18CSC205J - Operating Systems

LAB MANUAL

Bachelor of Technology

Semester IV

Academic Year: 2021-2022 EVEN SEMESTER

FACULTY OF ENGINEERING AND TECHNOLOGY

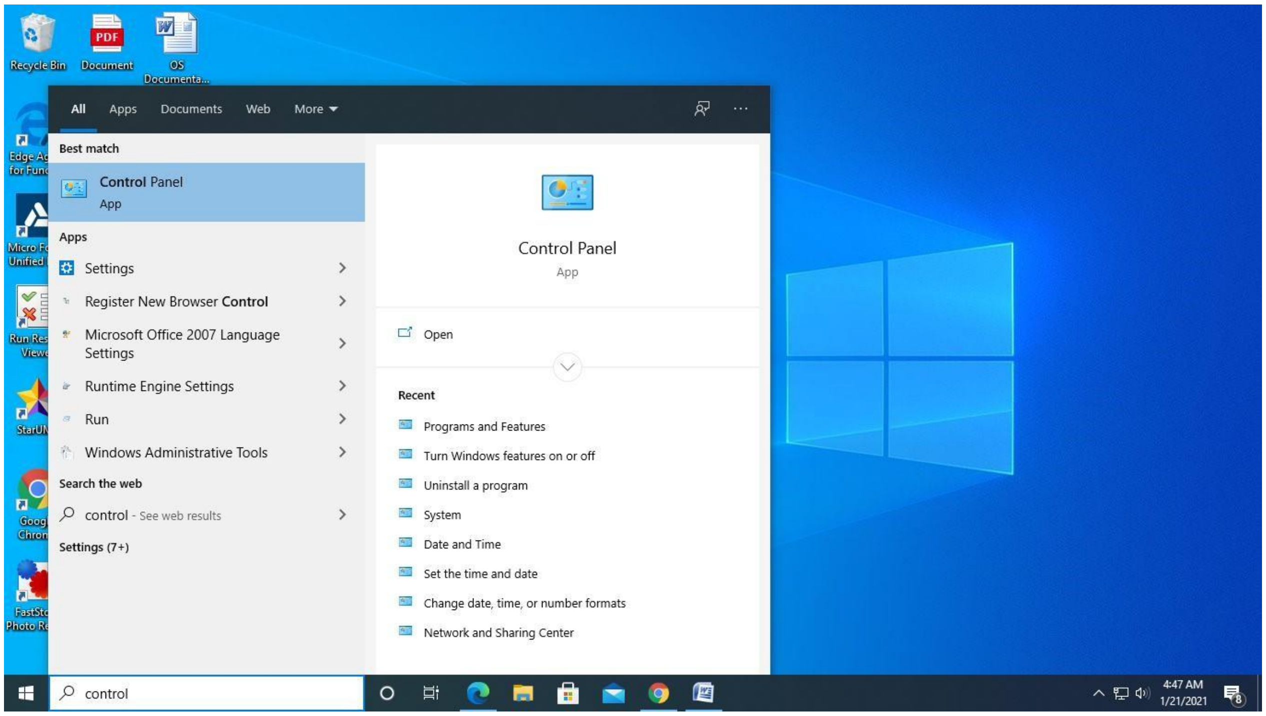
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

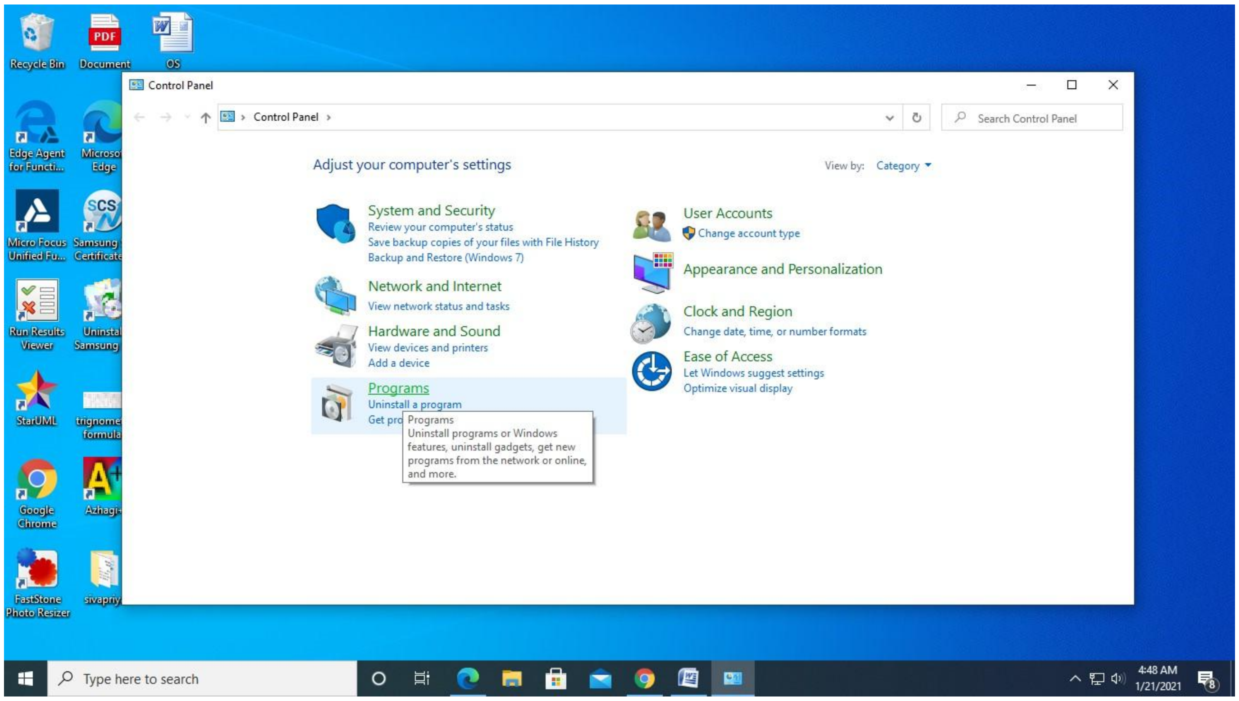
RAMAPURAM CAMPUS,

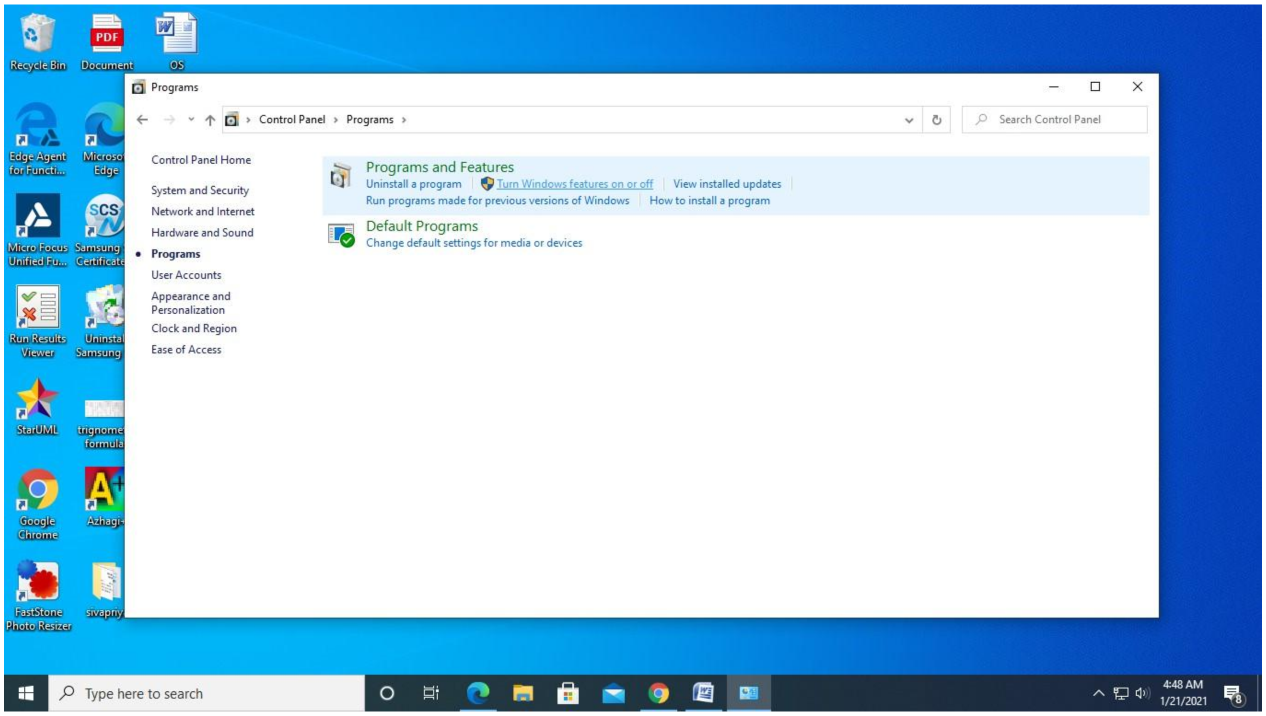
CHENNAI- 600 089

Ubuntu installation guidelines in windows

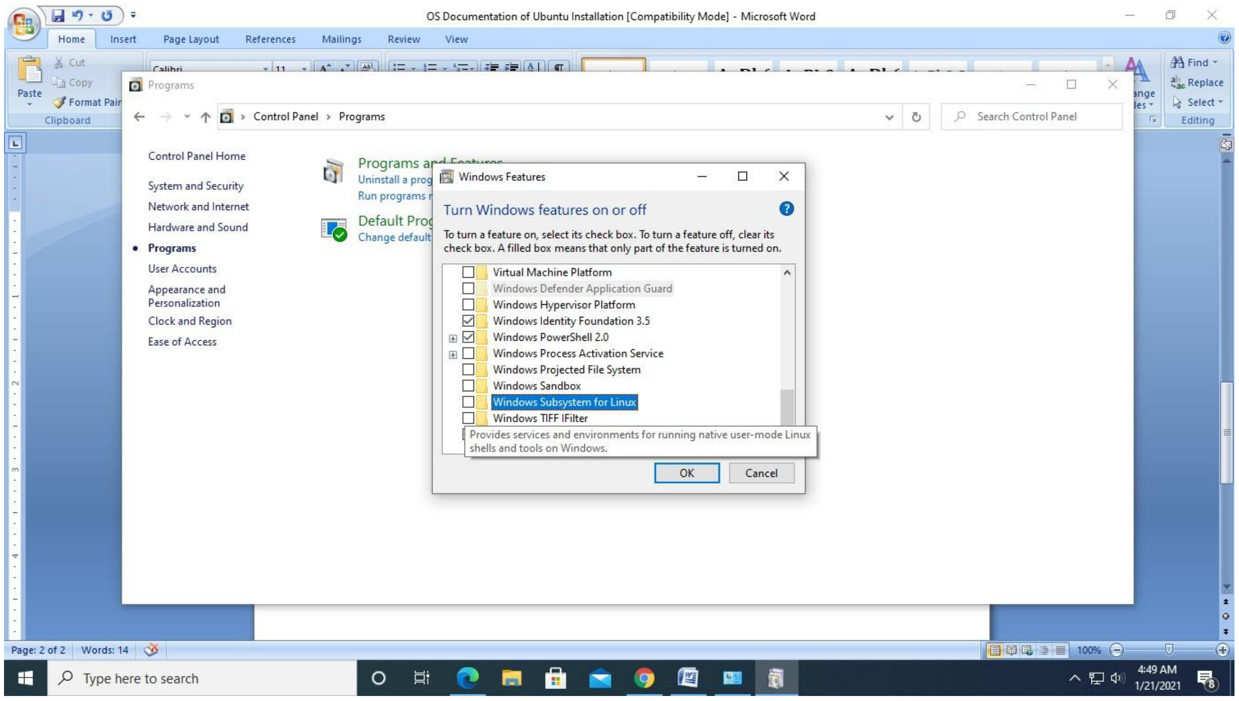
Installing Ubuntu in windows 10- 64 bit




Control Panel
Control Panel
Search Control Panel
Adjust your computer's settings
View by: Category
System and Security
Review your computer's status
Save backup copies of your files with File History
Backup and Restore (Windows 7)
Network and Internet
View network status and tasks
Hardware and Sound
View devices and printers
Add a device
Programs
Uninstall a program
User Accounts
Change account type
Appearance and Personalization
Clock and Region
Change date, time, or number formats
Ease of Access
Let Windows suggest settings
Optimize visual display
Programs
Uninstall programs or Windows features, uninstall gadgets, get new programs from the network or online, and more.
Type here to search
4:48 AM
1/21/2021



Select Turn Windows features On or Off



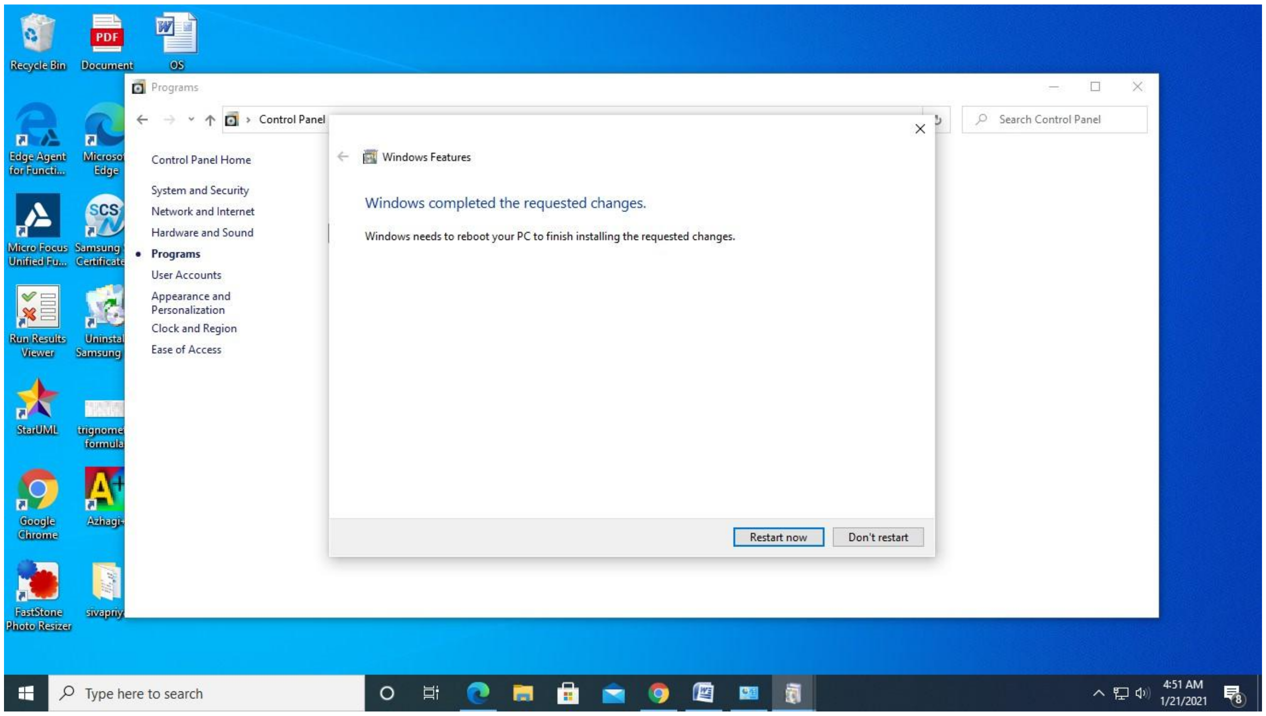
Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- [ ] Virtual Machine Platform
- [ ] Windows Defender Application Guard
- [ ] Windows Hypervisor Platform
- [✓] Windows Identity Foundation 3.5
- [✓] Windows PowerShell 2.0
- [ ] Windows Process Activation Service
- [ ] Windows Projected File System
- [ ] Windows Sandbox
- [ ] Windows Subsystem for Linux
- [ ] Windows TIFF IFilter

Provides services and environments for running native user-mode Linux shells and tools on Windows.

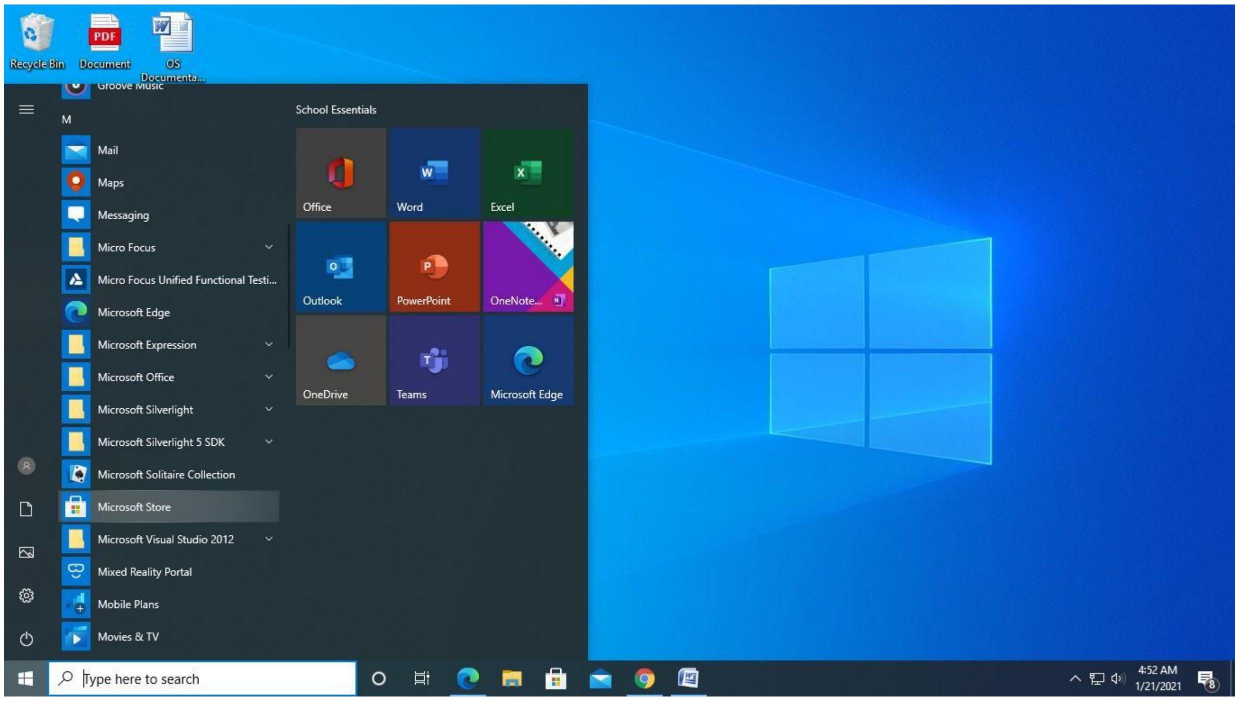
Select Windows subsystem for linux then press Ok



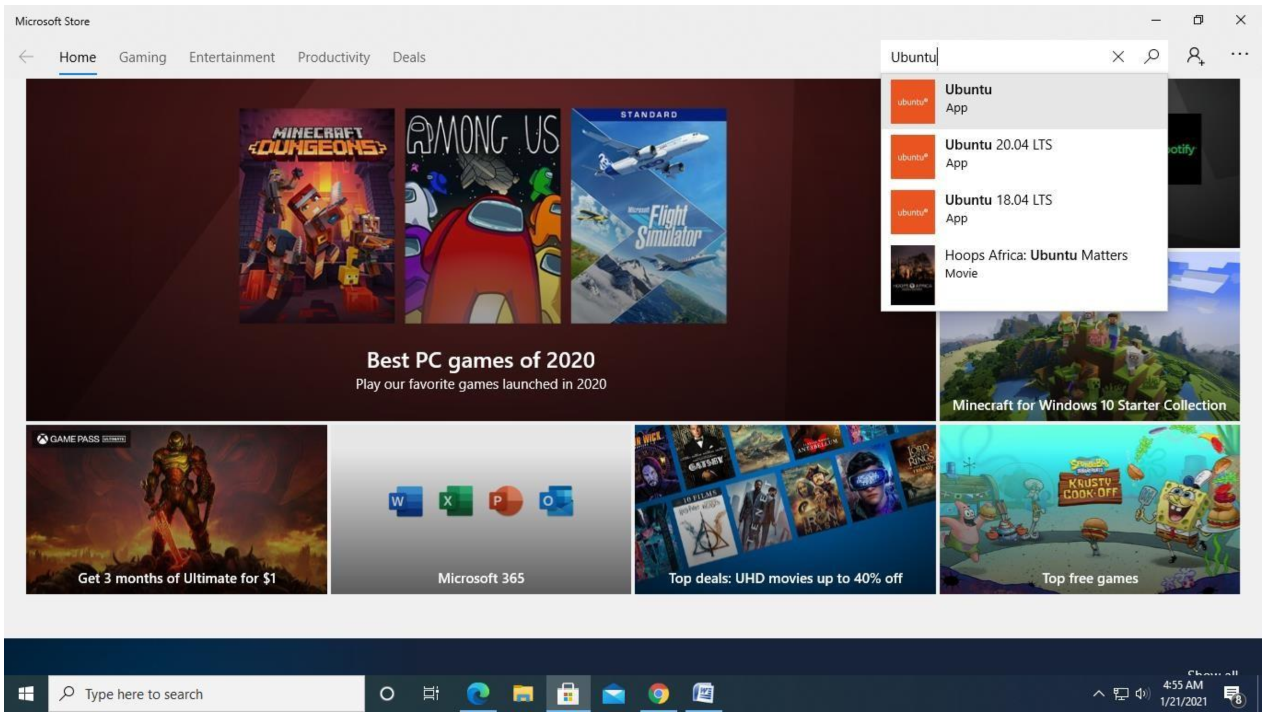
Windows completed the requested changes.

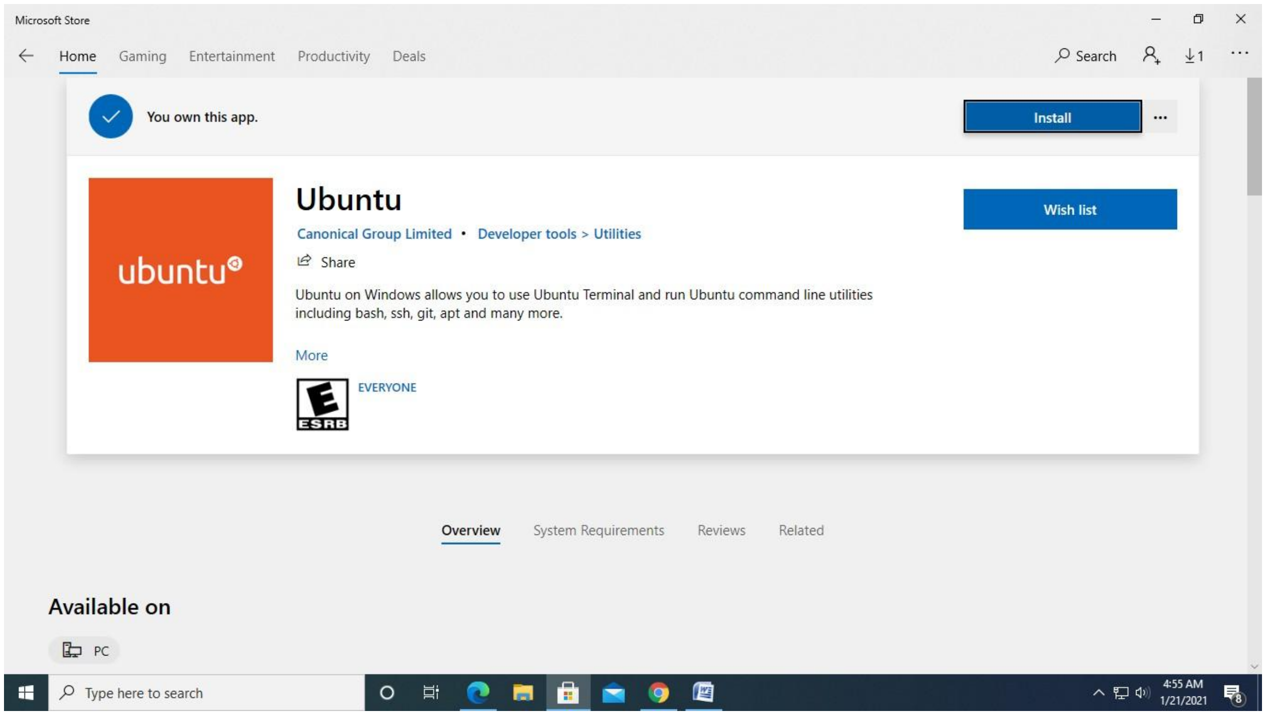
Windows needs to reboot your PC to finish installing the requested changes.

Now restart the PC to apply the changes



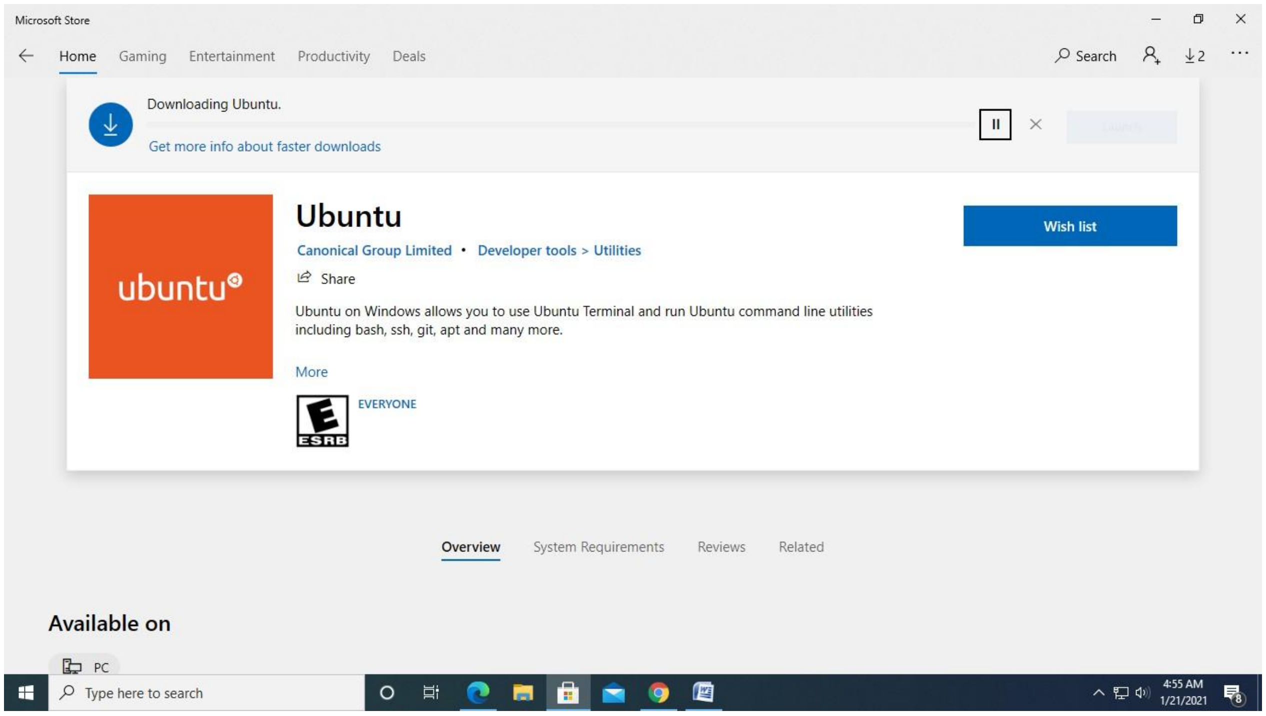
Choose Microsoft Store and search for Ubuntu and Install it





Ubuntu on Windows allows you to use Ubuntu Terminal and run Ubuntu command line utilities including bash, ssh, git, apt and many more.

Click on Install



Downloading Ubuntu.

Get more info about faster downloads

Ubuntu

Canonical Group Limited • Developer tools > Utilities

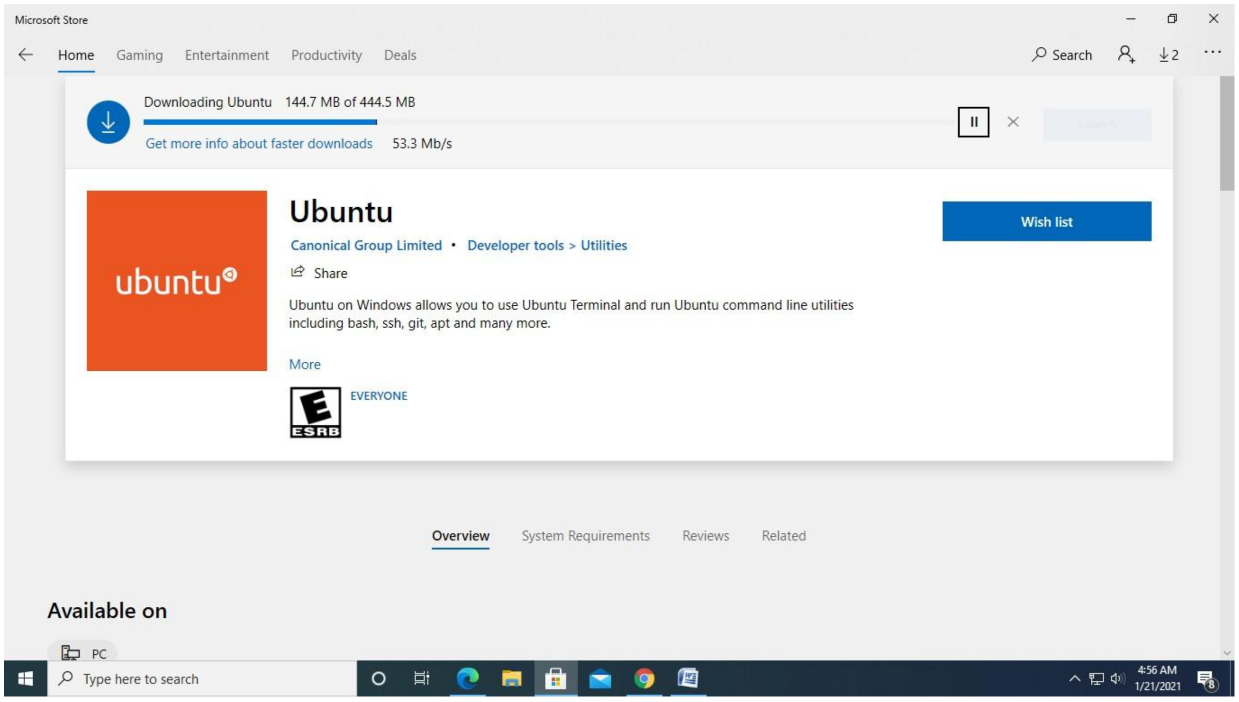
Share

Ubuntu on Windows allows you to use Ubuntu Terminal and run Ubuntu command line utilities including bash, ssh, git, apt and many more.

More

EVERYONE

Available on



Downloading Ubuntu 144.7 MB of 444.5 MB

Get more info about faster downloads 53.3 Mb/s

Ubuntu

Canonical Group Limited • Developer tools > Utilities

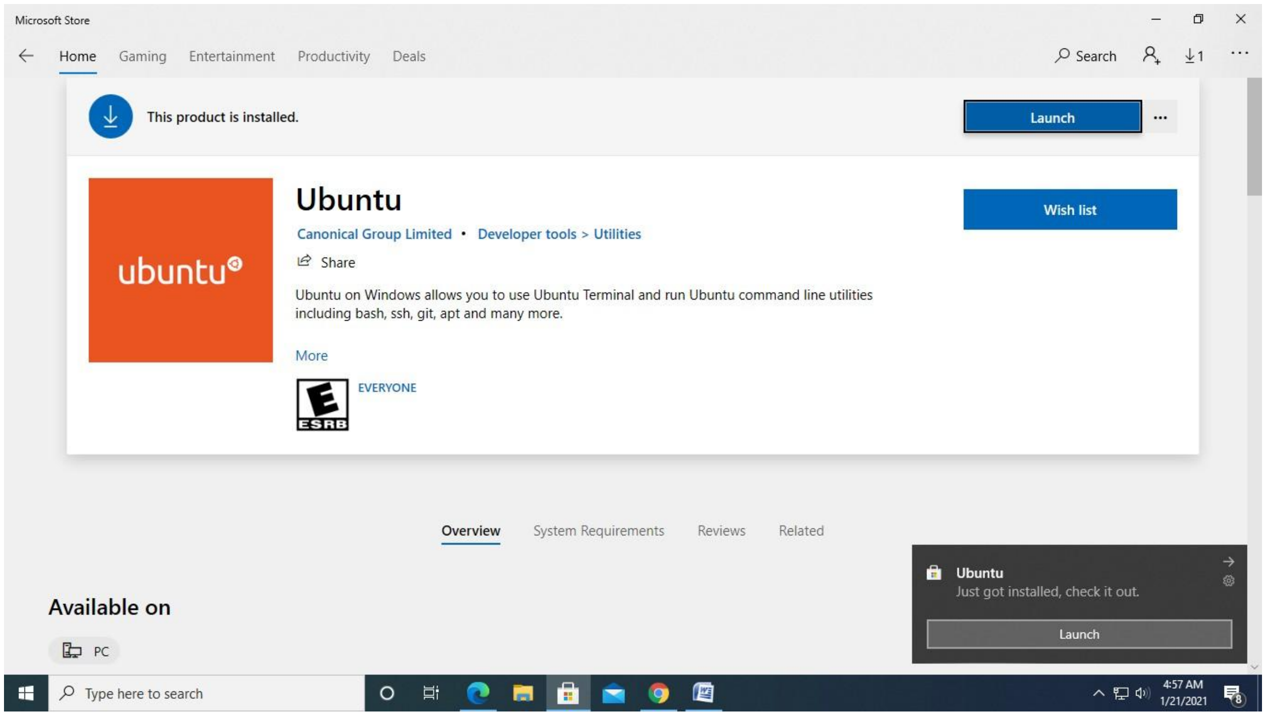
Share

Ubuntu on Windows allows you to use Ubuntu Terminal and run Ubuntu command line utilities including bash, ssh, git, apt and many more.

More

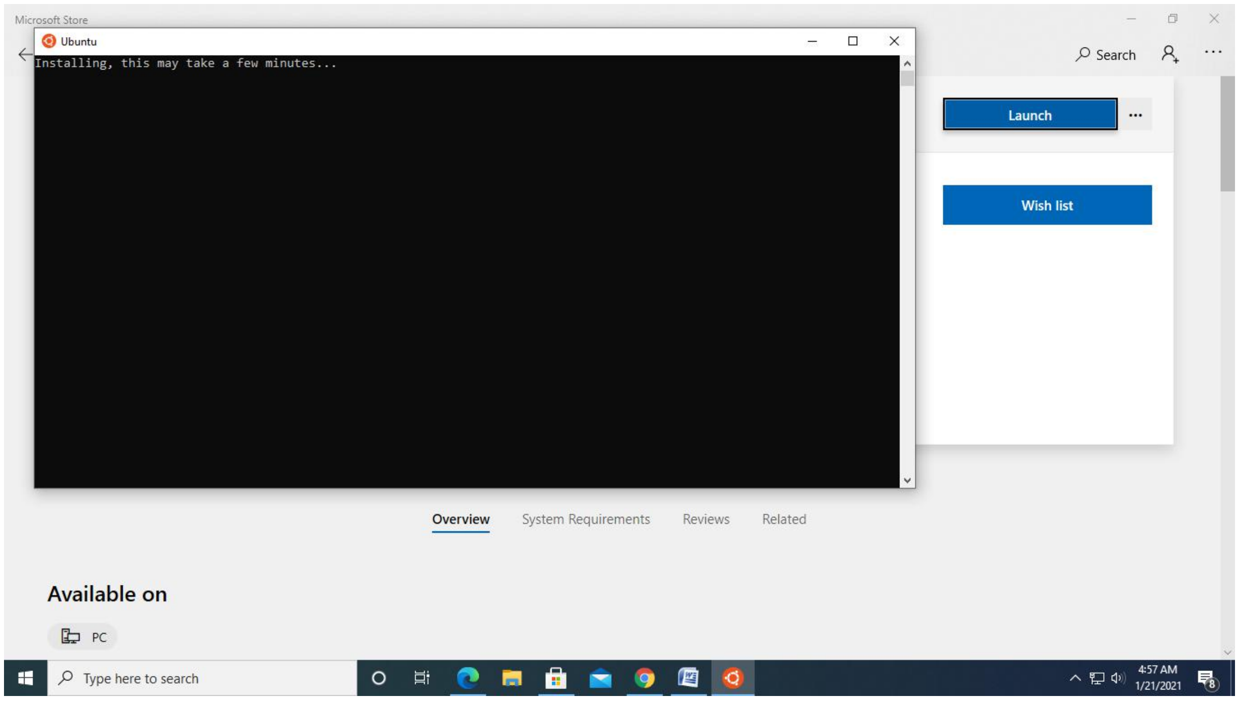
EVERYONE

Available on

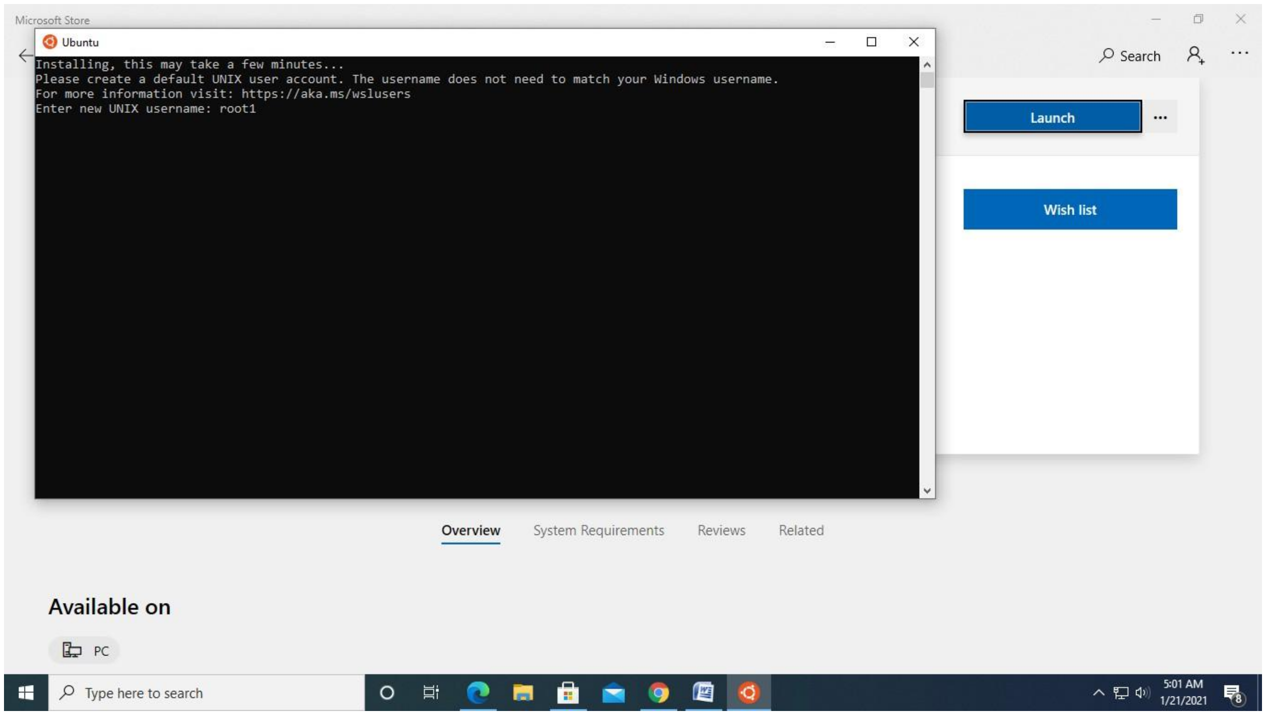


Ubuntu on Windows allows you to use Ubuntu Terminal and run Ubuntu command line utilities including bash, ssh, git, apt and many more.

Now Launch the Ubuntu



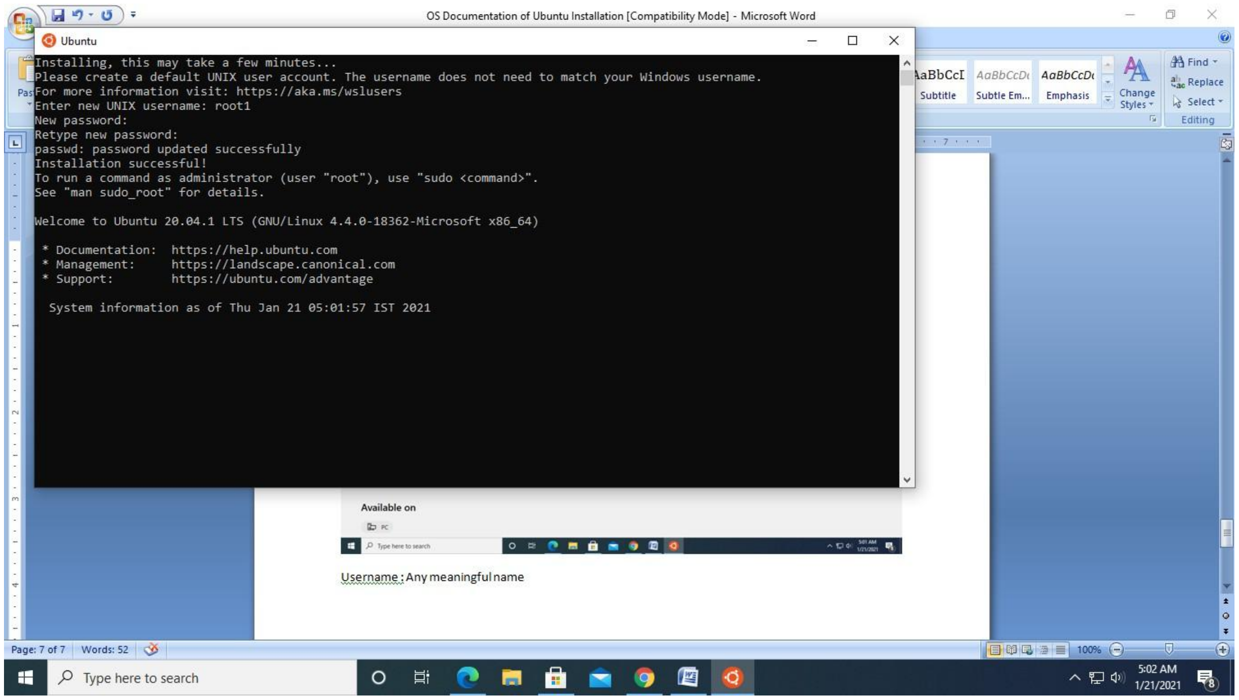
```
Installing, this may take a few minutes...
```



```
Installing, this may take a few minutes...
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: root1
```

Username : Any meaningful

name Password: any name

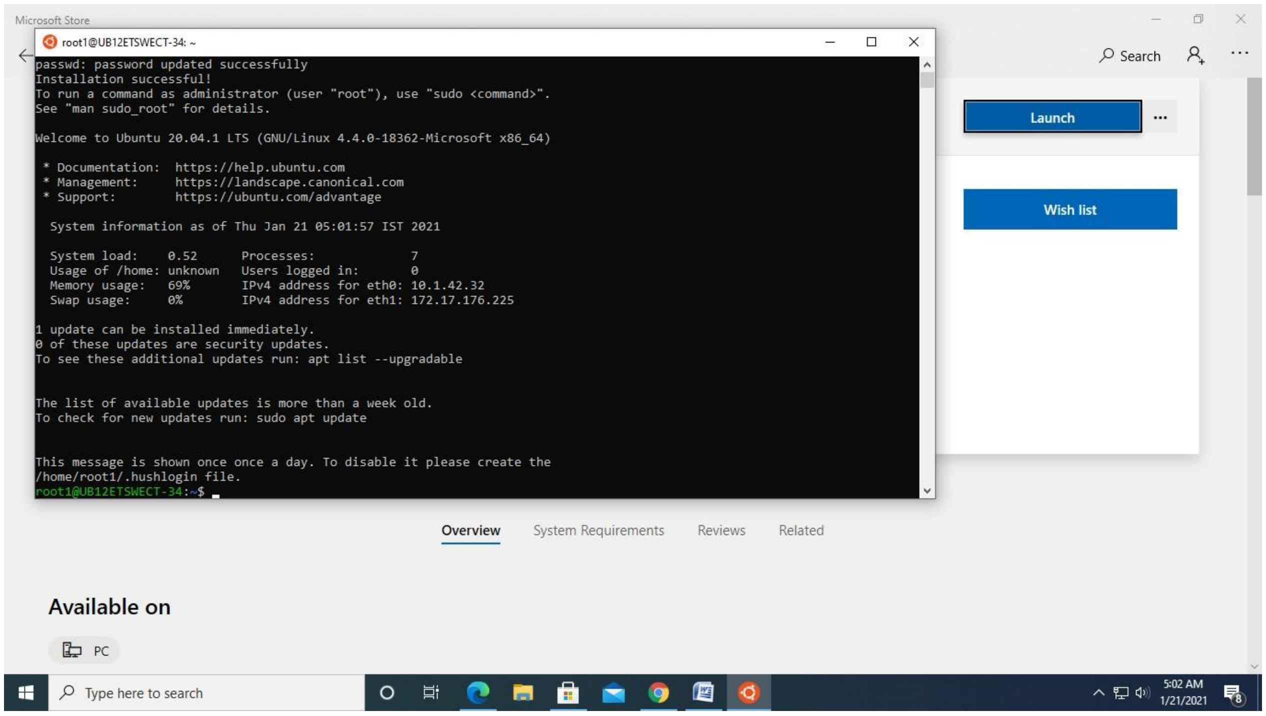


```
Installing, this may take a few minutes...
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: root1
New password:
Retype new password:
passwd: password updated successfully
Installation successful!
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 4.4.0-18362-Microsoft x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu Jan 21 05:01:57 IST 2021
```



```
passwd: password updated successfully
Installation successful!
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 4.4.0-18362-Microsoft x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu Jan 21 05:01:57 IST 2021

  System load:    0.52      Processes:             7
  Usage of /home: unknown   Users logged in:       0
  Memory usage:   69%       IPv4 address for eth0: 10.1.42.32
  Swap usage:     0%        IPv4 address for eth1: 172.17.176.225

1 update can be installed immediately.
0 of these updates are security updates.
To see these additional updates run: apt list --upgradable


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


This message is shown once once a day. To disable it please create the
/home/root1/.hushlogin file.
root1@UB12ETSWECT-34:~$
```

Finally you will get the prompt on successful installation of Ubuntu in windows 10 64 bit

LIST OF EXPERIMENTS

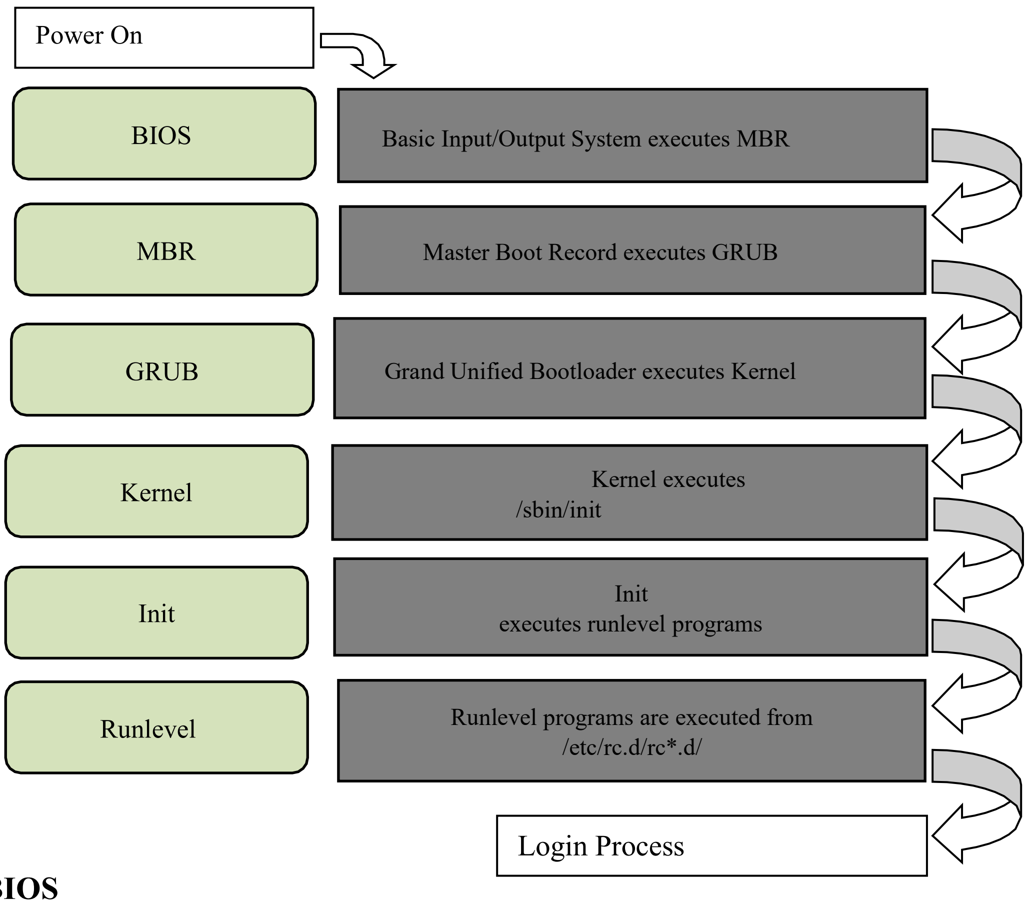
Ex.No.	Experiment Name
1	Understanding the booting process of Linux
2	Understanding the Linux file system
3	Understanding the various Phases of Compilation of a ‘C’ Program
4	System admin commands – Basics
5	System admin commands – Simple task automations
6	Linux commands
7	Shell Programs – Basic level
8	Process Creation
9	Overlay concept
10	Overlay concept
11	IPC using Pipes
12	IPC using shared memory and Message queues
13	Process synchronization
14	Study of OS161
15	Understanding the OS161 filesystem and working with test programs

Ex. No. 1	UNDERSTANDING THE BOOTING PROCESS OF LINUX

Objective:

To study various stages of Linux boot process.

Press the power button on your system, and after few moments you see the Linux login prompt. From the time you press the power button until the Linux login prompt appears, the following sequence occurs. The following are the 6 high level stages of a typical Linux boot process.


Power On
BIOS
Basic Input/Output System executes MBR
MBR
Master Boot Record executes GRUB
GRUB
Grand Unified Bootloader executes Kernel
Kernel
Kernel executes /sbin/init
Init
Init executes runlevel programs
Runlevel
Runlevel programs are executed from /etc/rc.d/rc*.d/
Login Process

Step 1.BIOS

- BIOS stands for Basic Input/ Output System
- Performs some system integrity checks
- Searches, loads, and executes the boot loader program.
- It looks for boot loader in floppy, CD-ROMs, or hard drive. You can press a key (typically F12 or F2, but it depends on your system) during the BIOS startup to change the boot sequence.
- Once the boot loader program is detected and loaded into the memory, BIOS gives the control to it.
- So, in simple terms BIOS loads and executes the MBR boot loader.

Step 2. MBR

- MBR stands for Master Boot Record.
- It is located in the 1st sector of the bootable disk. Typically /dev/hda, or /dev/sda
- MBR is less than 512 bytes in size. This has three components 1) primary boot loader info in 1st 446 bytes 2) partition table info in next 64 bytes 3) mbr validation check in last 2 bytes.
- It contains information about GRUB (or LILO in old systems).
- So, in simple terms MBR loads and executes the GRUB boot loader.

Step 3. GRUB

- GRUB stands for Grand Unified Bootloader.
- If you have multiple kernel images installed on your system, you can choose which one to be executed.
- GRUB displays a splash screen, waits for few seconds, if you don't enter anything, it loads the default kernel image as specified in the grub configuration file.
- GRUB has the knowledge of the filesystem (the older Linux loader LILO didn't understand filesystem).
- Grub configuration file is /boot/grub/grub.conf (/etc/grub.conf is a link to this).
- As you notice from the above info, it contains kernel and initrd image.
- So, in simple terms GRUB just loads and executes Kernel and initrd images.

Step 4. Kernel

- Mounts the root file system as specified in the "root=" in grub.conf
- Kernel executes the /sbin/init program
- Since init was the 1st program to be executed by Linux Kernel, it has the process id (PID) of 1. Do a 'ps -ef | grep init' and check the pid.
- initrd stands for Initial RAM Disk.
- initrd is used by kernel as temporary root file system until kernel is booted and the real root file system is mounted. It also contains necessary drivers compiled inside, which helps it to access the hard drive partitions, and other hardware.

Step 5. Init

- Looks at the /etc/inittab file to decide the Linux run level.
- Following are the available run levels
    - 0 – halt
    - 1 – Single user mode
    - 2 – Multiuser, without NFS
    - 3 – Full multiuser mode
    - 4 – unused
    - 5 – X11
    - 6 – reboot

- Init identifies the default initlevel from /etc/inittab and uses that to load all appropriate program.
- Execute ‘grep initdefault /etc/inittab’ on your system to identify the default run level
- If you want to get into trouble, you can set the default run level to 0 or 6. Since you know what 0 and 6 means, probably you might not do that.
- Typically you would set the default run level to either 3 or 5.

Step 6. Runlevel programs

- When the Linux system is booting up, you might see various services getting started. For example, it might say “starting sendmail …. OK”. Those are the runlevel programs, executed from the run level directory as defined by your run level.
- Depending on your default init level setting, the system will execute the programs from one of the following directories.
    - Run level 0 – /etc/rc.d/rc0.d/
    - Run level 1 – /etc/rc.d/rc1.d/
    - Run level 2 – /etc/rc.d/rc2.d/
    - Run level 3 – /etc/rc.d/rc3.d/
    - Run level 4 – /etc/rc.d/rc4.d/
    - Run level 5 – /etc/rc.d/rc5.d/
    - Run level 6 – /etc/rc.d/rc6.d/
- Please note that there are also symbolic links available for these directory under /etc directly. So, /etc/rc0.d is linked to /etc/rc.d/rc0.d.
- Under the /etc/rc.d/rc*.d/ directories, you would see programs that start with S and K.
- Programs starts with S are used during startup. S for startup.
- Programs starts with K are used during shutdown. K for kill.
- There are numbers right next to S and K in the program names. Those are the sequence number in which the programs should be started or killed.
- For example, S12syslog is to start the syslog deamon, which has the sequence number of 12. S80sendmail is to start the sendmail daemon, which has the sequence number of 80. So, syslog program will be started before sendmail.

Login Process

1. Users enter their username and password
2. The operating system confirms your name and password.
3. A "shell" is created for you based on your entry in the "/etc/passwd" file
4. You are "placed" in your "home"directory.
5. Start-up information is read from the file named "/etc/profile". This file is known as the system login file. When every user logs in, they read the information in this file.
6. Additional information is read from the file named ".profile" that is located in your "home" directory. This file is known as your personal login file.

Outcome:

Learned the various stages of Linux boot process.

Ex. No. 2	UNDERSTANDING THE LINUX FILE SYSTEM

Objective:

To study various Linux file system and file system structure.

Linux File System

Linux File System or any file system generally is a layer which is under the operating system that handles the positioning of your data on the storage, without it; the system cannot knows which file starts from where and ends where.

Linux offers many file systems types like:

- **Ext**: an old one and no longer used due to limitations.
- **Ext2**: first Linux file system that allows 2 terabytes of data allowed.
- **Ext3**: came from Ext2, but with upgrades and backward compatibility.
- **Ext4**: faster and allow large files with significant speed. (Best Linux File System). It is a very good option for SSD disks and you notice when you try to install any Linux distro that this one is the default file system that Linux suggests.
- **JFS**: old file system made by IBM. It works very well with small and big files, butit failed and files corrupted after long time use, reports say.
- **XFS**: old file system and works slowly with small files.
- **Btrfs:** made by Oracle. It is not stable as Ext in some distros, but you can say that it is a replacement for it if you have to. It has a good performance.
- **Nfs:** The network file system used to access disks located on remote computers.
- **Ntfs:** replaces Microsoft Window's FAT file systems (VFAT, FAT32). It has reliability, performance, and space- utilization.
- **Umsdos: It** is an extended DOS file system used by Linux.

File System Structure

A file system is a logical collection of files on a partition or disk. A partition is a container for information and can span an entire hard drive if desired. UNIX uses a hierarchical file system structure, much like an upside-down tree, with root (/) at the base of the file system and all other directories spreading from there.

The following table provides a short overview of the most important higher-level directories you find on a Linux system

Directory	Contents
/	Root directory—the starting point of the directory tree.
/bin	Essential binary files. Binary Executable files
/boot	Static files of the boot loader.
/dev	Files needed to access host-specific devices.
/etc	Host-specific system configuration files.
/lib	Essential shared libraries and kernel modules.
/media	Mount points for removable media.
/mnt	Mount point for temporarily mounting a file system.
/opt	Add-on application software packages.
/root	Home directory for the super user root.
/sbin	Essential system binaries.
/srv	Data for services provided by the system.
/proc	Contains all processes marked as a file by process number or other information that is dynamic to the system
/tmp	Temporary files.
/usr	Secondary hierarchy with read-only data.
/var	Variable data such as log files
/kernal	Contains kernel files

EDITORS AND FILTERS

VI EDITOR

- vi fname → to open the file fname

- There are two types of mode in vi editor
  Escape mode – used to give commands – to switch to escape mode, press <Esc> key
  Command mode – used to edit the text – to switch to command mode, press any one the following inserting text command

a) Inserting Text

i	→ insert text before the cursor
a	→ append text after the cursor
I	→ insert text at the beginning of the line
A	→ append text to the end of the line
r	→ replace character under the cursor with the next character typed
R	→ Overwrite characters until the end of the line
o	→ (small o) open new line after the current line to type text
O	→ (capital O) open new line before the current line to type text

b) Cursor movements

h	→ left
j	→ down
k	→up
l	→ right

(The arrow keys usually work also)

^**F**	→forward one screen
^**B**	→back one screen
^**D**	→down half screen
^**U**	→up half screen

(^ indicates control key; case does not matter)

0 → (zero) beginning of line
$ → end of line

c) Deleting text

Note : (n) indicates a number, and is optional

dd → deletes current line
(n) **dd** → deletes (n) line(s)   ex. 5dd → deletes 5 lines
(n)**dw** → deletes (n) word(s)
D → deletes from cursor to end of line

x	→ deletes current character
(n)**x**	→ deletes (n) character(s)
X	→ deletes previous character

d) Saving files

:w	→ to save & resume editing (write & resume)
:wq	→ to save & exit (write & quit)
:q!	→ quit without save

e) Cut, Copy and Paste

yy → copies current line
(n) yy → copies (n) lines from the current line. ex. 4yy copies 4 lines.
p → paste deleted or yanked (copied) lines after the cursor

FILTERS

1. cut

- Used to cut characters or fileds from a file/input

Syntax : **cut** **-c**chars filename
-ffieldnos filename

- By default, tab is the filed separator(delimiter). If the fileds of the files are separated by any other character, we need to specify explicitly by **–d** option

cut **-d**delimitchar **-f**fileds filname

2. paste

- Paste files vertically. That is nth line of first file and nth line of second file are pasted as the nth line of result

Syntax : **paste** file1 file2

-ddchar	option is used to paste the lines using the delimiting character *dchar*
-s	option is used paste the lines of the file in a single line

3. tr

- Used to translate characters from standard input

Syntax : **tr** char1 char2 < filename

It translates char1 into char2 in file filename

- Octal representation characters can also be used

Octal value	**Character**
'\7'	Bell
'\10'	Backspace
'\11'	Tab

'\12'	Newline
'\33'	Escape

Ex. tr : '\11' < f1 *translates all* **:** *into tab of f ile f1*

-s Option translate multiple occurrences of a character by single character.
-d Option is to delete a character

4. grep

- Used to search one or more files for a particular pattern.

Syntax : **grep** pattern filename(s)

- Lines that contain the *pattern* in the file(s) get displayed
- pattern can be any regular expressions
- More than one files can be searched for a pattern

-v option displays the lines that do not contain the *pattern*
-l list only name of the files that contain the *pattern*
-n displays also the line number along with the lines that matches the *pattern*

5. sort

- Used to sort the file in order

Syntax : **sort** filename

- Sorts the data as text by default
- Sorts by the first filed by default

-r	option sorts the file in descending order
-u	eliminates duplicate lines
-o filename	writes sorted data into the file *fname*
-tdchar	sorts the file in which fileds are separated by *dchar*
-n	sorts the data as number
+1n	skip first filed and sort the file by second filed numerically

6. Uniq

- Displays unique lines of a sorted file

Syntax : **uniq** filename

-d option displays only the duplicate lines
-c displays unique lines with no. of occurrences.

7. cmp

- Used to compare two files

Syntax : **cmp** f1 f2
compare two files f1 & f2 and prints the line of first difference .

8. diff

- Used to differentiate two files

  Syntax : **diff** f1 f2
  compare two files f1 & f2 and prints all the lines that are differed between f1 & f2.

9. comm

- Used to compare two sorted files
  Syntax : **comm** file1 file2

  Three columns of output will be displayed.
  First column displays the lines that are unique to file1
  Second column displays the lines that are unique to file2
  Third column displays the lines that are appears in both the files

-1	option suppress first column
-2	option suppress second column
-3	option suppress third column
-12	option display only third column
-13	option display only second column
-23	option display only first column

Outcome:

Learned and used various Linux file system, file system structure, VI editor and various filter commands in Linux and executed.

Ex. No. 3	UNDERSTANDING THE VARIOUS PHASES OF COMPILATION OF C PROGRAM

Objective:

To practice how to create and execute C Programs in Linux.

Compilation of C Program

Step 1: Open the terminal and edit your program using vi editor/gedit editor and save with extension ".c"

```
Ex.     vi test.c
        (or)
            gedit text.c
```

Step 2: Compile your program using gcc compiler

```
Ex.     gcc test.c → Output file will be "a.out"
        (or)
        gcc –o test text.c → Output file will be "test"
```

Step 3: Correct the errors if any and run the program

```
Ex.     ./a.out
        or
        ./test
```

Optional Step: In order to avoid. / prefix each time a program is to be executed, insert the following as the last line in the file **.profile**

```
export PATH=.:$PATH
```

This Step needs only to be done once.

Debug C Programs using gdb debugger

Step 1 : Compile C program with debugging option –g

```
Ex.     gcc –g test.c
```

Step 2 : Launch gdb. You will get gdb prompt

```
Ex.     gdb a.out
```

Step 3 : Step break points inside C program

Ex. (gdb) b 10

Break points set up at line number 10. We can have any number of break points

Step 4: Run the program inside gdb

Ex. (gdb) r

Step 5: Print variable to get the intermediate values of the variables at break point

Ex. (gdb) p i → Prints the value of the variable ‘i'

Step 6: Continue or stepping over the program using the following gdb commands

c → continue till the next break
n → Execute the next line. Treats function as single statement
s → Similar to ‘n’ but executes function statements line by line
l → List the program statements

Step 7: Quit the debugger

(gdb) q

Outcome:

Learned and practiced how to create and execute C Programs in Linux.

Ex. No. 4	**LINUX COMMANDS**

Objective:

To practice various basic Linux commands.

a) Basics Commands

1.	echo SRM	→ to display the string SRM
2.	clear	→ to clear the screen
3.	date	→ to display the current date and time
4.	cal 2003	→ to display the calendar for the year 2003
	cal 6 2003	→ to display the calendar for the June-2003
5.	passwd	→ to change password
6.	free –m	→ to view the size of RAM in MB
	free –g	→ to view the size of RAM in GB
7.	df –h	→ to view the disk space available and used.
8.	uptime	→ to view the system up time
9.	bc	→ to open a basic calculator
10.	ps	→ to view the current terminal running processes
11.	history	→ to get the history of all the past commands
12.	whoami	→ to know which user i am

b) Working with Files

1.	ls	→ list files in the present working directory
	ls –l	→ list files with detailed information (long list)
	ls –a	→ list all files including the hidden files
	ls –r root	→ list the directory recursively
	ls –lh	→ list the current location content in human redable format
	ls –lt	→ to list the files based on modification time
	ls –li	→ to view the inode number of files and directories
	lscpu	→ to view the system specifications
2.	cat > f1	→ to create a file (Press ^d to finish typing)
3.	cat f1	→ display the content of the file f1
4.	wc f1	→ list no. of characters, words & lines of a file f1
	wc –c f1	→ list only no. of characters of file f1
	wc –w f1	→ list only no. of words of file f1

wc –l f1 → list only no. of lines of file f1

5. cp f1 f2 → copy file f1 into f2

6. mv f1 f2 → rename file f1 as f2

7. rm f1 → remove the file f1

8. head –5 f1 → list first 5 lines of the file f1
   tail –5 f1 → list last 5 lines of the file f1

c) Working with Directories

1. mkdir elias → to create the directory elias
2. cd elias → to change the directory as elias
3. rmdir elias → to remove the directory elias

4. pwd → to display the path of the present working directory

5. cd → to go to the home directory
   cd .. → to go to the parent directory
   cd - → to go to the previous working directory
   cd / → to go to the root directory

d) File name substitution

1. ls f? → list files start with 'f' and followed by any one character
2. ls *.c → list files with extension 'c'
3. ls [gpy]et → list files whose first letter is any one of the character g, p or y and followed by the word et
4. ls [a-d,l-m]ring → list files whose first letter is any one of the character from a to d and l to m and followed by the word ring.

e) I/O Redirection

1. Input redirection

   wc –l < ex1 → To find the number of lines of the file 'ex1'

2. Output redirection

   who > f2 → the output of 'who' will be redirected to file f2

3. cat >> f1 → to append more into the file f1

f) Piping

Syntax : Command1 | command2

Output of the command1 is transferred to the command2 as input. Finally output of the command2 will be displayed on the monitor.

ex. cat f1 | more → list the contents of file f1 screen by screen

head –6 f1 |tail –2 → prints the 5th & 6th lines of the file f1.

g) Environment variables

1. echo $HOME → display the path of the home directory
2. echo $PS1 → display the prompt string $
3. echo $PS2 → display the second prompt string (> symbol by default)
4. echo $LOGNAME → login name
5. echo $PATH → list of pathname where the OS searches for an executable file

h) File Permission

-- chmod command is used to change the access permission of a file.

Method-1

Syntax : chmod [ugo] [+/-] [rwxa] filename

u : user, g : group, o : others
+ : Add permission - : Remove the
permission r : read, w : write, x : execute, a :
all permissions

ex. chmod ug+rw f1
adding 'read & write' permissions of file f1 to both user and group members.

Method-2

Syntax : chmod octnum file1

The 3 digit octal number represents as follows

- first digit -- file permissions for the user
- second digit -- file permissions for the group
- third digit -- file permissions for others

Each digit is specified as the sum of following
4 – read permission, 2 – write permission, 1 – execute

permission ex. chmod 754 f1

it change the file permission for the file as follows

- read, write & execute permissions for the user ie; 4+2+1 = 7
- read, & execute permissions for the group members ie; 4+0+1 = 5
- only read permission for others ie; 4+0+0 = 4

QUESTIONS FOR PRACTICE:

Q1. Write a command to cut 5 to 8 characters of the file *f1*.

$

Q2. Write a command to display user-id of all the users in your system.
$

Q3. Write a command to paste all the lines of the file *f1* into single line
$

Q4. Write a command to cut the first field of file *f1* and second field of file *f2* and paste intothe file *f3*.
$

Q5. Write a command to change all small case letters to capitals of file *f2*.
$

Q6. Write a command to replace all *tab* character in the file *f2* by **:**
$

Q7. Write a command to check whether the user j*udith* is available in your system or not.(use grep)
$

Q8. Write a command to display the lines of the file *f1* starts with SRM.
$

Q9. Write a command to display the name of the files in the directory */etc/init.d* thatcontains the pattern *grep*.
$

Q10. Write a command to display the names of nologin users. (Hint: the command *nologin* is specified in the last filed of the file /etc/passwd for nologin users)
$

Q11. Write a command to sort the file /etc/passwd in descending order
$

Q12. Write a command to sort the file /etc/passwd by user-id numerically. (Hint : user-id isin 3rd field)
$

Q13. Write a command to sort the file *f2* and write the output into the file *f22*. Alsoeliminate duplicate lines.
$

Q14. Write a command to display the unique lines of the sorted file *f21*. Also display thenumber of occurrences of each line.
$

Q15. Write a command to display the lines that are common to the files *f1* and *f2*.

```
$
```

Outcome:

Various basic Linux commands are learned and executed.

Ex. No. 5	SYSTEM ADMIN COMMANDS

Objective:

To study about various system admin commands used to manage software installation, users,file system and Network configuration.

INSTALLING SOFTWARE

Procedure:

- Open the Ubuntu software Center.
- To install any package, open the terminal (Ctrl + Alt + T) and type sudo apt-get install <package name>.
- For instance, to get Chrome type sudo apt-get install chrome-browser.
- Likewise user can work package update, remove and reinstall the package using the following commands.

To update the package repositories

```
sudo apt-get update
```

To update installed software

```
sudo apt-get upgrade
```

To install a package/software

```
sudo apt-get install  <package-name>
```

To remove a package from the system

```
sudo apt-get remove <package-name>
```

To reinstall a package

```
sudo apt-get install  <package-name> --reinstall
```

To completely remove a software and it's dependent packages run the apt-get purge

```
sudo apt-get purge <package-name>
```

To remove all Debian (.deb) files those are no longer installed
---files in /var/cache/apt/archives

```
sudo apt-get autoclean
```

To empty whole cache files – to reduce the space consumption

```
sudo apt-get clean
```

To remove old dependent files and footprints installed by previous applications

```
sudo apt-get automove
```

To configure installed package

```
sudo dpkg –configure –a
```

To download but not install package

```
sudoapt-get download <package-name>
```

MANAGING USERS

- Managing users is a critical aspect of server management.
- In Ubuntu, the root user is disabled for safety.
- Root access can be completed by using the sudo command by a user who is in the “admin” group.
- When you create a user during installation, that user is added automatically to the admin group.

To add a user:

```
sudo adduser username
```

To disable a user:

```
sudo passwd -l username
```

To enable a user:

```
sudo passwd -u username
```

To delete a user:

```
sudo userdel –r username
```

To create a group:

```
sudo addgroup groupname
```

To delete a group:

```
sudo delgroup groupname
```

To create a user with group:

```
sudo adduser username groupname
```

To see the password expiry value for a user,

```
sudo chage -l username
```

To make changes:

```
sudo chage username
```

GUI TOOL FOR USER MANAGEMENT

GUI Tool allow the admin to run the commands in terminal to manage users and groups.

To install a GUI add-on

```
sudo apt install gnome-system-tools
```

Once done, type

```
users-admin
```

MANAGING THE FILE SYSTEM

A filesystem is a permanent storage for containing data. Any non-volatile storage device like hard disk, usb etc has a filesystem in place, on top of which data is stored. While installing Linux, you may opt for either EXT4 or EXT3 file system.

Ext3 : A journaling filesystem: logs changes in a journal to increase reliability in case of power failure or system crash.

EXT4: It is an advanced file syste. This file system supports 64-bit storage limits, columns up to 1 exabytes and you may store files up to 16 terabytes

Disk Partitions can be viewed by the command `sudo fdisk -l`

File system information are available in the file `/etc/fstab`

MANGING THE NETWORK CONFIGURATION

Most networking is configured by editing two files:

- `/etc/network/interfaces`
  - Ethernet, TCP/IP, bridging
- `/etc/resolv.conf`
  - DNS

Other networking files:

- `/etc/hosts`
- `/etc/dhcp3/dhcpd.conf`

To test any host's connectivity

```
ping <ip-address>
```

To start/stop/restart/reload networking services

```
sudo /etc/init.d/mnetworking <function>
```

Note : <function> can be any one of `stop` or `start` or `reload` or `restart`

To list of all active network interface cards, including wireless and the loopback interface

```
sudo ifconfig
```

To display host Fully Qualified Domain Name

```
sudo hostname
```

To display arp table (ip to mac resolution)

```
sudo arp –a
```

To remove entry from arp table

```
sudo arp –d <user name>
```

To display or change network card settings, use ethtool

```
sudo ethtool eth0
```

To displays extensive status information when queried with the service iptables status command

```
sudo service iptables status
```

To start/stop services

```
sudo service iptables start/stop
```

INSTALLING INTERNET SERVICES

Installing Apache server

```
sudo apt-get install apache2
```

Configuration file for Apache server

```
apache2.conf
```

Restart apache services after any configuration changes made

```
sudo /etc/init.d/mnetworking restart
```

Similarly all services can be installed, configured and restarted

MANAGING BACKGROUND JOBS

To display jobs running in background

```
sudo jobs
```

To check the process id of background processes

```
sudo jobs –p
```

To bring a background job to the foreground

```
sudo fg
```

To start the Jobs suspended in background

```
sudo bg
```

QUESTIONS FOR PRACTICE:

Q1. Update the package repositories

Q2. Install the package "simplescreenrecorder"

Q3. Remove the package "simplescreenrecorder"

Q4. Create a user 'elias'. Login to the newly created user and exit.

Q5. Disable the user 'elias', try to login and enable again.

Q6. Create a group 'cse' and add the user 'elias' in that group

Q7. List the account expiry information of the user 'elias'

Q8. Change the 'Number of days warning before password expires' as 5 for the user 'elias'

Q9. Delete the user 'elias' and then delete the group 'cse'

Q10. List the partitions available in your system

Q11. What are the file systems used in your system

Q12. Stop the networking service and then start the service

Q13. Check the connectivity of the host with IP address 127.0.0.1

Q14. Find the IP address of the localhost

Q15. Find the IP address of the DNS Server (name server)

Q16. Install mysql server

Q17. Restart mysql server

Q18. Check the configuration file for mysql server

Q19. Log on as root into mysql server
Q20. Create a new database for mysql server

Outcome:

Learned various Linux based System admin commands successfully.

Ex. No. 6	SIMPLE TASK AUTOMATION

Objective:

To study about simple Task Automation using Linux Crontab utility.

Crontab

Linux Cron utility is an effective way to schedule a routine background job at a specific time and/or day on an on-going basis. User can use this to schedule activities, either as one-time events or as recurring tasks.

Scheduling of Tasks (For Ubuntu)

Step 1 : Open terminal and type the command `crontab -e`

Step 2 : Choose the editor. Better to select `nano` editor

Step 3 : Edit the file based on the syntax given above

Step 4 : Save and Exit the file

Step 5 : Start cron daemon using the following command

```
systemctl start cron
```

Linux Crontab Format

MIN HOUR DOM MON DOW CMD

Table: Crontab Fields and Allowed Ranges (Linux Crontab Syntax)

Field	Description	Allowed Value
MIN	Minute field	0 to 59
HOUR	Hour field	0 to 23
DOM	Day of Month	1-31
MON	Month field	1-12
DOW	Day Of Week	0-6
CMD	Command	Any command to be executed

Create a new crontab file, or edit an existing file
# crontab -e [username]

where *username* specifies the name of the user's account for which you want to create or edit a crontab file.

Verify your crontab file changes
# crontab -l [username]

Install crontab

crontab -a filename

Edit the crontab

# crontab -e

Display crontab

crontab -l

Display the last edit the crontab file

crontab -v

Remove crontab

crontab -r

Following are the syntax for cron

minute(s) hour(s) day(s) month(s) weekday(s) command(s) "Argument1" "Argument2"

1 * 3 4 5 /path/to/command arg1 arg2

If you don't have parameter put star(*)
Commands:

1) **-l** - List or manage the task with crontab command
2) **-e** - edit crontab entry.
3) **-u** - To list scheduled jobs of a particular user called **tecmint** using.
4) **-r -** parameter will remove complete scheduled jobs without confirmation from crontab.
5) **-i** - prompt you confirmation from user before deleting user's crontab.

Allowed special character (*, -, /, ?, #)

1. **Asterik(*)** – Match all values in the field or any possible value.

2. **Hyphen(-)** – To define range.
3. **Slash (/)** – 1st field /10 meaning every ten minute or increment of range.
4. **Comma (,)** – To separate items.

System Wide Cron Schedule

System administrator can use predefine cron directory as shown below.

1. /etc/cron.d
2. /etc/cron.daily
3. /etc/cron.hourly
4. /etc/cron.monthly
5. /etc/cron.weekly

To Schedule a Job for Specific Time

The below jobs delete empty files and directory from **/tmp** at **12:30** am daily.
User need to mention user name to perform crontab command.
In below example **root** user is performing cron job.

```
# crontab –e

30 0 * * *  root  find /tmp -type f -empty –delete
```

Special Strings for Common Schedule

Strings	**Meanings**
@reboot	Command will run when the system reboot.
@daily	Once per day or may use @midnight.
@weekly	Once per week.
@yearly	Once per year.user can use @annually keyword also.

Multiple Commands with Double ampersand (&&)

To run the command1 and command2 daily

```
# crontab -e

@daily <command1> && <command2>
```

Disable Email Notification.
By default cron send mail to user account executing cronjob. If user want to disable using **>/dev/null 2>&1** option at the end of the file will redirect all the output of the cron results under **/dev/null**.

```
[root@tecmint ~]# crontab -e

* * * * * >/dev/null 2>&1
```

Scheduling a Job for a Specific Time

The basic usage of cron is to execute a job in a specific time as shown below. This will execute the full backup shell script (full-backup) on **10th June 08:30 AM**.

The below time field uses 24 hours format. So, for 8 AM use 8, and for 8 PM use 20.

```
30 08 10 06 * /home/username/full-backup
```

- **30** – 30th Minute
- **08** – 08 AM
- **10** – 10th Day
- **06** – 6th Month (June)
- ***** – Every day of the week

Schedule a Job for More Than One Instance (e.g. Twice a Day)

The following script takes a incremental backup twice a day every day. This example executes the specified incremental backup shell script (incremental-backup) at 11:00 and 16:00 on every day. The comma separated value in a field specifies that the command needs to be executed in all the mentioned time.

```
00 11,16 * * * /home/username/bin/incremental-backup
```

- **00** – 0th Minute (Top of the hour)
- **11,16** – 11 AM and 4 PM
- ***** – Every day
- ***** – Every month
- ***** – Every day of the week

Schedule a Job for Specific Range of Time (e.g. Only on Weekdays)

- To schedule the job for every hour with in a specific range of time then use the following.

Cron Job everyday during working hours

This example checks the status of the database everyday (including weekends) during the working hours 9 a.m – 6 p.m

```
00 09-18 * * * /home/username/bin/check-db-status
```

- **00** – 0th Minute (Top of the hour)
- **09-18** – 9 am, 10 am,11 am, 12 am, 1 pm, 2 pm, 3 pm, 4 pm, 5 pm, 6 pm
- ***** – Every day
- ***** – Every month
- ***** – Every day of the week

Schedule a Job for Every Minute Using Cron.

Ideally user may not have a requirement to schedule a job every minute. But understanding this example will help user understand the other examples mentioned below in this article.

```
* * * * * CMD
```

The * means all the possible unit — i.e every minute of every hour throughout the year. More than using this * directly, user will find it very useful in the following cases.

- When user specify */5 in minute field means every 5 minutes.
- When user specify 0-10/2 in minute field mean every 2 minutes in the first 10 minute.
- Thus the above convention can be used for all the other 4 fields.

Schedule a Background Cron Job For Every 10 Minutes.

Use the following, to check the disk space every 10 minutes.

```
*/10 * * * * /home/username/check-disk-space
```

It executes the specified command check-disk-space every 10 minutes throughout the year.

There are special cases in which instead of the above 5 fields you can use @ followed by a keyword — such as reboot, midnight, yearly, hourly.

Table: Cron special keywords and its meaning

Keyword	Equivalent
@yearly	0 0 1 1 *
@daily	0 0 * * *
@hourly	0 * * * *
@reboot	Run at startup.

Schedule a Job for First Minute of Every Year using @yearly

User can specify a job to be executed on the first minute of every year, then user can use the **@yearly** cron keyword as shown below.

This will execute the system annual maintenance using annual-maintenance shell script at 00:00 on Jan 1st for every year.

```
@yearly /home/username/red-hat/bin/annual-maintenance
```

Schedule a Cron Job Beginning of Every Month using @monthly

Executes the command monthly once using **@monthly** cron keyword.

This will execute the shell script tape-backup at 00:00 on 1st of every month.

```
@monthly /home/username/suse/bin/tape-backup
```

Schedule a Background Job Every Day using @daily

Using the @daily cron keyword, this will do a daily log file cleanup using cleanup-logs shell scriptat 00:00 on every day.

```
@daily /home/username/arch-linux/bin/cleanup-logs "day started"
```

To Execute a Linux Command After Every Reboot using @reboot

Using the **@reboot** cron keyword, this will execute the specified command once after the machine got booted every time.

```
@reboot CMD
```

To Disable/Redirect the Crontab Mail Output using MAIL keyword

By default crontab sends the job output to the user who scheduled the job. To redirect the output to a specific user, add or update the MAIL variable in the crontab as shown below.

```
username@dev-db$ crontab -l
MAIL="username"
@yearly /home/username/annual-maintenance
*/10 * * * * /home/username/check-disk-space
```

[Note: Crontab of the current logged in user with MAIL variable]

To stop the crontab output to be emailed, add or update the MAIL variable in the crontab as shown below.

```
MAIL=""
```

Specify PATH Variable in the Crontab

To set absolute path of the Linux command or the shell-script :

Instead of specifying /home/username/tape-backup, user can specify tape-backup, then add the path /home/username to the PATH variable in the crontab as shown below.

```
username@dev-db$ crontab -l
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/home/username
@yearly annual-maintenance
*/10 * * * * check-disk-space
```

[Note: Crontab of the current logged in user with PATH variable]

To Install Crontab from a Cron File

Instead of directly editing the crontab file, user can also add all the entries to a cron-file first.

Once userhave all thoese entries in the file, user can upload or install them to the cron as shown below.

```
username@dev-db$ crontab -l
no crontab for username
$ cat cron-file.txt
@yearly /home/username/annual-maintenance
*/10 * * * * /home/username/check-disk-space
username@dev-db$ crontab cron-file.txt
username@dev-db$ crontab -l
@yearly /home/username/annual-maintenance
*/10 * * * * /home/username/check-disk-space
```

To View Crontab Entries:

View Current Logged-In User's Crontab entries
To view crontab entries type
crontab -l

```
Username@dev-db$ crontab -l
@yearly /home/username/annual-maintenance

*/10 * * * * /home/username/check-disk-space
```

[Note: This displays crontab of the current logged in user]

To View Root Crontab entries

Login as root user (su – root) and do crontab -l as shown below.

```
root@dev-db# crontab -l

no crontab for root
```

To View Other Linux User's Crontabs entries:
To view crontab entries of other Linux users,

login to root and use **-u {username} -l**

```
root@dev-db# crontab -u username -l
@monthly /home/username/monthly-backup
00 09-18 * * * /home/username/check-db-status
```

To Edit Crontab Entries:

Edit Current Logged-In User's Crontab entries
To edit a crontab entries,

use crontab –e

By default this will edit the current logged-in users crontab.

```
username@dev-db$ crontab -e

@yearly /home/username/centos/bin/annual-maintenance

*/10 * * * * /home/username/debian/bin/check-disk-space
"/tmp/crontab.XXXXyjWkHw" 2L, 83C
```

[Note: This will open the crontab file in Vim editor for editing.

Please note cron created a temporary /tmp/crontab.XX...]
When user save the above temporary file with :wq, it will save the crontab and display the following message indicating the crontab is successfully modified.

QUESTIONS FOR PRACTICE:
Q1. Schedule a task to display the following message on the monitor for every 2 minutes.
Q2. Schedule a task to take backup of your important file (say file f1) for every 30 minutes
Q3. Schedule a task to take backup of login information everyday 9:30am

Outcome:

Students Learned about all the simple task automation commands.

Ex. No. 7	SHELL PROGRAMS

How to run a Shell Script

- Edit and save your program using editor
- Add execute permission by *chmod* command
- Run your program using the name of your program

```
./program-name
```

Important Hints

- No space before and after the assignment operator Ex. `sum=0`
- *Single quote* ignores all special characters. Dollar sign, Back quote and Back slash are not ignored inside *Double quote*. *Back quote* is used as command substitution. *Back slash* is used to remove the special meaning of a character.
- Arithmetic expression can be written as follows : `i=$((i+1)` or `i=$(expr $i + 1)`
- Command line arguments are referred inside the programme as `$1, $2`, ..and so on
- `$*` represents all arguments, `$#` specifies the number of arguments
- `read` statement is used to get input from input device. Ex. `read a b`

Syntax for if statement

```
if [ condition ]
then
      ...
elif [ condition ]
then
      ...
else
      ...
fi
```

Syntax for case structure

```
case value in
pat1) ...
               statement;;
pat2)  ...
               Statement;;
*)     ...
               Statement;;
esac
```

Syntax for for-loop

```
for var in list-of-values
do
          ...
          ...
done
```

Syntax for While loop

```
while commandt
do
         ...
         ...
done
```

Syntax for printf statement

```
printf  "string and format"  arg1  arg2 … …
```

- Break and continue statements functions similar to C programming
- Relational operators are –lt, -le, -gt, -ge, -eq,-ne
- Ex. (i>= 10) is written as [$i -ge 10]
- Logical operators (and, or, not) are -o, -a, !
- Ex. (a>b) && (a>c) is written as [$a –gt $b –a $a –gt $c]
- Two strings can be compared using = operator

Q1. Given the following values

num=10, x=*, y=\`date\` a="Hello, 'he said'"

Execute and write the output of the following commands

Command	**Output**
echo num	
echo $num	
echo $x	
echo '$x'	
echo "$x"	
echo $y	
echo $(date)	
echo $a	
echo \$num	
echo \$$num	

Q2. Find the output of the following shell scripts

```
$ vi  ex51
    echo Enter value for n
    read n
    sum=0
    i=1
    while [ $i -le $n ]
    do
          sum=$((sum+i))
          i=$((i+2))
    done
    echo Sum is $sum
```

Output :

Q3. Write a program to check whether the file has execute permission or not. If not, add the permission.

```
$ vi ex52
```

Q4. Write a shell script to print a greeting as specified below.
If hour is greater than or equal to 0 (midnight) and less than or equal to 11 (up to 11:59:59), "Good morning" is displayed.
If hour is greater than or equal to 12 (noon) and less than or equal to 17 (up to 5:59:59 p.m.), "Good afternoon" is displayed.
If neither of the preceding two conditions is satisfied, "Good evening" is displayed.

```
$ vi  ex53
  hour=$(date | cut -c12-13)
  if [ "$hour" -ge 0 -a "$hour" -le 11 ]
  then                                     ← complete the program
```

Q5. Write a shell script to list only the name of sub directories in the present working directory

```
$ vi ex54
```

Q6. Write a program to check all the files in the present working directory for a pattern (passed through command line) and display the name of the file followed by a message stating that the pattern is available or not available.

```
$ vi ex55
```

Outcome:

Thus the basics of shell programming have been learnt successfully and output is verified.

Ex. No. 8	PROCESS CREATION	Date :

Syntax for process creation
int fork();

Returns 0 in child process and child process ID in parent process.

Other Related Functions

int getpid()	→ returns the current process ID
int getppid()	→ returns the parent process ID
wait()	→ makes a process wait for other process to complete

Virtual fork
vfork() function is similar to fork but both processes shares the same address space.

Q1. Find the output of the following program

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

int main()
{
  int a=5,b=10,pid;

  printf("Before fork a=%d b=%d \n",a,b);
  pid=fork();

  if(pid==0)
  {
    a=a+1;
    b=b+1;
    printf("In child a=%d b=%d \n",a,b);
  }
  else
  {
    sleep(1);
    a=a-1;
    b=b-1;
    printf("In Parent a=%d b=%d \n",a,b);
  }
  return 0;
}
```

Output :-

Q2. Rewrite the program in Q1 using vfork() and write the output

Q3. Calculate the number of times the text “SRMIST” is printed.

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

    int main()
    {
        fork();
        fork();
        fork();
        printf(“SRMIST\n”);
        return 0;
  }
```

Output :

Q4. Complete the following program as described below :
The child process calculates the sum of odd numbers and the parent process calculate the sum of even numbers up to the number ‘n’. Ensure the Parent process waits for the child process to finish.

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

int main()
{
  int pid,n,oddsum=0,evensum=0;

  printf("Enter the value of n : ",a);
  scanf(“%d”,&n);
  pid=fork();
  // Complete the program









  return 0;
}
```

Sample Output :

```
Enter the value of n           10
Sum of odd numbers             25
Sum of even numbers            30
```

Q5. How many child processes are created for the following code?
Hint : Check with small values of ‘n’.

```
for (i=0; i<n; i++)
      fork();
```

Output :

Q6. Write a program to print the Child process ID and Parent process ID in both Child and Parent processes

```
    #include <stdio.h>
    #include<unistd.h>
    int main()
    {









return 0;
}
```

Sample Output:

```
In Child Process
Parent Process ID       :     18
Child Process ID        :     20

In Parent Process
Parent Process ID       :     18
Child Process ID        :     20
```

Q7. How many child processes are created for the following code?

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

    int main()
    {
        fork();
        fork()&&fork()||fork();fork();
        printf("Yes ");
        return 0;

}
```

Output :

Outcome:

Learned creation of process successfully .

Ex. No. 9 & 10	OVERLAY CONCEPTS

Objective:

To understand the overlay concepts and practice how to overlay the current process to new process in Linux using C.

Overlay

Overlay is the concept which enables the user to run another new process from the currently running process address space.

System call used :

Exec() System Call

The exec() system call replaces (overwrites) the current process with the new process image. The PID of the new process remains the same however code, data, heap and stack of the process are replaced by the new program. There are 6 system calls in the family of exec(). All of these functions mentioned below are layered on top of execve(), and they differ from one another and from execve() only in the way in which the program name, argument list, and environment of the new program are specified

Syntax:

int execl(const char* path, const char* arg, ...)

int execlp(const char* file, const char* arg, ...)

int execle(const char* path, const char* arg, ..., char* const envpl))

int execv(const char* path, const char* argv[])

int execvp(const char* file, const char* argv[])

int execvpe(const char* file, const char* argv[], char *const envpl])

- The names of the first five of above functions are of the form execXY.
- X is either 1 or v depending upon whether arguments are given in the list format (argo, argl, ..., NULL) or arguments are passed in an array (vector).

- Y is either absent or is either a p or an e. In case Y is p, the PATH environment variable is used to search for the program. If Y is e, then the environment passed in envp array is used.
- In case of execvpe, X is v and Y is e. The execvpe function is a GNU extension. It is named so as to differentiate it from the execve system call.

Simple overlay concept

Procedure:

- Stepl: Create two different c programs. Name it as example.c file and hello.c file
- Step2: Make example .c is the current running process,
- Step3: call the function execv() which takes the hello.c as an argument.
- Step4: Print process id of both the processes(hello.c and example.c processes).
- Step5: trace the system control by having simple print statement in both programs.

Expected Output:

```
$ process id of example.c=4733

We are in hello.c

Process id of hello.c=4733
```

COMBINING FORK() AND EXEC() SYSTEM CALL

Procedure:

- Stepl: Create two different c programs. Name it as example.c file and hello.c file
- Step2: Make example .c is the current running process and use fork() system call to create child process,
- Step3: call the function execv() in child process which takes the hello.c as an argument.
- Step4: Print Process id of both parent, child and overlay processes (hello.c and example.c

processes).

- Step5: trace the system control by having simple print statement in both programs.

Expected Output:

```
$ process id of example.c=4790
The control is in parent process
The control is in child process
Process id of child = 4791
Calling hello.c from child
We are in hello.c
Process id of hello.c=4791
```

PRACTICE QUESTIONS:

1. Execute the Following Program and write the output
$vi ex1.c.

```
#include <stdio.h>
#include <unistd.h>
 int main()
printf("Transfer to execlp function \n");
execlp("head", "head",""-2","fl",NULL); // Assume fl is any text file
printf("This line will not execute \n");
return 0;
```

Output :

Why second printf statement is not executing?

2. Rewrite question 1 with execl() function. Pass the 3rd and 4th argument of the function

execl() through command line arguments.

$vi ex2.c

Input : /a.out -3 fi

Output:

Outcome:

Learned and implemented the overlay concept in Linux using C.

Ex. No: 11	INTER PROCESS COMMUNICATION- Pipe()

Objective:

Inter process communication between the processes using pipe concept.

Inter process communication (IPC):

A process can be of two types:

- Independent process.
- Co-operating process.

An independent process is not affected by the execution of other processes while a co operating process can be affected by other executing processes.

Inter process communication (IPC) is a mechanism which allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other through both:

- Shared Memory
- Message passing

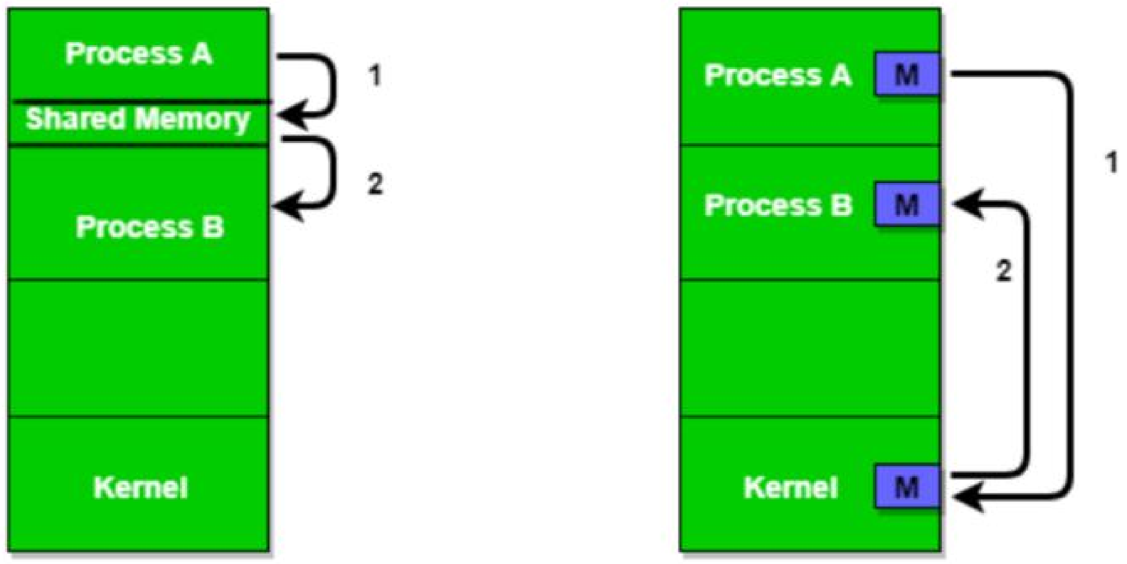


Figure 1 - Shared Memory and Message Passing

Inter process communication (IPC) is used for exchanging data between multiple threads in one or more processes or programs. The Processes may be running on single or multiple computers connected by a network. The full form of IPC is Inter-process communication.

It is a set of programming interface which allow a programmer to coordinate activities among various program processes which can run concurrently in an operating system. This allows a specific program to handle many user requests at the same time.

Since every single user request may result in multiple processes running in the operating system, the process may require to communicate with each other. Each IPC protocol approach has its own advantage and limitation, so it is not unusual for a single program to use all of the IPC methods.

Important methods for inter process communication:

Pipes

Pipe is widely used for communication between two related processes. This is a half-duplex method, so the first process communicates with the second process. However, in order to achieve a full-duplex, another pipe is needed.

Message Passing:

It is a mechanism for a process to communicate and synchronize. Using message passing, the process communicates with each other without resorting to shared variables.

IPC mechanism provides two operations:

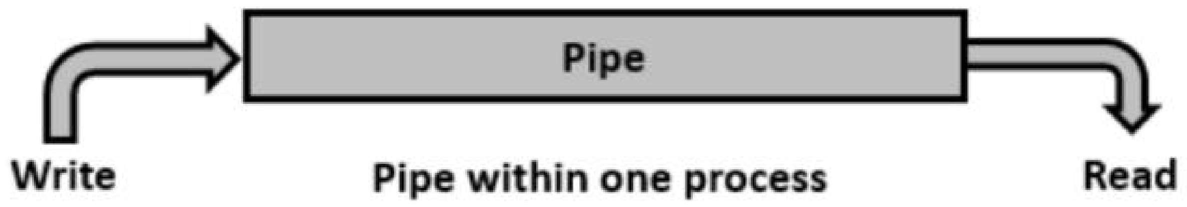
- Send (message)- message size fixed or variable
- Received (message)

Message Queues:

A message queue is a linked list of messages stored within the kernel. It is identified by a message queue identifier. This method offers communication between single or multiple processes with full-duplex capacity.

PIPES

8MPipe is a communication medium between two or more related or interrelated processes. It can be either within one process or a communication between the child and the parent processes. Communication can also be multi-level such as communication between the parent, the child and the grand-child, etc. Communication is achieved by one process writing into the pipe and other reading from the pipe. To achieve the pipe system call, create two files, one to write into the file and another to read from the file.



```
#include<unistd.h>

int pipe(int pipedes[2]);
```

This system call would create a pipe for one-way communication i.e., it creates two descriptors, first one is connected to read from the pipe and other one is connected to write into the pipe.

Descriptor pipedes[0] is for reading and pipedes[1] is for writing. Whatever is written into pipedes[1] can be read from pipedes[0].

This call would return zero on success and -1 in case of failure. To know the cause of failure, check with errno variable or perror() function.

```
#include <sys/types.h> #include <sys/stat.h>

#include <fontl.h>

int open(const char *pathname, int flags);

 int open(const char *pathname, int flags, mode_t mode);
```

Even though the basic operations for file are read and write, it is essential to open the file before performing the operations and closing the file after completion of the required operations. Usually, by default, 3 descriptors opened for every process, which are used for input (standard input - stdin), output (standard output - stdout) and error (standard error stderr) having file descriptors 0, 1 and 2 respectively.

Uni directional pipe:

Algorithm

- **Step 1** - Create a pipe using pipe() system call.
- **Step 2** – Send a message to the one end of the pipe.

- **Step 3** – Retrieve the message from the other end of the pipe and write it to the standard output.

Expected Output:

```
hello, world #1

hello, world #2

 hello, world #3
```

Implementing command line pipe using exec() family of function:

Follow the steps to transfer the output of a process to pipe:

(i) Close the standard output descriptor
(ii) Use the following system calls, to take duplicate of output file descriptor of the pipe int dup(int fd); int dup2(int oldfd, int newfd);
(iii) Close the input file descriptor of the pipe
(iv) Now execute the process.

Follow the steps to get the input from the pipe for a process:

(i) Close the standard input descriptor
(ii) Take the duplicate of input file descriptor of the pipe using dup() system call
(iii) Close the input file descriptor of the pipe
(iv) Now execute the process

Named pipe

Named pipe (also known as FIFO) is one of the inter process communication tool. The system for FIFO is as follows

int mkfifo(const char *pathname, mode_t mode);

mkfifo() makes a FIFO special file with name pathname. Here mode specifies the FIFO's permissions. The permission can be like : O_CREAT|0644 Open FIFO in read-mode (O_RDONLY) to read and write-mode (O_WRONLY) to write.

Write and read two messages using pipe.

Algorithm

Step 1 - Create a pipe.

Step 2 – Send a message to the pipe.

Step 3 – Retrieve the message from the pipe and write it to the standard output.

Step 4 - Send another message to the pipe.

Step 5 - Retrieve the message from the pipe and write it to the standard output.

Note – Retrieving messages can also be done after sending all messages.

Expected Output:

```
Writing to pipe - Message 1 is Hi
Reading from pipe - Message 1 is Hi
Writing to pipe - Message 2 is Hello
Reading from pipe - Message 2 is Hello
```

2. Program to write and read two messages through the pipe using the parent and the child processes.

Algorithm

Step 1 - Create a pipe.

Step 2 - Create a child process.

Step 3 – Parent process writes to the pipe.

Step 4 - Child process retrieves the message from the pipe and writes it to the standard output.

Step 5 – Repeat step 3 and step 4 once again.

Expected Output:

```
Parent Process - Writing to pipe - Message 1 is Hi
Parent Process - Writing to pipe - Message 2 is Hello
Child Process - Reading from pipe - Message 1 is Hi
Child Process - Reading from pipe – Message 2 is Hello
```

Two-way Communication Using Pipes

Following are the steps to achieve two-way communication -

Step 1 - Create two pipes. First one is for the parent to write and child to read, say as pipe l.

Second one is for the child to write and parent to read, say as pipe2.

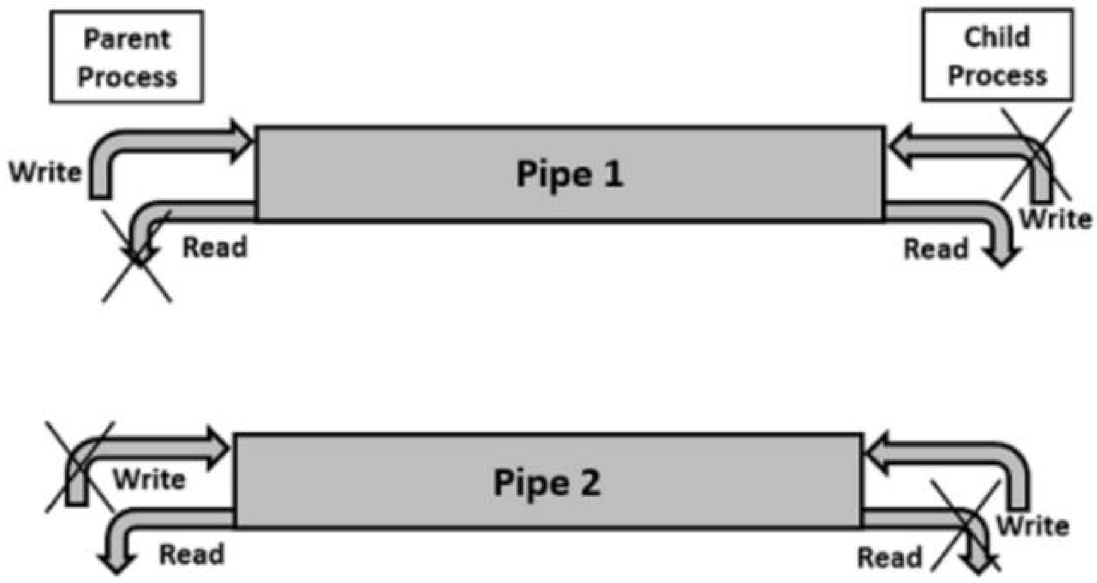
Step 2 - Create a child process.

Step 3 – Close unwanted ends as only one end is needed for each communication.

Step 4 - Close unwanted ends in the parent process, read end of pipel and write end of pipe2.

Step 5 – Close the unwanted ends in the child process, write end of pipel and read end of pipe2.

Step 6 - Perform the communication as required.



3. Achieving two-way communication using pipes

Algorithm

Step 1 - Create pipel for the parent process to write and the child process to read.

Step 2 - Create pipe2 for the child process to write and the parent process to read.

Step 3 – Close the unwanted ends of the pipe from the parent and child side.

Step 4 - Parent process to write a message and child process to read and display on the screen. **Step 5** – Child process to write a message and parent process to read and display on the screen.

Expected Output:

In Parent: Writing to pipe 1 - Message is Hi

In Child: Reading from pipe 1 - Message is Hi

In Child: Writing to pipe 2 – Message is Hello

In Parent: Reading from pipe 2 – Message is Hello

Practice:
Q. Write the output of the following program

#include<stdio.h> #include<unistd.h> #include <sys/wait.h> int main()

int p[2]; char buff[25]; if(fork()==0)

printf("Child : Writing to pipe n"); write(p[1],"Welcome”,8); printf("Child Exiting\n");

else

wait(NULL); printf("Parent : Reading from pipe \n"); read(p[0],buff,8); printf("Pipe content is : %s \n",buff);

return 0;

Output:

Outcome:

Inter process communication using pipes concept learned and implemented.

Ex. No: 12 (a)	INTER PROCESS COMMUNICATION -Shared memory

Objective:

The program to implement Interprocess Communication using shared memory and message queue concept.

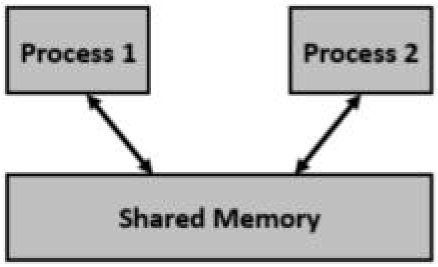
Inter Process Communication:

The popular IPC techniques are Shared Memory and Message Queues.

Shared memory:

We know that to communicate between two or more processes, we use shared memory but before using the shared memory what needs to be done with the system calls,

• Create the shared memory segment or use an already created shared memory segment (shmget())
• Detach the process from the already attached shared memory segment (shmdt())
• Control operations on the shared memory segment (shmctl())



Inter Process Communication through shared memory is a concept where two or more process can access the common memory. And communication is done via this shared memory where changes made by one process can be viewed by another process.

The problem with pipes, fifo and message queue – is that for two process to exchange information. The information has to go through the kernel.

• Server reads from the input file.
• The server writes this data in a message using either a pipe, fifo or message queue.
• The client reads the data from the IPC channel, again requiring the data to be copied from kernel's IPC buffer to the client's buffer.
• Finally the data is copied from the client's buffer.

A total of four copies of data are required (2 read and 2 write). So, shared memory provides a way by letting two or more processes share a memory segment. With Shared Memory the data is only copied twice – from input file into shared memory and from shared memory to the output file.

SYSTEM CALLS USED ARE:

- **ftok**(): is use to generate a unique key.

- **shmget**(): int shmget(key_t, size_tsize, intshmflg); upon successful completion, shmget() returns an identifier for the shared memory segment.

- **shmat()**: Before you can use a shared memory segment, you have to attach yourself to it using shmat(). void *shmat(int shmid , void *shmaddr , int shmflg);

- shmid is shared memory id. shmaddr specifies specific address to use but we should set it to zero and OS will automatically choose the address.

- **shm**dt(): When you're done with the shared memory segment, your program should detach itself from it using shmdt(). int shmdt(void *shmaddr);

- **shmctl()**: when you detach from shared memory, it is not destroyed. So, to destroy shmctl() is used. shmctl(int shmid,IPC_RMID,NULL); The second argument, cmd, is the command to perform the required control operation on the shared memory segment.

Valid values for cmd are -

- **IPC_STAT** - Copies the information of the current values of each member of struct shmid_ds to the passed structure pointed by buf. This command requires read permission to the shared memory segment.
- **IPC_SET** - Sets the user ID, group ID of the owner, permissions, etc. pointed to by structure buf.
- **IPC_RMID** – Marks the segment to be destroyed. The segment is destroyed only after the last process has detached it. **IPC_INFO** - Returns the information about the shared memory limits and parameters in the structure pointed by buf.
- **SHM_INF**O – Returns a shm_info structure containing information about the consumed system resources by the shared memory.

Program:1- Shared memory implementation using readers writers problem.

Writer process:

Algorithm:

- **Step 1** - Create a shared memory using (shmget()) function.
- **Step 2** - attach the current process in to created shared memory be calling shmat() function.
- **Step 3** - Write into shared memory after attaching in to it.
- **Step 4-** After completing write operation detach the process from shared memory area.

Reader process:

Algorithm:

- **Step 1** - Create a shared memory using (shmget()) function.
- **Step 2** - attach the current process in to created shared memory be calling shmat() function.
- **Step 3** - read the data which is already written by the reader process from shared memory after attaching in to it.
- **Step 4-** Print the string and detach the process from shared memory area.

Expected Output:

Writer.c

```
Write Data : Operating System Data Written in memory: Operating System
```

Reader.c

```
Data read from memory: Operating System
```

Outcome:

Interprocess communication using shared memory concept learned and implemented.

Ex. No: 12b	**INTER PROCESS COMMUNICATION -Message Queue**

Objective:

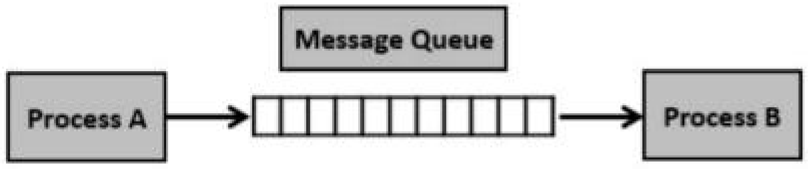
The program to implement Inter process communication using message queue concept.

Inter Process Communication-Message Queue

A message queue is a linked list of messages stored within the kernel and identified by a message queue identifier. A new queue is created or an existing queue opened **by msgget().**

New messages are added to the end of a queue by msgsnd(). Every message has a positive long integer type field, a non-negative length, and the actual data bytes *(*corresponding to the length), all of which are specified to msgsnd() when the message is added to a queue. Messages are fetched from a queue by **msgrcv().** We don't have to fetch the messages in a first-in, first-out order. Instead, we can fetch messages based on their type field.

All processes can exchange information through access to a common system message queue. The sending process places a message (via some (OS) message-passing module) onto a queue which can be read by another process. Each message is given an identification or type so that processes can select the appropriate message. Process must share a common key in order to gain access to the queue in the first place.



System calls used for message queues:

- **ftok():** is use to generate a unique key.
- **msgget():** either returns the message queue identifier for a newly created message queue or returns the identifiers for a queue which exists with the same key value.
- **msgsnd():** Data is placed on to a message queue by calling msgsnd().
- **msgrcv():** messages are retrieved from a queue.
- **msgctl():** It performs various operations on a queue. Generally it is use to destroy message queue.

Program :To perform communication using message queues, following are the steps -

Writer Process:

- **Step 1** - Create a message queue or connect to an already existing message queue (msgget())
- **Step 2** – specify the message type as 1.
- **Step 3-** Write into message queue (msgsnd())
- **Step 4**- terminate the process

Reader Process:

- **Step 1** - Create a message queue or connect to an already existing message queue (msgget())
- **Step 2** – specify the message type as 1.
- **Step 3** – Read from the message queue (msgrev())
- **Step 4** - Perform control operations on the message queue (msgctl())
- **Step 5** – terminate the reader process

Expected Output:

Writer.c

Write Message : Hello

Sent Message : Hello

Reader.c

Received Message is : Hello

Outcome:

Thus the concept of Interprocess Communication using message Queue has been implemented using readers writers problem.