failing :Mutual exclusion");
            for(j=0;j<r;j++)
            { /*checking for mutual exclusion*/
                if(avail[j]<need[i][j])
                    avail[j]=need[i][j];
            }fun();
            printf("By allocating required resources to process %d dead lock is
prevented \n",i);
            printf("lack of preemption\n");
            for(j=0;j<r;j++)
            {
                if(avail[j]<need[i][j])
                    avail[j]=need[i][j];
                alloc[i][j]=0;
            }
            fun( );
            printf("\n\n dead lock is prevented by allocating needed resources");
            printf(" \n \n failing:Hold and Wait condition ");
            for(j=0;j<r;j++)
            {
                if(avail[j]<need[i][j])
                    avail[j]=need[i][j];
            }
            fun( );
            printf("\n AVOIDING ANY ONE OF THE CONDITION, U CAN
PREVENT DEADLOCK");
        }
    }
    return 0;
}
int fun()
{
    while(1)
    {
        for(flag=0,i=0;i<p;i++)
        {
            if(finish[i]==0)
            {
                for(j=0;j<r;j++)
                {
                    if(need[i][j]<=avail[j])
                        continue;
                    else
                        break;
                }
                if(j==r)
                {
                    for(j=0;j<r;j++)
                        avail[j]+=alloc[i][j];
                    flag=1;
                    finish[i]=1;
                }
            }
        }
        if(flag==0)
            break;
    }
    return 0;
}
```

OUTPUT:

```
SIMULATION OF DEADLOCK PREVENTION
Enter no. of processes, resources :
3
2
Enter allocation matrix :
2 4 5
3 4 5
Enter max matrix :
4 3 4
5 6 1
enter available matrix :
2
2


Failing :Mutual exclusion By allocating required resources to process 3 dead lock is prevented
lack of preemption

dead lock is prevented by allocating needed resources

 failing:Hold and Wait condition
 AVOIDING ANY ONE OF THE CONDITION, U CAN PREVENT DEADLOCK
Process finished with exit code 0
```