



Lab Manual

Operating System Lab

(CC-217-3L)

Faculty of Computing & Information Technology (FCIT)

University of the Punjab, Lahore.

www.pucit.edu.pk

OPERATING SYSTEM LABORATORY MANUAL



UNIVERSITY OF THE PUNJAB

**FACULTY OF COMPUTING & INFORMATION TECHNOLOGY, LAHORE
DEPARTMENT OF COMPUTER SCIENCE**

Course:	Operating System Lab	Date:
Course Code:	CC-217-3L	Max Marks: 40
Faculty/Instructor's Name & Email:	Dr. Ahmad Hassan Butt (ahmad.hassan@pucit.edu.pk)	

**LAB MANUAL # 1
(SPRING 2023)**

Name: _____ Enroll No: _____

Objective(s) :

To understand basic concepts of Operating System.

Lab Tasks :

Task 1: What is an Operating System?

Task 2: Which OS is being used in the Lab?

Task 3: What are LINUX distributions? Why there are various distributions of LINUX?

Task 4: What is a Virtual Machine? Differentiate between Guest and Host OS.

Lab Grading Sheet :

Task	Max Marks	Obtained Marks	Comments(<i>if any</i>)
1.	10		
2.	10		
3.	10		
4.	10		
Total	40		Signature

Note : Attempt all tasks and get them checked by your Instructor

Lab 01: Operating System Concepts

Objective(s):

To understand basic concepts of Operating System.

Tool(s) used:

Ubuntu

Introduction to Operating System

Operating system is software which performs executive functions. It provides services and controls sharing of information to the users. We have batch processing, Multi-Programming, Multitasking and Multithreading operating systems. Modern operating systems for desktop and laptop environments use operating systems like Windows, LINUX and Solaris.

They have layered architecture, Kernel being the lowest to provide hardware abstraction to upper layers of operating system, utilities and User interface. The windows and Linux operating systems are shown in the figure 1 and 2.

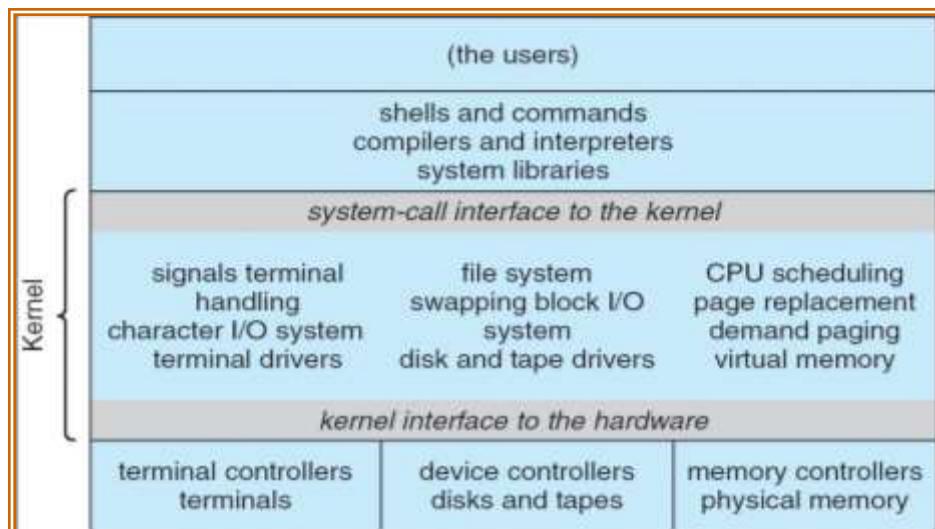


Figure 1: Linux Operating System

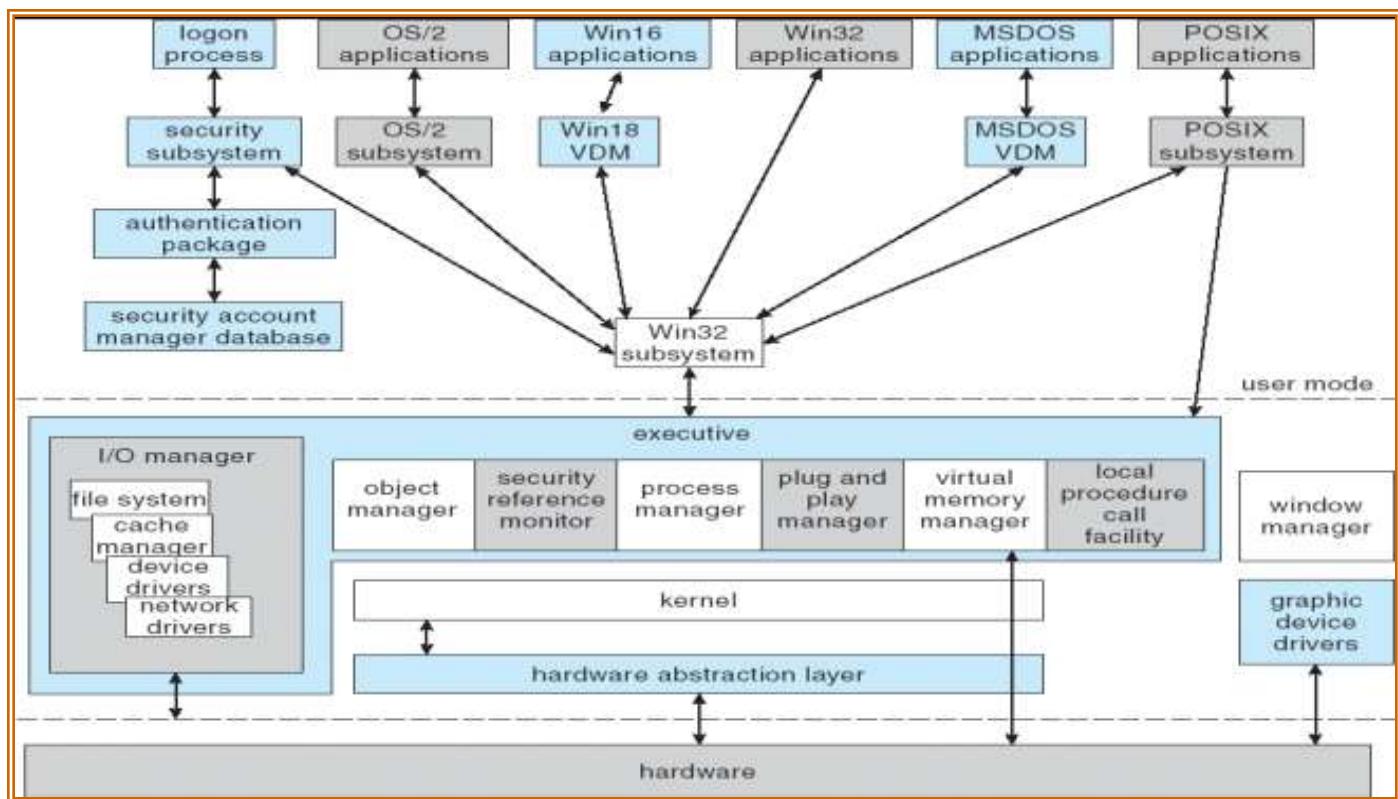


Figure 2: Windows Operating System

Windows and Mac OS are predominantly found on personal computing devices such as desktop and laptop computers. Other operating systems, such as Symbian, are found on small devices such as phones and PDAs, while mainframes and supercomputers found in major academic and corporate labs use specialized operating systems such as AS/400 and the Cray OS. Linux, which began its existence as a server OS and has become useful as a desktop OS, can also be used on all of these devices.

Introduction to LINUX

We will use LINUX operating system for most of our lab work to be able to learn how the operating systems software is developed to implement various algorithms. Linux is a generic term referring to Unix-like computer operating systems based on the Linux kernel. Their development is one of the most prominent examples of free and open source software collaboration; typically all the underlying source code can be used, freely modified, and redistributed by anyone.

The name "Linux" comes from the Linux kernel, originally written in 1991 by Linus Torvalds. The rest of the system usually comprises components such as the Apache HTTP Server, the X Window System,

the K Desktop Environment, and utilities and libraries from the GNU operating system (announced in 1983 by Richard Stallman).

Many quantitative studies of free / open source software focus on topics including market share and reliability, with numerous studies specifically examining Linux. The Linux market is growing rapidly, and the revenue of servers, desktops, and packaged software running Linux was expected to exceed \$35.7 billion by 2008.

LINUX File System

A file system is the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk. The word is also used to refer to a partition or disk that is used to store the files or the type of the file system.

The difference between a disk or partition and the file system it contains is important. A few programs (including, reasonably enough, programs that create file systems) operate directly on the raw sectors of a disk or partition; if there is an existing file system there it will be destroyed or seriously corrupted. Most programs operate on a file system, and therefore won't work on a partition that doesn't contain one (or that contains one of the wrong type). Before a partition or disk can be used as a file system, it needs to be initialized, and the bookkeeping data structures need to be written to the disk. This process is called making a file system. Figure 3 shows the typical LINUX file system. Linux uses a single hierarchical directory structure. Everything starts from the root directory, represented by /, and then expands into sub-directories instead of having so-called 'drives'. The filenames are case sensitive.

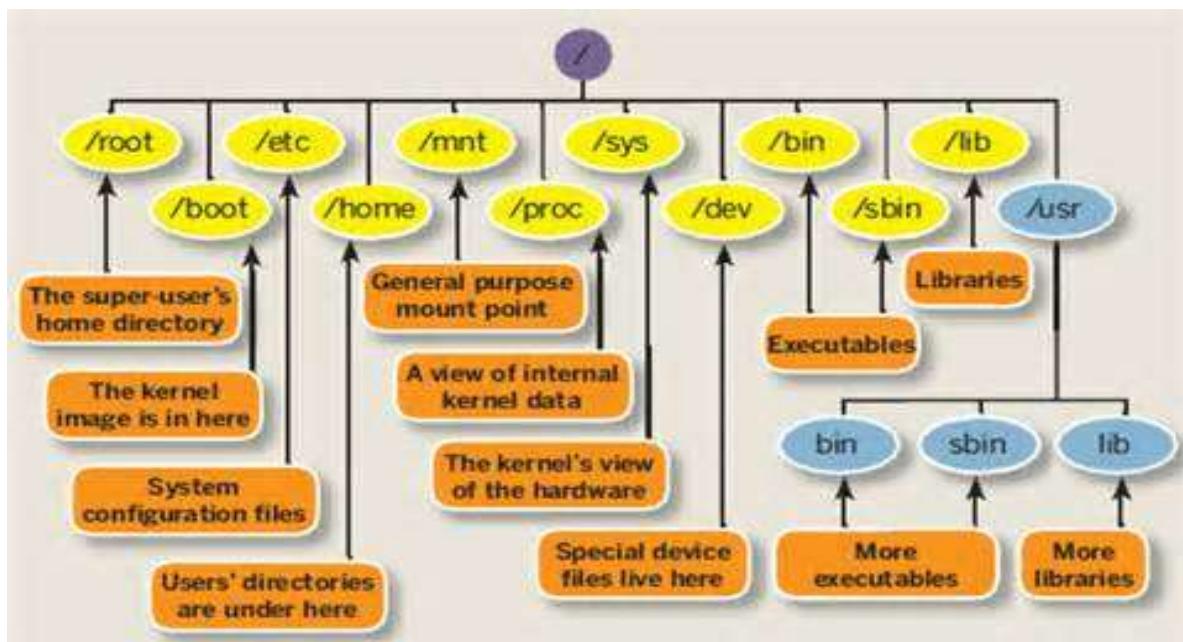


Figure 3: LINUX File Systems

/bin	Directory contains several useful commands that are used by both the system administrator as well as non-privileged users.
/boot	Directory contains the system Map file as well as the Linux kernel. Lilo places the boot sector backups in this directory.
/dev	hda1, hda2 etc, which represent the various partitions on the first master drive of the system. /dev/cdrom and /dev/fd0 represent your CDROM drive and your floppy drive. One important characteristic of the Linux file system is that everything is a file or a directory.
/etc	Directory contains all the configuration files for the system.
/home	Contains user home directories, which can be found under /home/username.
/lib	Contains all the shared libraries that are required by system programs.
/lost+found	Linux should always go through a proper shutdown. Sometimes your system might crash or a power failure might take the machine down. Either way, at the next boot, a lengthy filesystem check using fsck will be done. Fsck will go through the system and try to recover any corrupt files that it finds. The result of this recovery operation will be placed in this directory.
/mnt	This directory usually contains mount points or sub-directories where you mount your floppy and your CD.
/opt	This directory contains all the software and add-on packages that are not part of the default installation.
/proc	This is a special directory on your system.
/root	We talked about user home directories earlier and well this one is the home directory of the user root.
/tmp	This directory contains mostly files that are required temporarily.
/usr	This is one of the most important directories in the system as it contains all the user binaries. /usr/src/linux contains the source code for the Linux kernel.
/var	This directory contains spooling data like mail and also the output from the printer daemon. The above content briefs about Linux and the file system of Linux.

Development of LINUX

The Linux kernel was first made available in 1991 after a student named Linus Torvalds finished developing it as an alternative to the MINIX Operating system. MINIX was an operating system based off of UNIX, which was developed in 1969 and had become a popular OS within the education environment and industry. The Linux kernel was free to use and it was quickly taken up by developers around the world. It wasn't long before there were several distributions of operating systems running with the Linux kernel. One of the especially notable projects was the GNU project, which was started by Richard Stallman in the mid 80's. Richard Stallman wanted to help start a community that was built around the idea of free and open software. Stallman decided the best place to start was with an Operating system, and by 1990 he had completed most of the GNU project with the exception of the kernel. Thus, when the Linux kernel was made available, the GNU suite of software was pieced together with the kernel to form the GNU/Linux operating system. This combined package would go on to be the back bone to many other Linux distributions. By the end of 1994, Linux version 1.0.0 had been released and the world had begun to take notice of the ever growing number of distributions.

LINUX popularity and evolution of distributions

As the Linux kernel and its distributions continued to improve and grow in popularity, larger companies began to offer more and more support for the free OS and the idea of free software in general. From 1994-1997, Linux began to be picked up in mainstream publications such as wired magazine as well as gain notice from tradeshows.

The year 1998 was especially fruitful for Linux, with support beginning to come from the Google search engine in May and software for Linux from companies such as Informix and Oracle in July. It was also announced during that Intel and Netscape had invested money into the Red Hat company (the Red Hat distribution was one of the earlier Linux distributions and would go on to be the back bone to many popular distributions and even end up on board of a submarine). With more and more improvements on the software, and the release of such notable desktop environments as KDE and Gnome in the late 90's and early 2000's, Linux was beginning to really take hold of the computer industry.

Understanding Virtual Machines

A virtual machine is a software computer that, like a physical machine, runs an operating system and applications. A virtual machine uses the physical resources of the physical machine on which it runs, which is called the host system. Virtual machines have virtual devices that provide the same functionality as physical hardware, but with the additional benefits of portability, manageability, and security. A virtual machine has an operating system and virtual resources that you manage in much the same way that you manage a physical computer. For example, you install an operating system in a virtual machine in the same way that you install an operating system on a physical computer. You must have a CD-ROM, DVD, or ISO image that contains the installation files from an operating system vendor.

Preparing to Create a New Virtual Machine

You use the New Virtual Machine wizard to create a new virtual machine in Workstation. The wizard prompts you to make decisions about many aspects of the virtual machine. You should make these decisions before you start the New Virtual Machine wizard.

Selecting a Virtual Machine Configuration

When you start the New Virtual Machine wizard, the wizard prompts you to select a typical or custom configuration.

Typical Configuration

If you select a typical configuration, you must specify or accept defaults for a few basic virtual machine settings.

- How you want to install the guest operating system.
- A name for the virtual machine and a location for the virtual machine files.
- The size of the virtual disk and whether to split the disk into multiple virtual disk files.
- Whether to customize specific hardware settings, including memory allocation, number of virtual processors, and network connection type.

OPERATING SYSTEM LABORATORY MANUAL



UNIVERSITY OF THE PUNJAB

**FACULTY OF COMPUTING & INFORMATION TECHNOLOGY, LAHORE
DEPARTMENT OF COMPUTER SCIENCE**

Course:	Operating System Lab	Date:
Course Code:	CC-217-3L	Max Marks: 40
Faculty/Instructor's Name & Email:	Dr. Ahmad Hassan Butt (ahmad.hassan@pucit.edu.pk)	

**LAB MANUAL # 2
(SPRING 2023)**

Name: _____ Enroll No: _____

Objective(s) :

Installation of VMWare and Ubuntu.

Lab Tasks :

Task 1 + 2 : Installation of VMWare

Task 3 + 4 : Installation of Ubuntu

Lab Grading Sheet :

Task	Max Marks	Obtained Marks	Comments(<i>if any</i>)
1.	10		
2.	10		
3.	10		
4.	10		
Total	40		Signature

Note : Attempt all tasks and get them checked by your Instructor

Lab 02: Installation of VMWare and Ubuntu

Objective(s):

Installation of VMWare and Ubuntu.

Tool(s) used:

Ubuntu

UNIX OPERATING SYSTEM

An operating system is the program that controls all the other parts of a computer system - both the hardware and the software. Most importantly, it allows you to make use of the facilities provided by the system. Examples of operating systems are Windows XP, Windows NT, UNIX, Linux, ..etc.

UNIX is an operating system which was first developed in the 1960s, and has been under constant development ever since. By operating system, we mean the suite of programs which make the computer work. It is a stable, multi-user, multi-tasking system for servers, desktops and laptops.

UNIX systems also have a graphical user interface (GUI) similar to Microsoft Windows which provides an easy-to-use environment. However, knowledge of UNIX is required for operations which aren't covered by a graphical program, or for when there is no windows interface available, for example, in a telnet session.

Different Versions of Unix

There are many different versions of UNIX, although they share common similarities. The most popular varieties of UNIX are:

Sun Solaris,
GNU/Linux, and
MacOS X.

UBUNTU Operating System

Ubuntu is a Debian-based Linux operating system for personal computers, tablets and smartphones, where Ubuntu Touch edition is used; and also runs network servers, usually with the Ubuntu Server edition, either on physical or virtual servers (such as on mainframes) or with containers, that is with enterprise-class features; runs on the most popular architectures, including server-class ARM-based.

Installation of UBUNTU

The installation system is easy to use even if you lack previous knowledge of Linux or computer networks. If you select default options, Ubuntu provides a complete desktop operating system, including productivity applications, Internet utilities, and desktop tools. Ubuntu Workstation is a reliable, user-friendly, and powerful operating system for your laptop or desktop computer. It supports a wide range of developers, from hobbyists and students to professionals in corporate environments.

Installing VMWare Tools

The following are general steps used to start the VMWare Tools installation in most VMWare products. Certain guest operating systems may require different steps, but these steps work for most operating systems. Links to more detailed steps for different operating systems are included in this article. Make sure to review the VMWare documentation for the product you are using.

VMware develops virtualization Software. Virtualization software creates an abstraction layer over computer hardware that allows the hardware elements of a single computer processors, memory, storage, and more to be divided into multiple virtual computers, commonly called virtual machines (VMs). Each virtual machine runs its own operating system (OS) and behaves like an independent computer, even though it is running on a portion of the actual underlying computer hardware. A VM is a software-based representation of a physical computer. An operating system (OS) running in a VM is called a guest OS.

Method 01: Setting up Ubuntu with Vmware

1. Installing VMware Workstation from given below link. There are two options for downloading one is Windows and other for Linux.

<https://customerconnect.vmware.com/en/downloads/details?downloadGroup=WKST-PLAYER-1624&productId=1039&rPId=91446>

The screenshot shows the VMware Customer Connect website. At the top, there's a navigation bar with links for Products and Accounts, Knowledge, Communities, Support, and Learning. On the far right are icons for search, user profile, Register, and Login. Below the navigation, a breadcrumb trail shows 'Home / VMware Workstation Player'. The main title is 'Download Product'. A dropdown menu for 'Select Version' shows '16.2.4'. Under 'Documentation', there's a link to 'Release Notes'. The 'Release Date' is listed as '2022-07-21'. The 'Type' is 'Product Binaries'. To the right, a sidebar titled 'Product Resources' includes links for View My Download History, Product Info, Documentation, Knowledge Base, Community, Self-Help Support, Support Policies, and Workstation Player Upgrade. At the bottom of the page, there are tabs for Product Downloads, Drivers & Tools, Open Source, Custom ISOs, and OEM Add-ons. The 'Product Downloads' tab is selected. It lists two download options: 'VMware Workstation 16.2.4 Player for Windows 64-bit Operating Systems' and 'VMware Workstation 16.2.4 Player for Linux 64-bit'. Each entry includes file size (584.34 MB), file type (exe), and a 'Read More' link. To the right of each entry is a blue 'DOWNLOAD NOW' button.

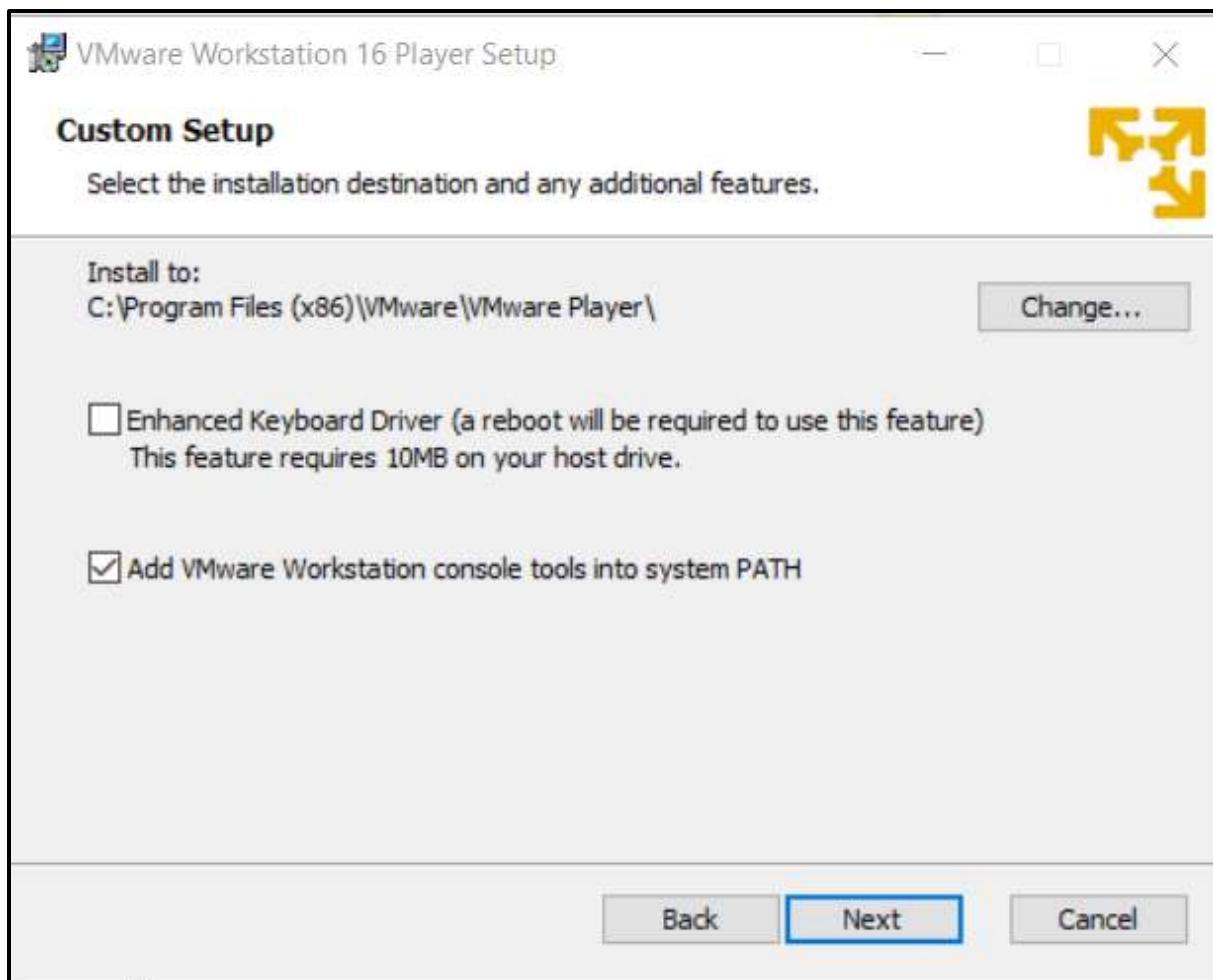
- Run the VMware downloaded File and Click on Next to the Installation wizard.



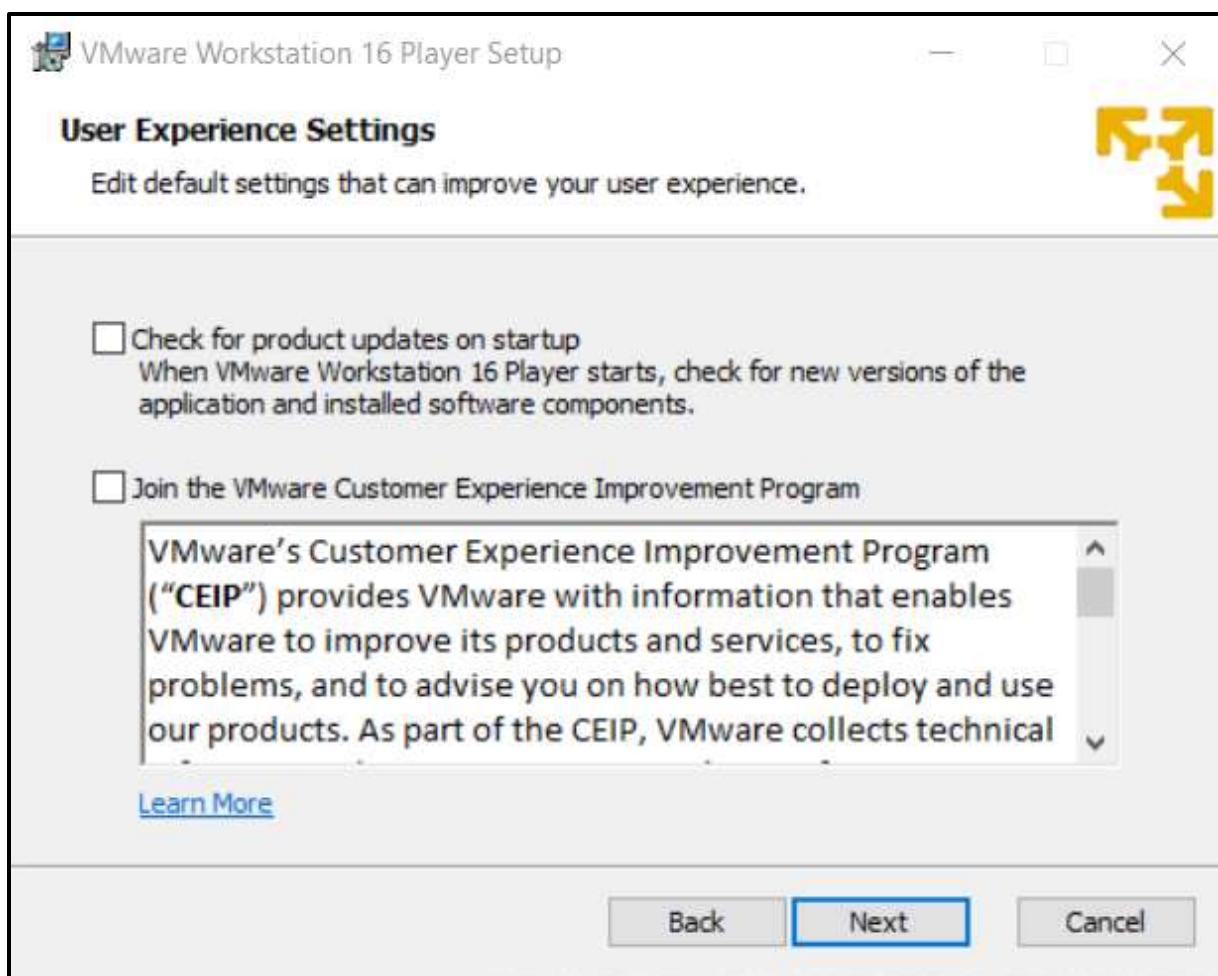
3. Accept user license agreement and click on next.



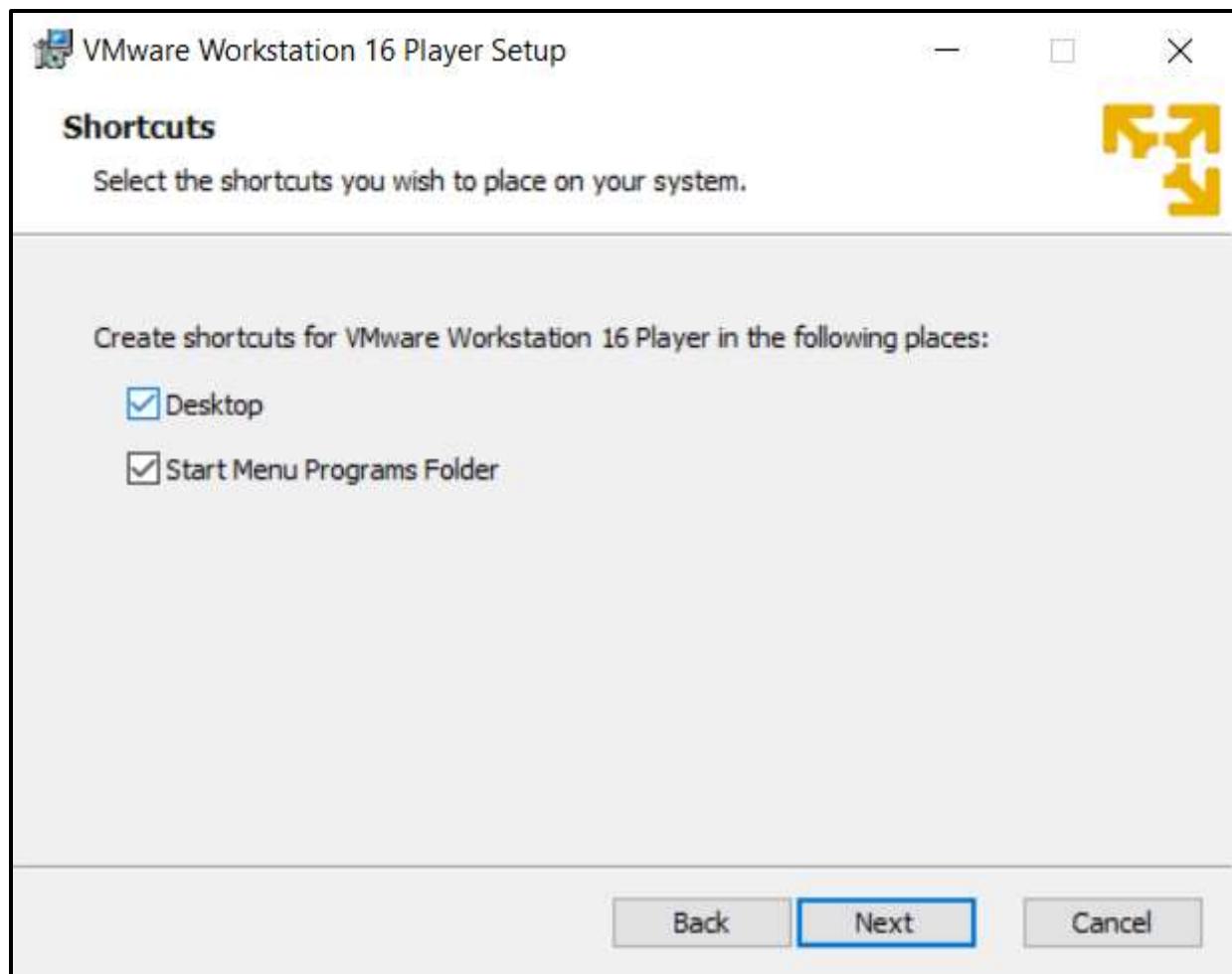
4. Specify the Installation directory. You can also enable Enhance keyboard driver here. Click Next to continue.



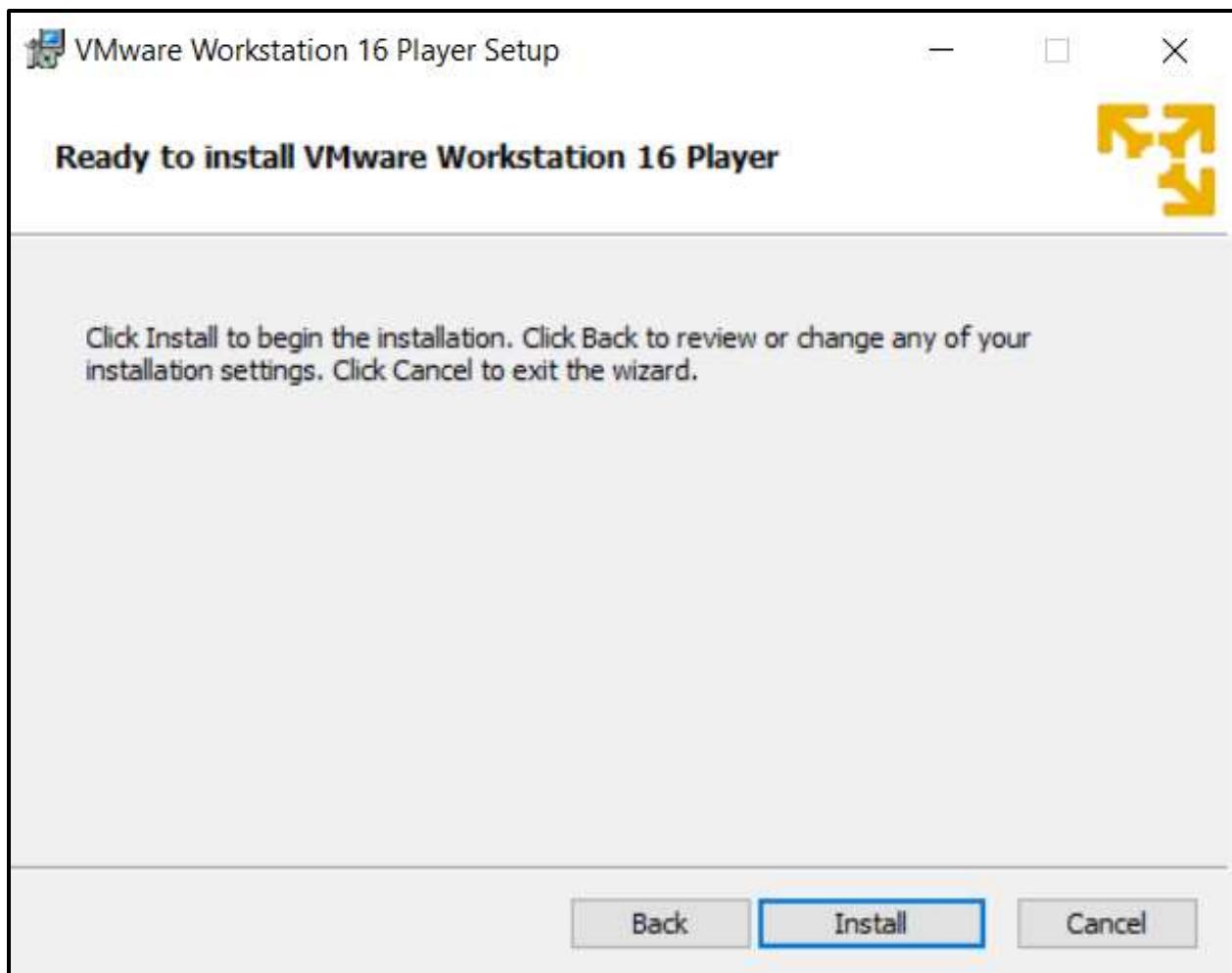
5. You can enable product startup and join the VMware Customer experience Improvement program here. Click Next to Continue.



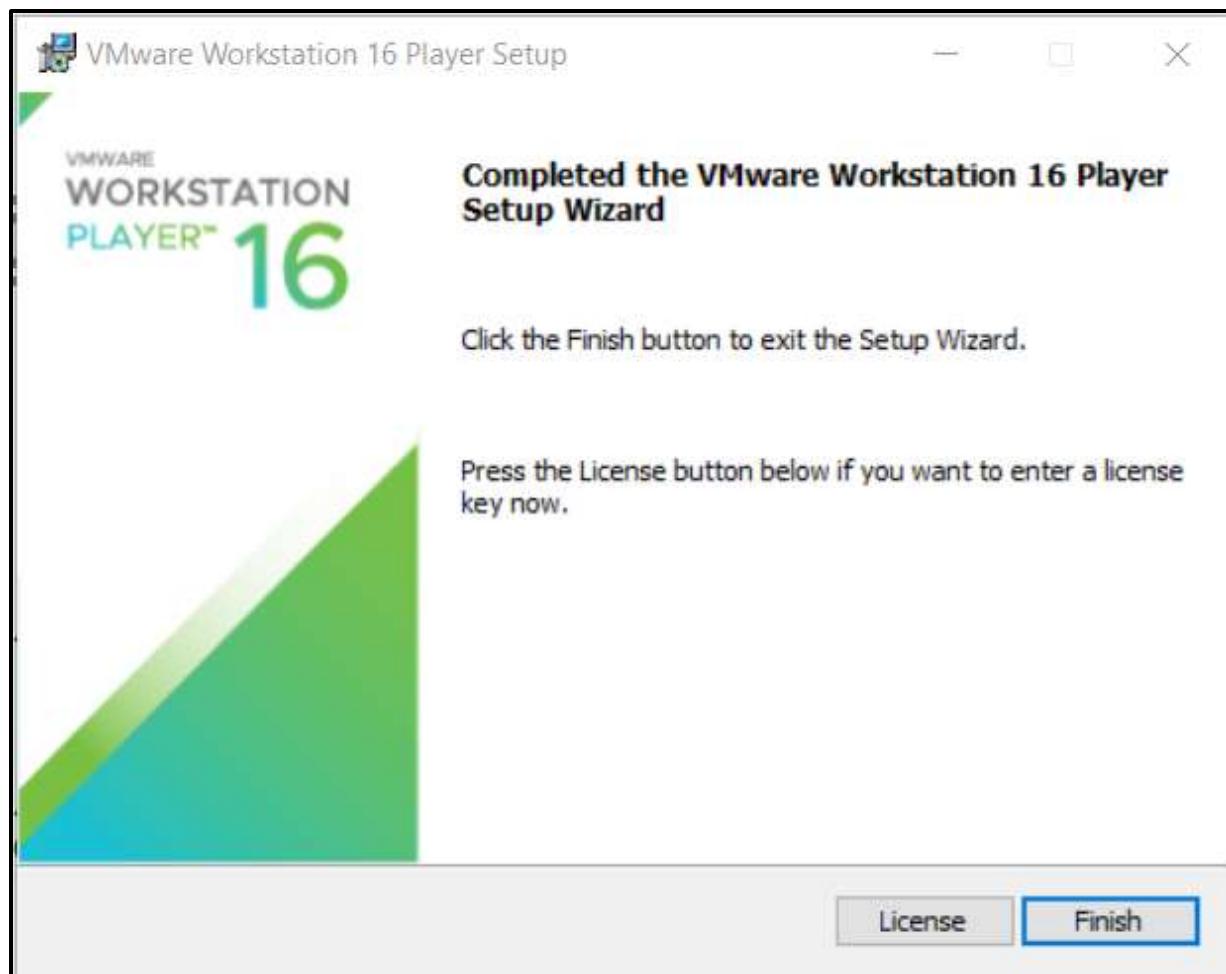
6. Select the shortcuts you want to create for easy access to VMware Workstation. Click Next to Continue.



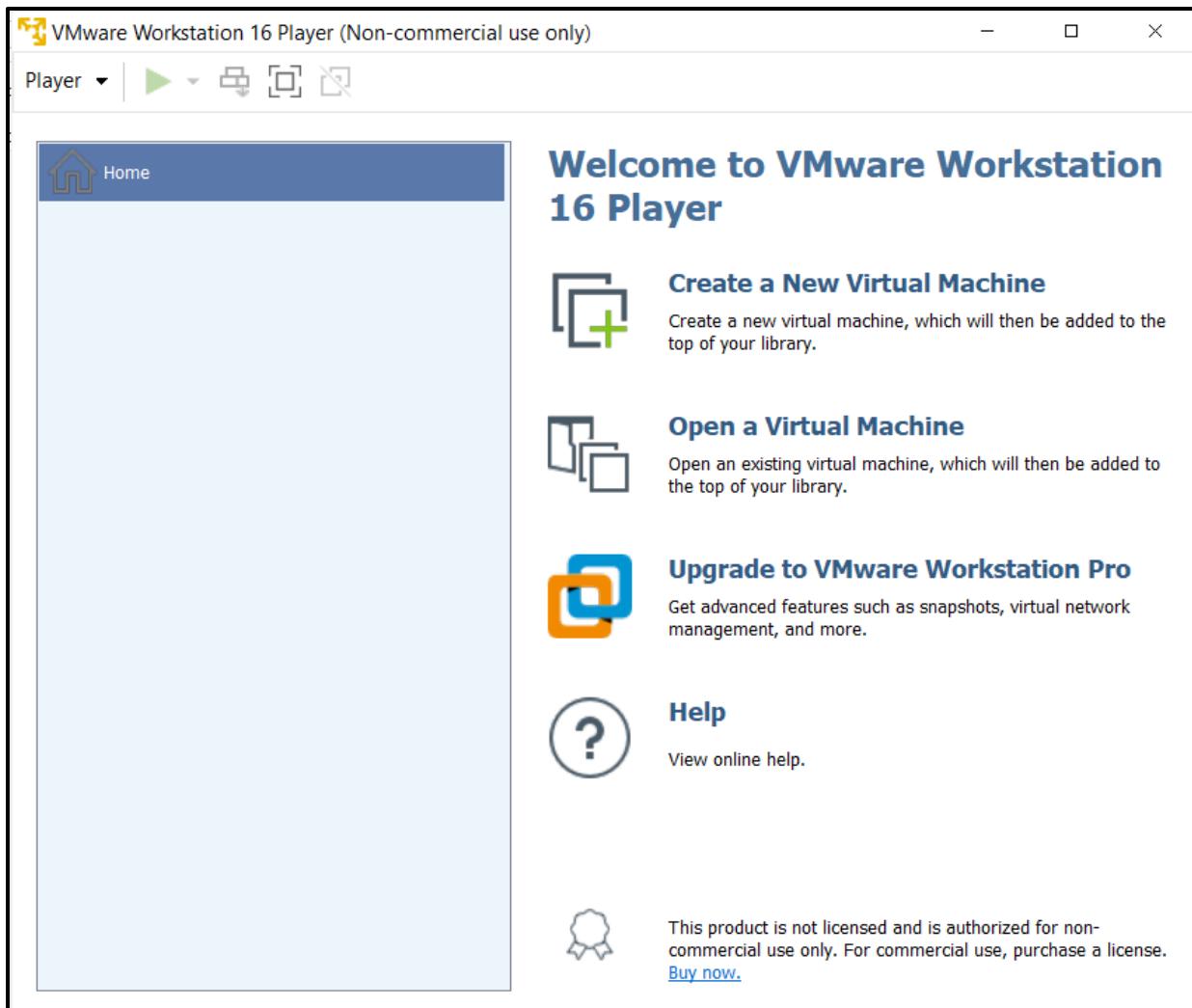
7. Click Install button to start the installation.



8. Installation will take just few seconds to complete. Click finish.

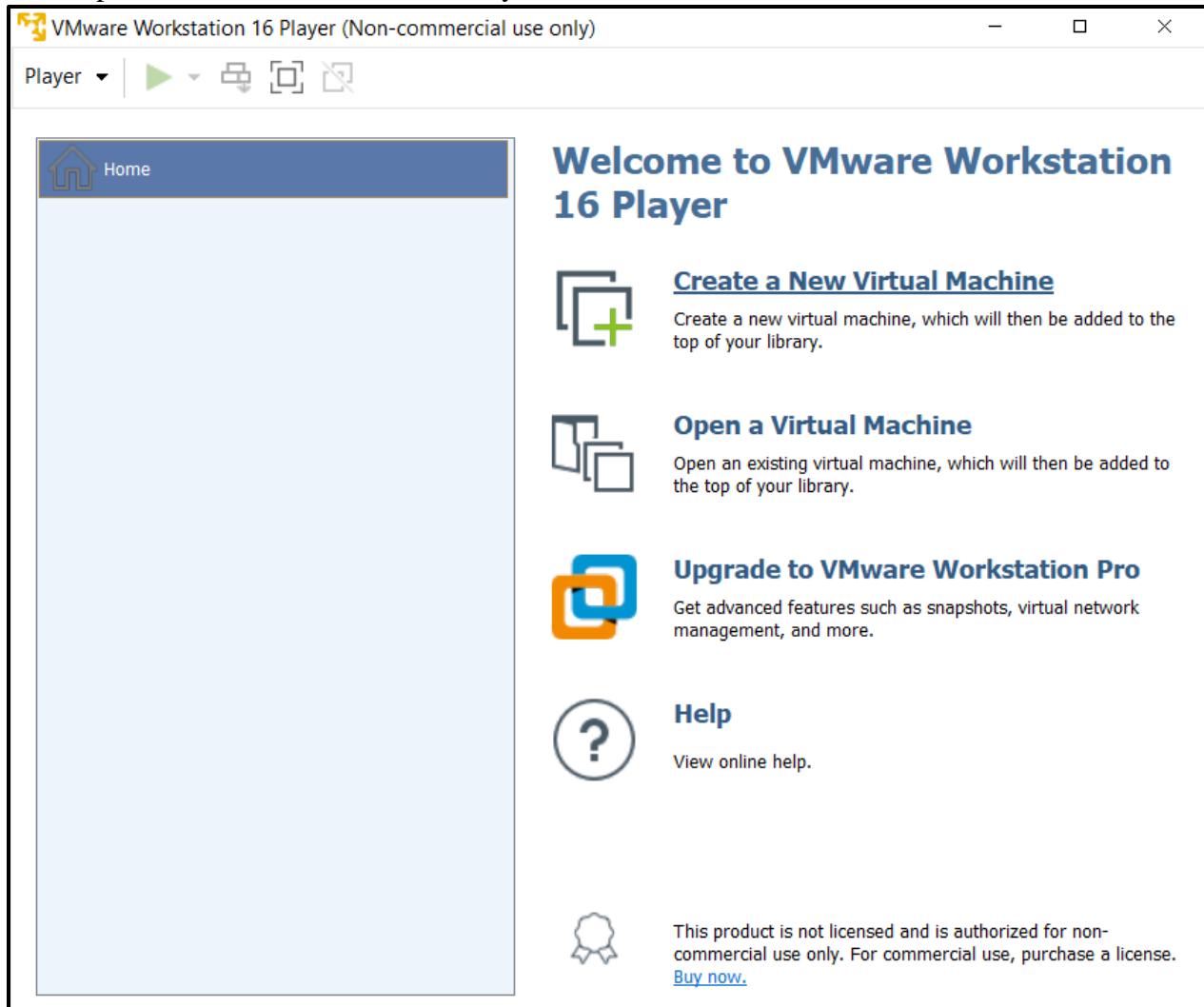


9. Now you can start the VMware Workstation Player by clicking on the shortcut on Desktop. Below is the home screen of the VMware Workstation player.

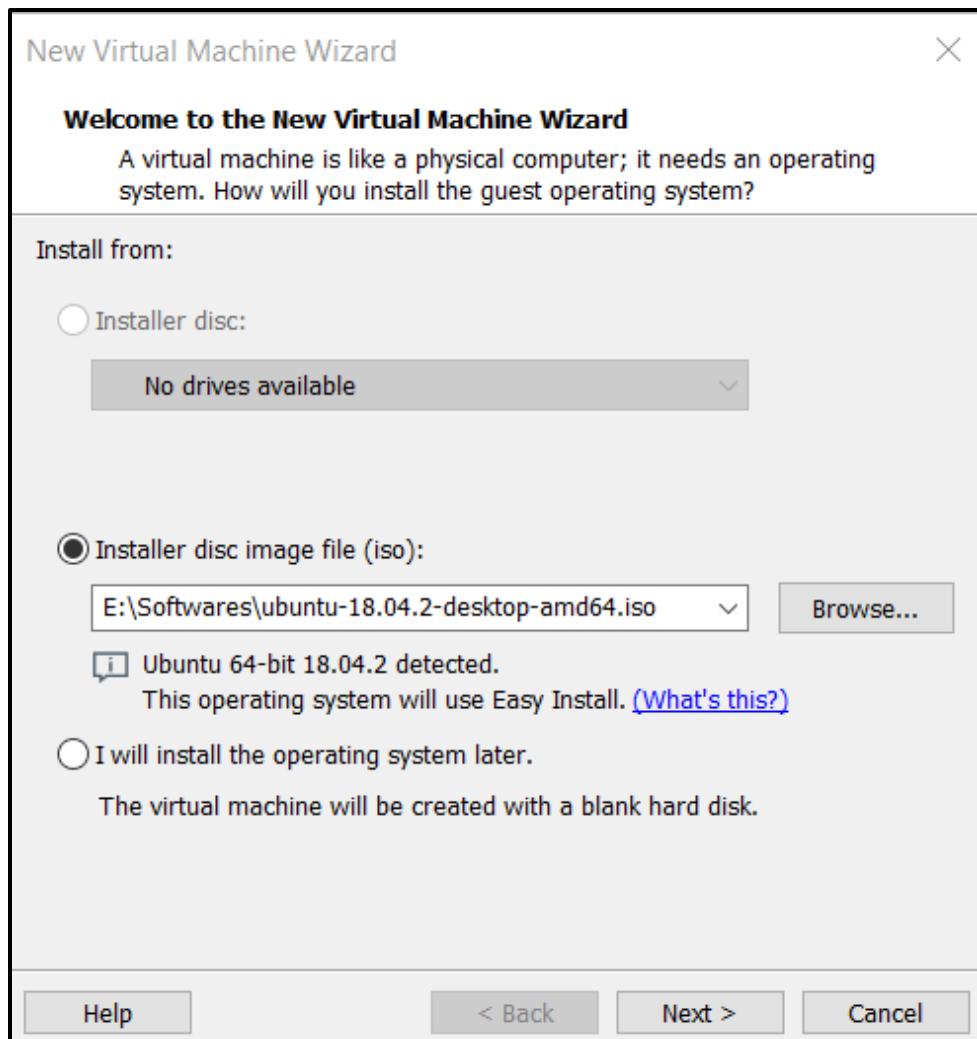


Install Ubuntu Linux on VMWare Workstation

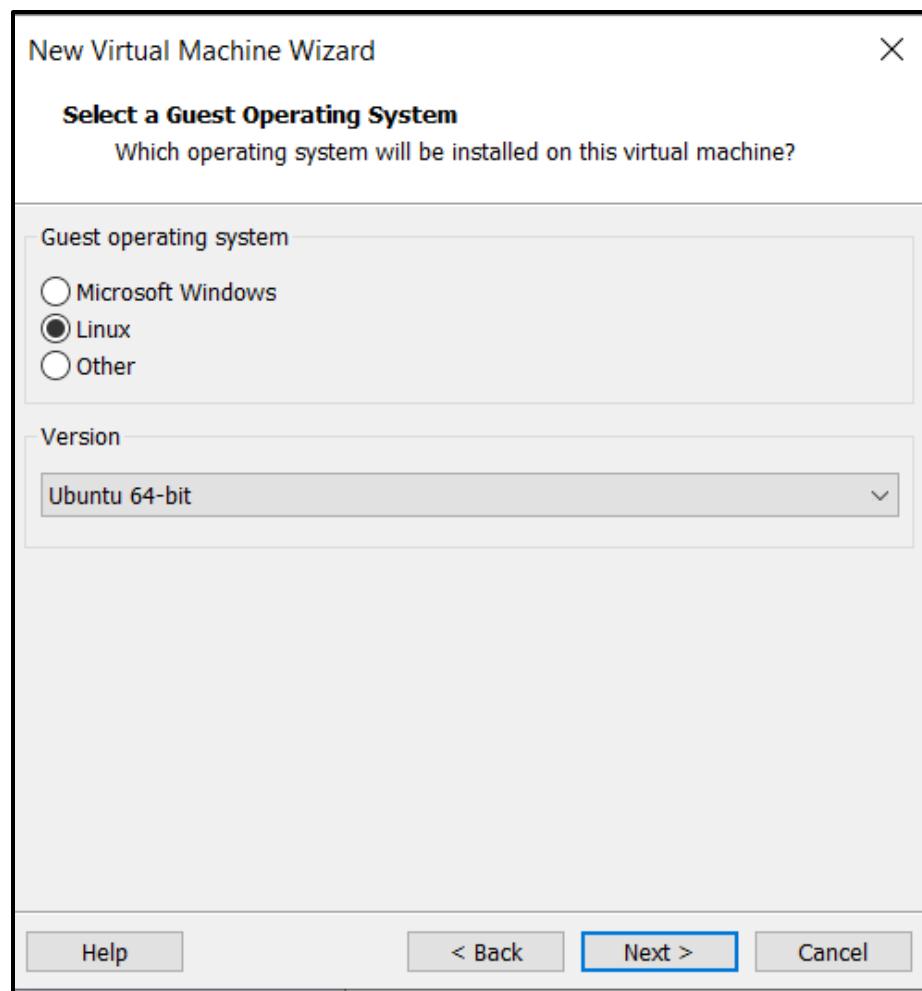
1. Open the VMware Workstation Player after installation. Create a new Virtual Machine.



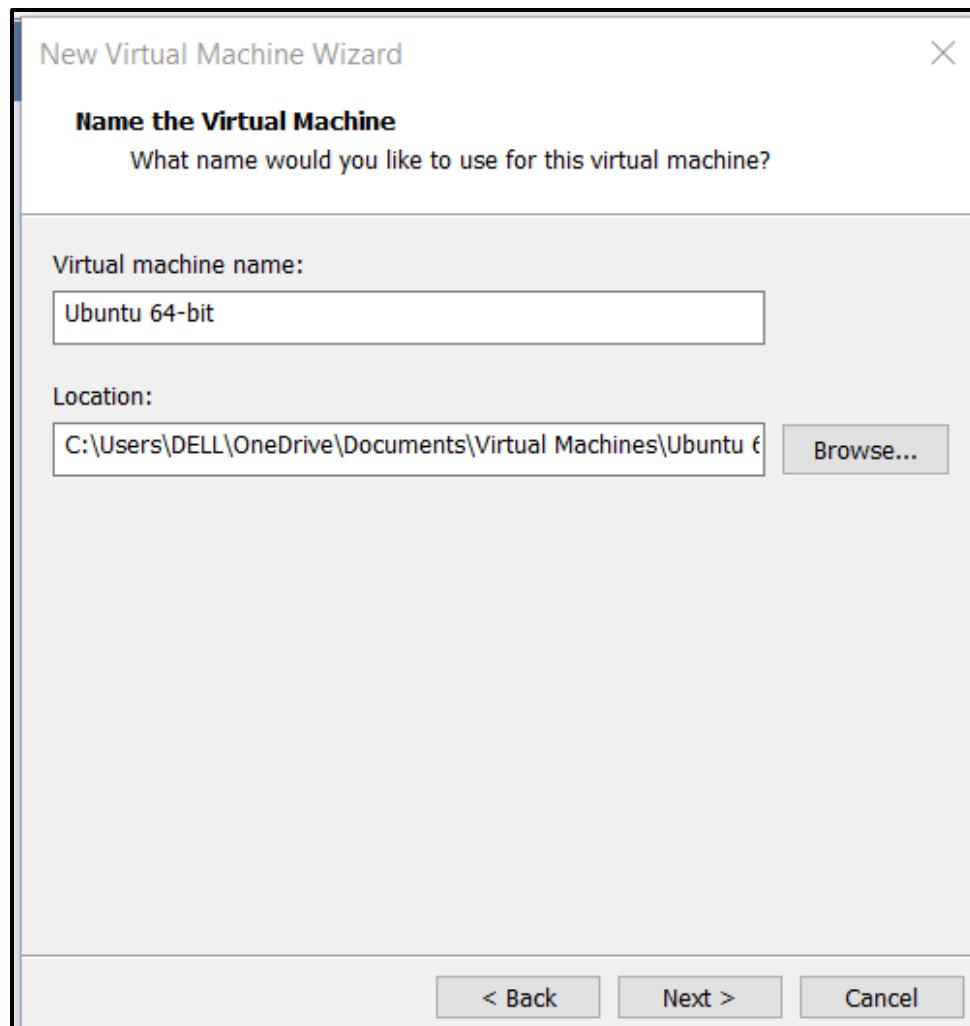
2. Select ubuntu iso file and click on next to continue.



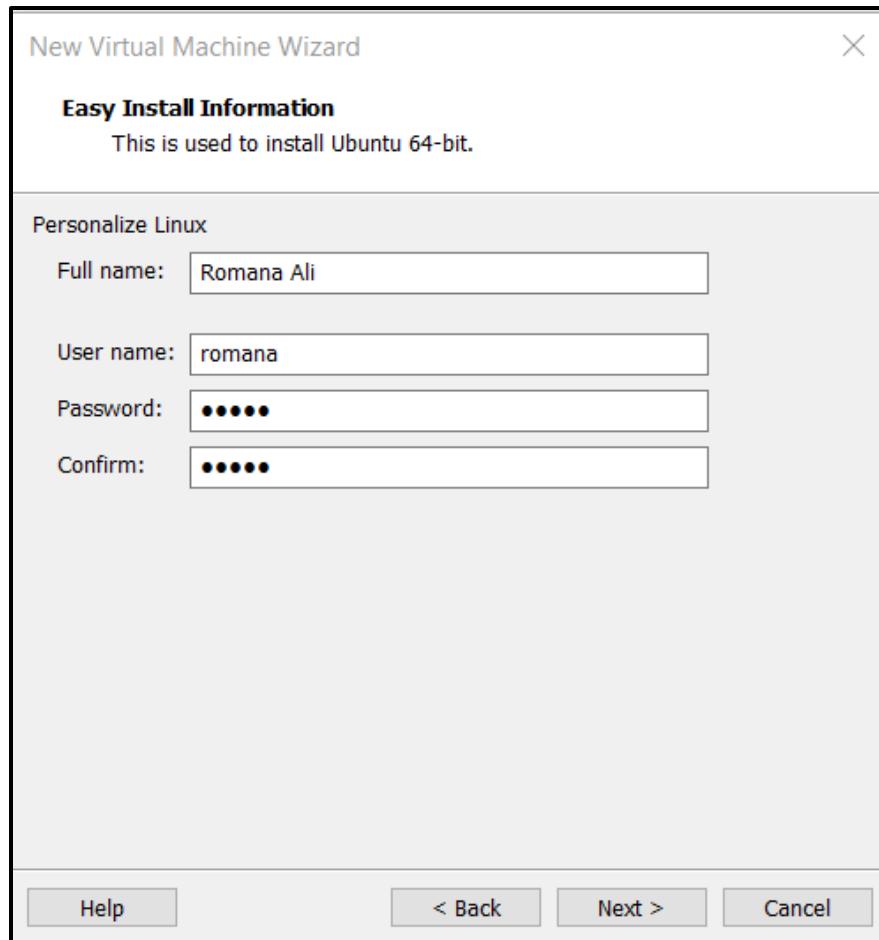
3. Select Linux and version of the Linux.



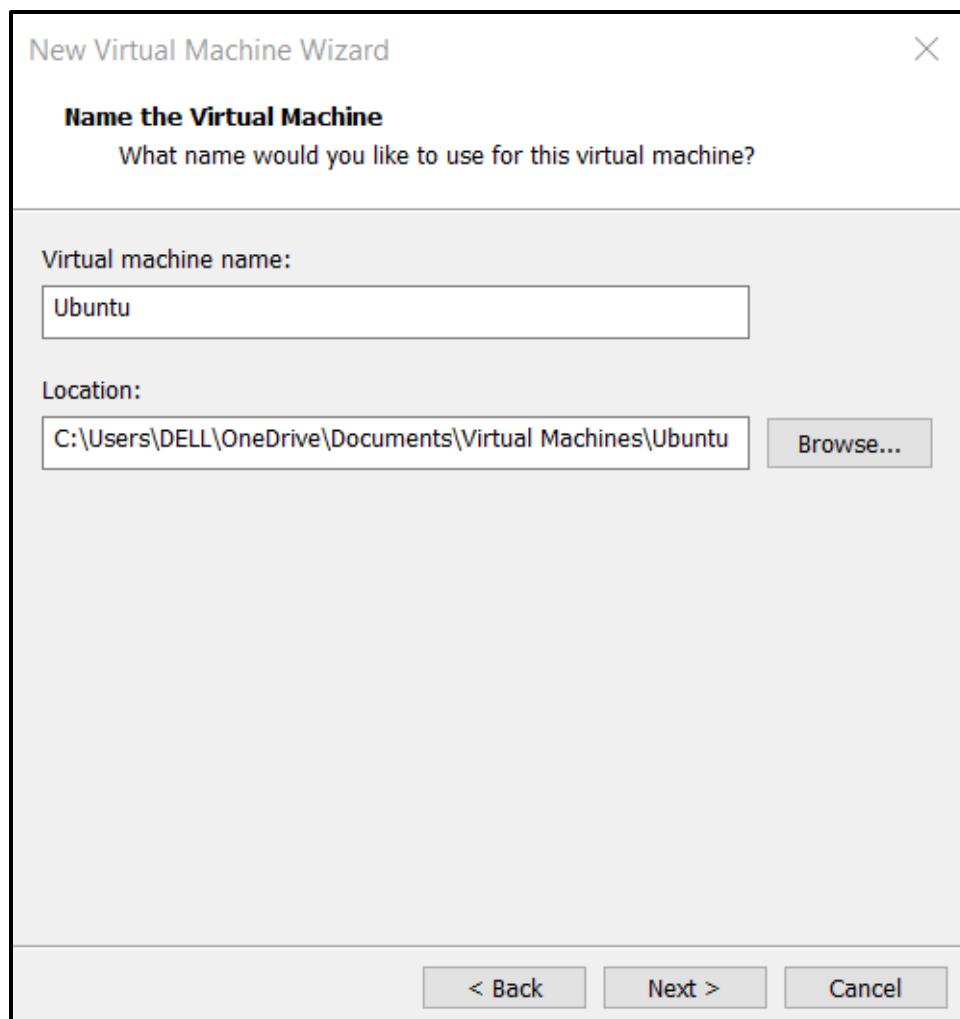
4. Name the virtual machine.



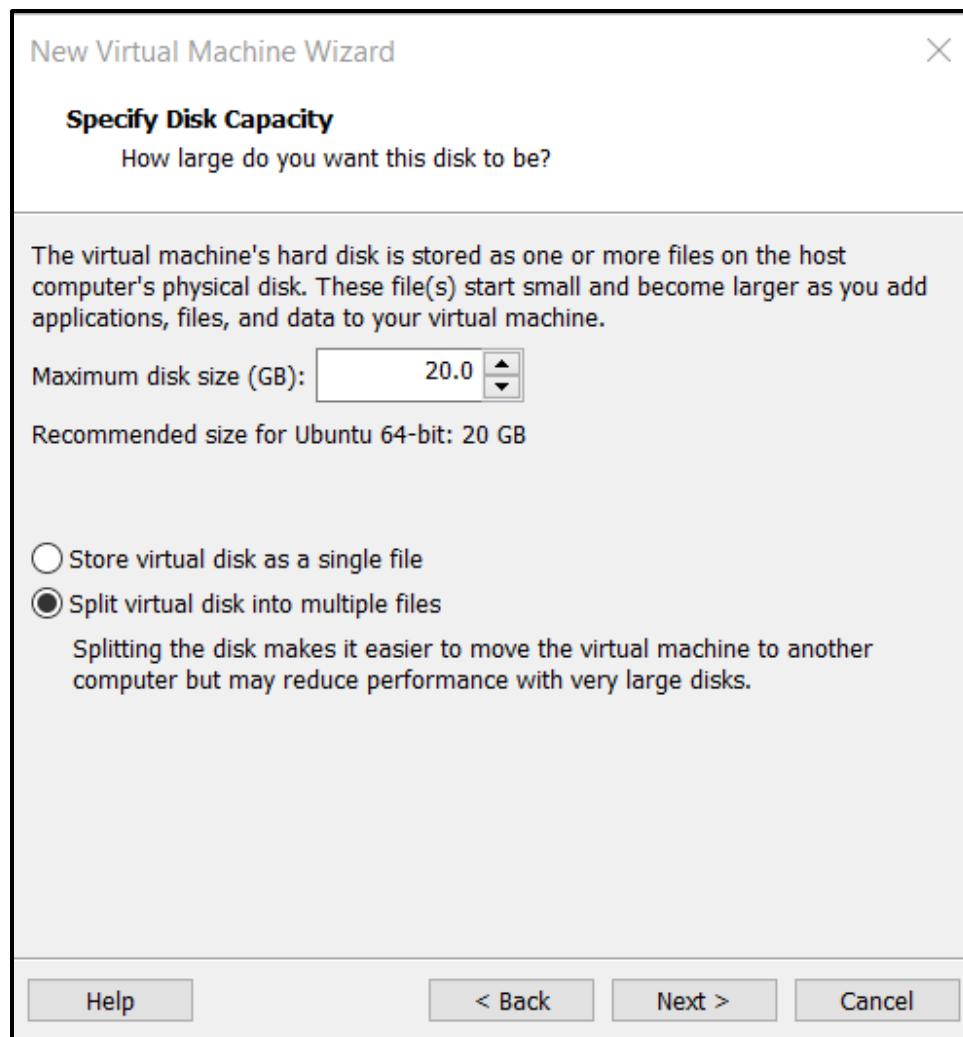
5. Provide credentials for new virtual machine. And click on next to continue.
Note: remember these credentials for future login.



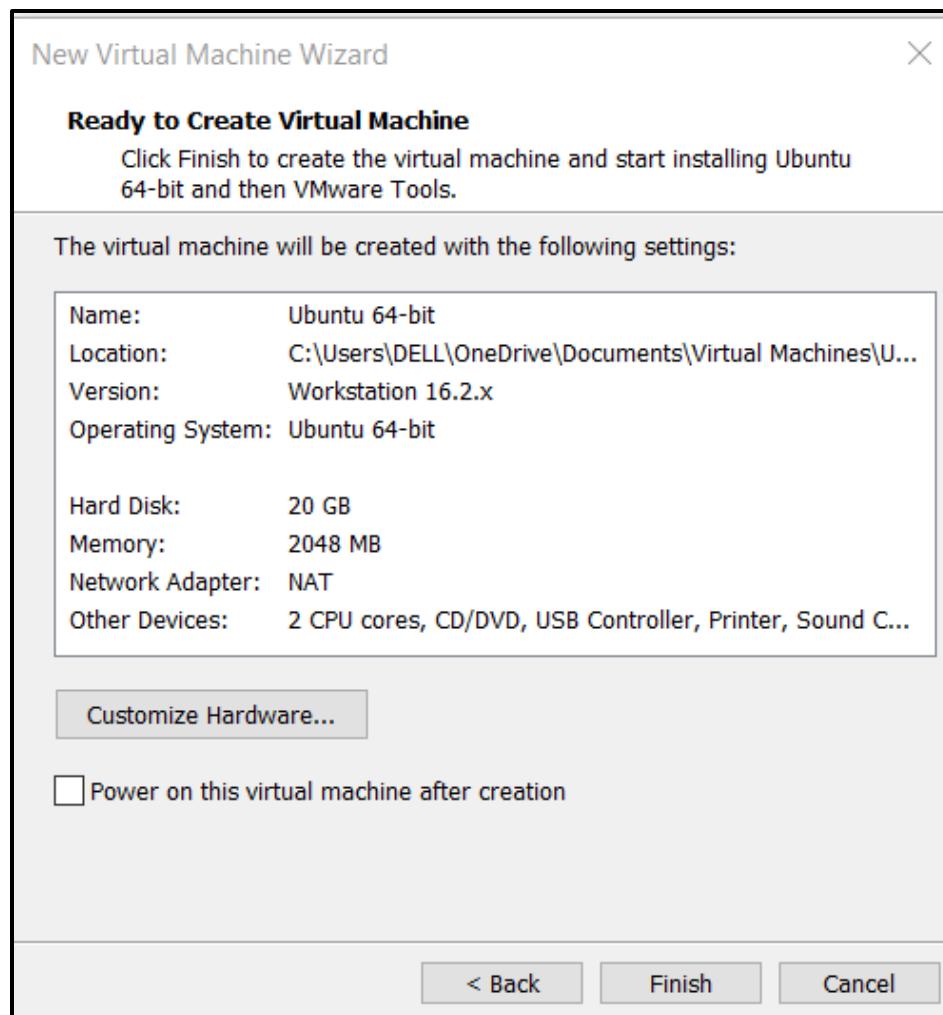
6. Set name for the new virtual machine. And click on next to continue.



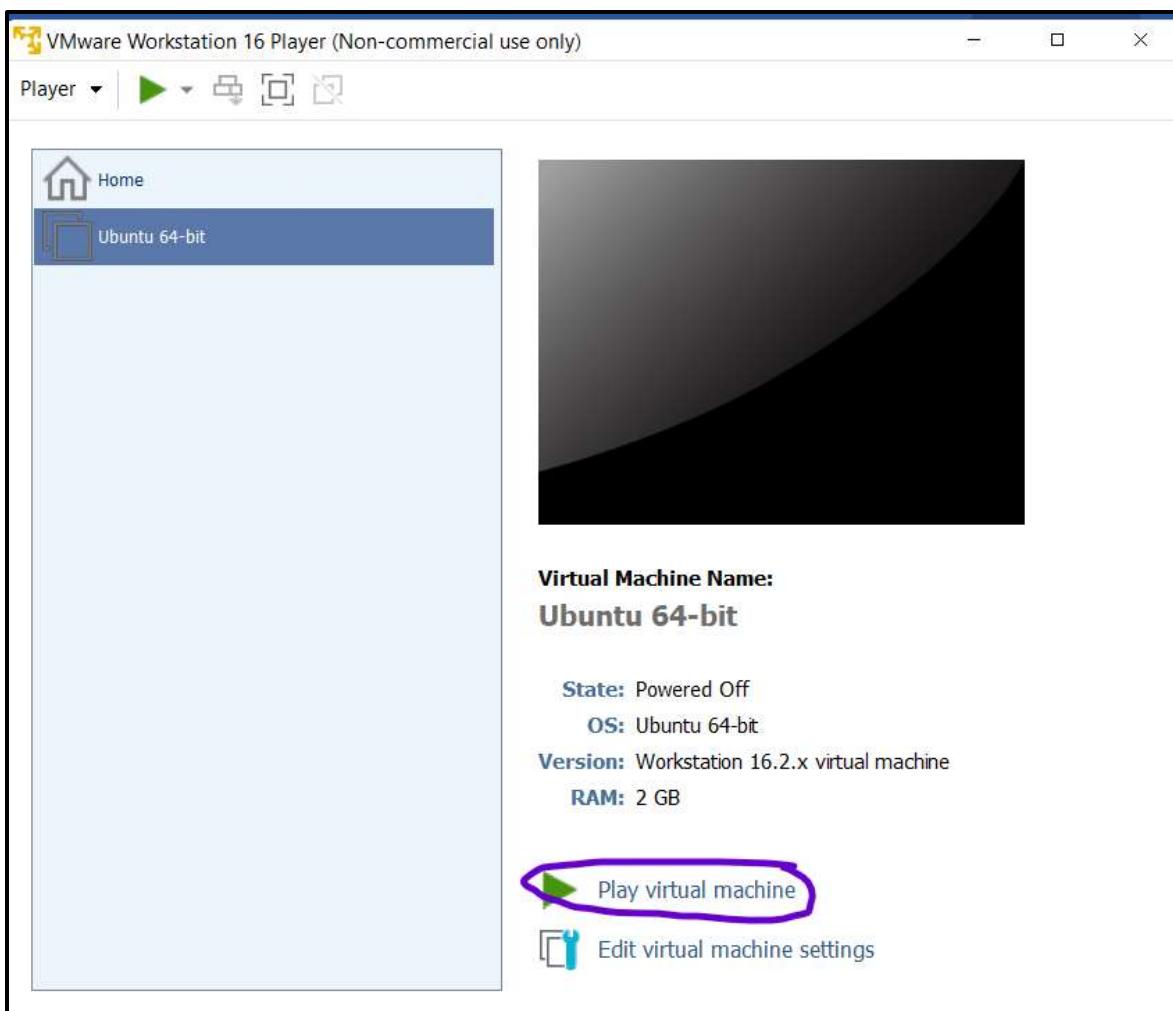
7. Use recommended settings but you can reduce its size. click on next to continue.



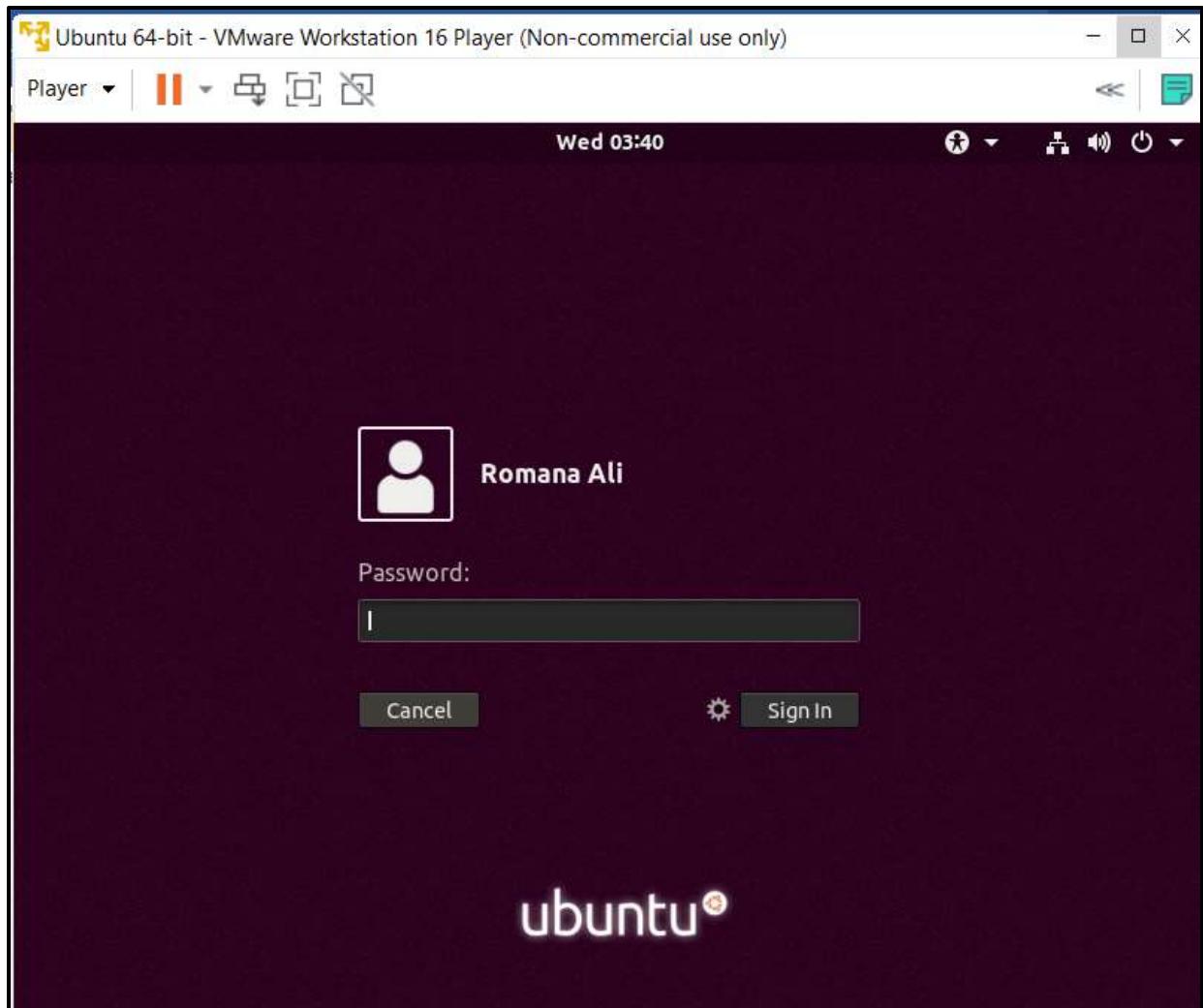
8. Click on finish to complete the setup.



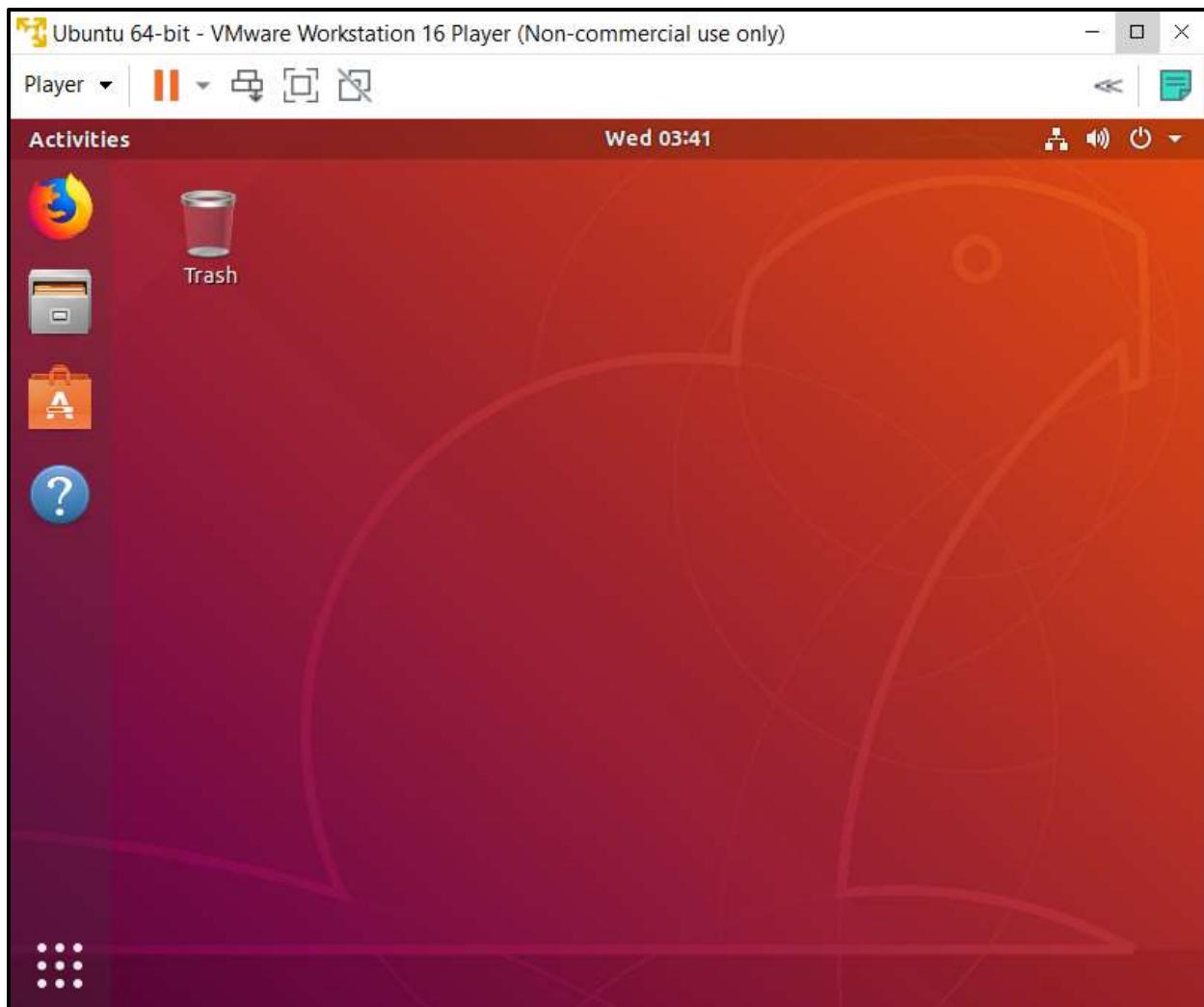
9. Play virtual machine. It will take some time for installation.



10. Provide the password as provided at step 5.



11. Welcome to Ubuntu Virtual Machine



Install G++ the C++ Compiler on Ubuntu VMWare

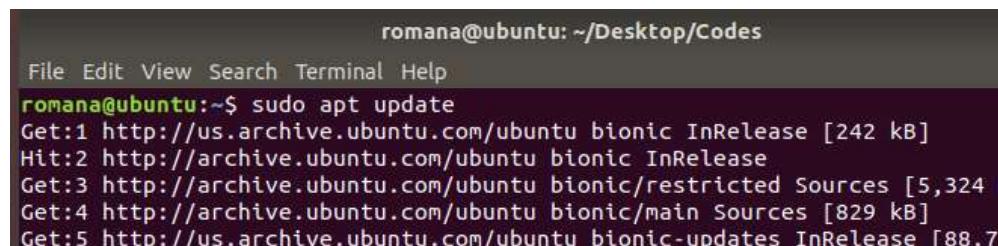
The GNU Compiler Collection (GCC) is a collection of compilers and libraries for C, C++. Many open-source projects, including the GNU tools and the Linux kernel, are compiled with GCC. To be able to add new repositories and install packages on your Ubuntu system, you must be logged in as root or user with sudo privileges.

Installing G++ on Ubuntu

The default Ubuntu repositories contain a meta-package named build-essential that contains the GCC compiler and a lot of libraries and other utilities required for compiling software.

1. Start by updating the packages list:

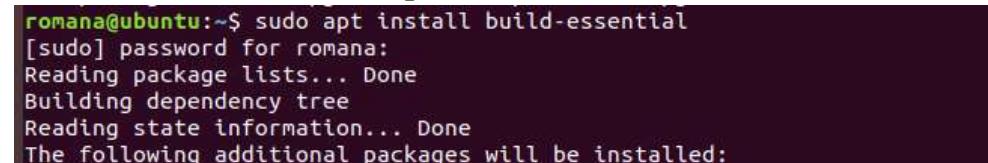
```
$ sudo apt update
```



```
romana@ubuntu: ~/Desktop/Codes
File Edit View Search Terminal Help
romana@ubuntu:~$ sudo apt update
Get:1 http://us.archive.ubuntu.com/ubuntu bionic InRelease [242 kB]
Hit:2 http://archive.ubuntu.com/ubuntu bionic InRelease
Get:3 http://archive.ubuntu.com/ubuntu bionic/restricted Sources [5,324 kB]
Get:4 http://archive.ubuntu.com/ubuntu bionic/main Sources [829 kB]
Get:5 http://us.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
```

2. Install the build-essential package by typing:

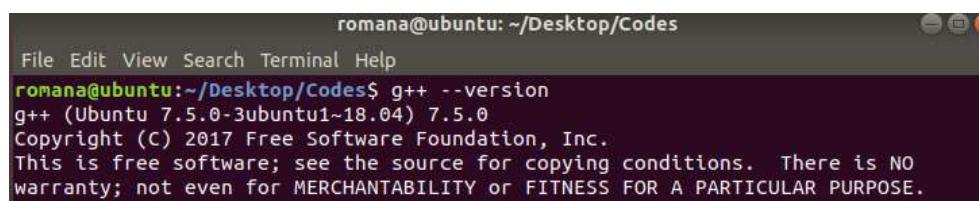
```
$ sudo apt install build-essential
```



```
romana@ubuntu:~$ sudo apt install build-essential
[sudo] password for romana:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
```

3. Check the version of g++ compiler by following command.

```
$ g++ --version
```



```
romana@ubuntu: ~/Desktop/Codes
File Edit View Search Terminal Help
romana@ubuntu:~/Desktop/Codes$ g++ --version
g++ (Ubuntu 7.5.0-3ubuntu1-18.04) 7.5.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Installing VSCode on Ubuntu

1. Install snap inorder to download latest version of vs code by typing.

```
$ sudo apt-get install snap
```

```
romana@ubuntu:~$ sudo apt-get install snap
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  snap
0 upgraded, 1 newly installed, 0 to remove and 580 not upgraded.
Need to get 375 kB of archives.
After this operation, 2,714 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu bionic/universe amd64 snap amd64 2013-11-29-8 [375 kB]
Fetched 375 kB in 3s (120 kB/s)
Selecting previously unselected package snap.
(Reading database ... 117109 files and directories currently installed.)
Preparing to unpack .../snap_2013-11-29-8_amd64.deb ...
Unpacking snap (2013-11-29-8) ...
Setting up snap (2013-11-29-8) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
```

2. Install VS code by following command.

```
$ sudo snap install --classic code
```

```
romana@ubuntu:~$ sudo snap install --classic code
code c722ca6c from Visual Studio Code (vscode✓) installed
```

Basic C++ Program

let's create hello world C++ program. Save the following code as hello.cpp text file and run it. Perform the steps below.

1. Create a text file hello.cpp by following command and write simple code as shown in figure

```
$ nano hello.cpp
```

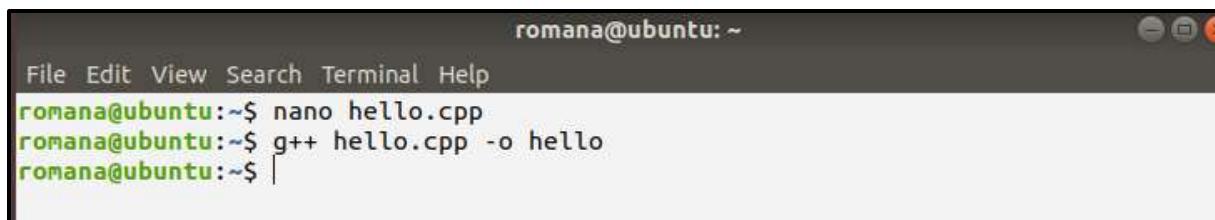


```
GNU nano 2.9.3          hello.cpp

#include <iostream>
using namespace std;
int main ()
{
    cout << " Hello World" << endl;
    return 0;
}
```

2. Close the editor and compile it by using

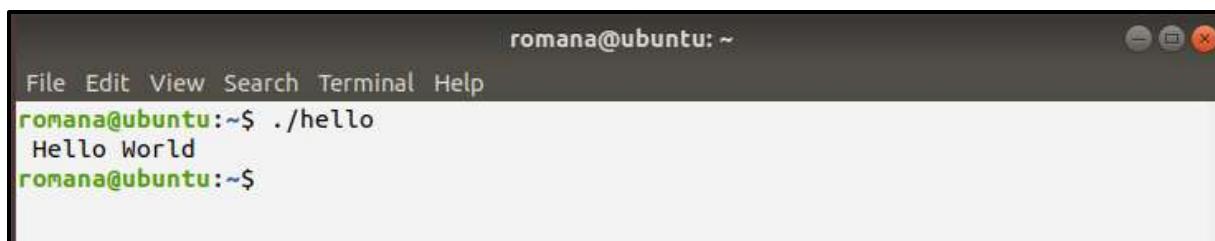
```
$ program-source-code.cpp -o executable-file-name
```



```
romana@ubuntu: ~
File Edit View Search Terminal Help
romana@ubuntu:~$ nano hello.cpp
romana@ubuntu:~$ g++ hello.cpp -o hello
romana@ubuntu:~$
```

3. To run or execute the program use following command

```
$ ./executable-file-name
```



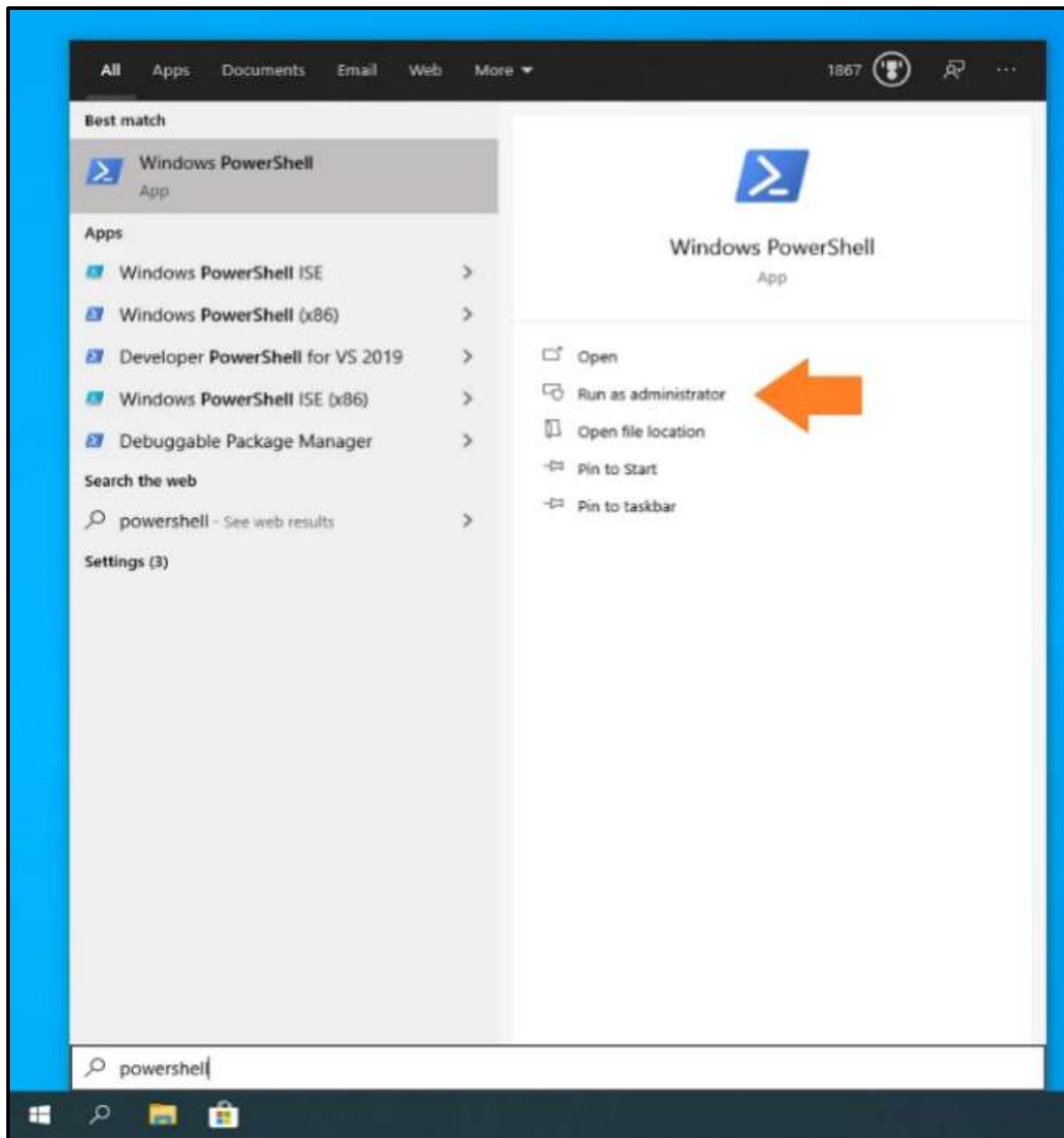
```
romana@ubuntu: ~
File Edit View Search Terminal Help
romana@ubuntu:~$ ./hello
Hello World
romana@ubuntu:~$
```

Method 02: Install Ubuntu on WSL2 on Windows 10

<https://ubuntu.com/tutorials/install-ubuntu-on-wsl2-on-windows-10#1-overview>

Windows Subsystem for Linux (WSL) allows you to install a complete Ubuntu terminal environment in minutes on your Windows machine, allowing you to develop cross-platform applications without leaving Windows.

1. Search for Windows PowerShell in your Windows search bar, then select **Run as administrator**.



2. At the command prompt type:

wsl --install

And wait for the process to complete. For WSL to be properly activated, you will now need to restart your computer.

3. WSL supports a variety of Linux distributions, including the latest Ubuntu release, Ubuntu 20.04 LTS and Ubuntu 18.04 LTS. You can find them by opening the Microsoft store app and searching for Ubuntu. Choose the distribution you prefer and then click on Get as shown in the following screenshot:



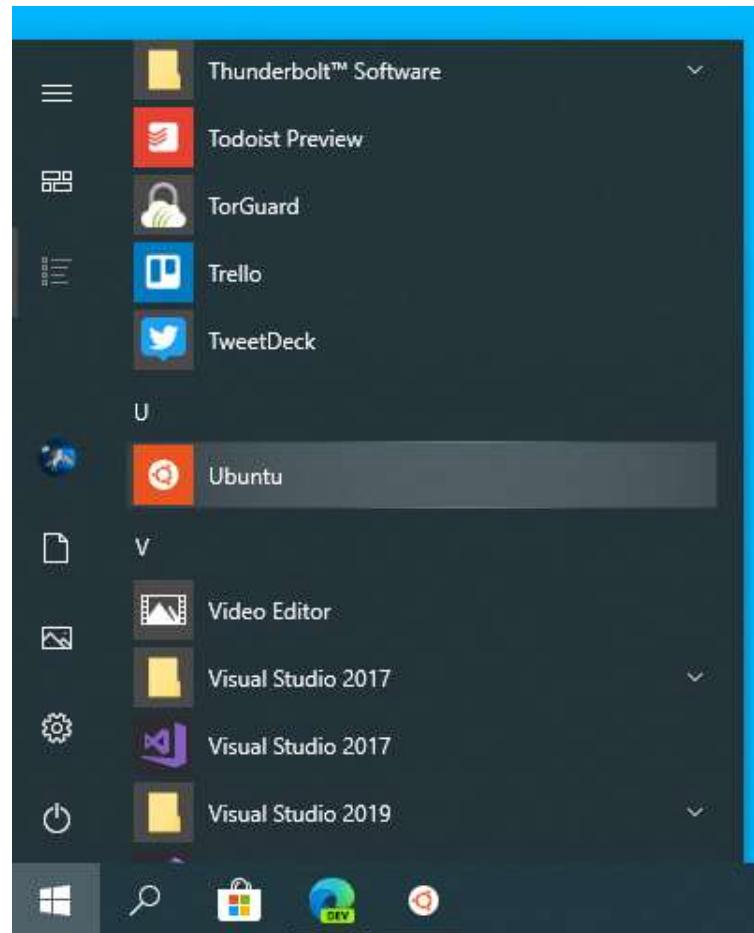
4. Ubuntu will then install on your machine.

5. There is a single command that will install both WSL and Ubuntu at the same time. When opening PowerShell for the first time, simply modify the initial instruction to:

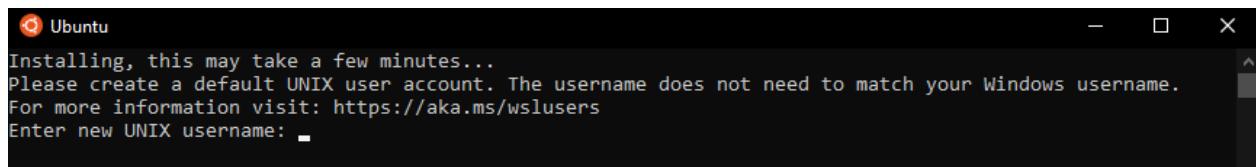
wsl --install -d ubuntu

This will install both WSL and Ubuntu! Don't forget to restart your machine before continuing.

6. Once installed, you can either launch the application directly from the store or search for **Ubuntu** in your Windows search bar.



7. Once Ubuntu has finished its initial setup you will need to create a username and password (this does not need to match your Windows user credentials).



8. Finally, it's always good practice to install the latest updates with the following commands, entering your password when prompted.

sudo apt update

Then

sudo apt upgrade

Press Y when prompted.

Install G++ the C++ Compiler and VS Code on Ubuntu WSL2

The GNU Compiler Collection (GCC) is a collection of compilers and libraries for C, C++. Many open-source projects, including the GNU tools and the Linux kernel, are compiled with GCC. To be able to add new repositories and install packages on your Ubuntu system, you must be logged in as root or user with sudo privileges.

Installing G++ on Ubuntu

Use following commands on Ubuntu to install compilers for running C++ programs on Ubuntu.

3. Start by updating the packages list:

```
$ sudo apt update
```

```
romana@ubuntu:~/Desktop/Codes
File Edit View Search Terminal Help
romana@ubuntu:~$ sudo apt update
Get:1 http://us.archive.ubuntu.com/ubuntu bionic InRelease [242 kB]
Hit:2 http://archive.ubuntu.com/ubuntu bionic InRelease
Get:3 http://archive.ubuntu.com/ubuntu bionic/restricted Sources [5,324 kB]
Get:4 http://archive.ubuntu.com/ubuntu bionic/main Sources [829 kB]
Get:5 http://us.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
```

4. Install the build-essential package by typing:

```
$ sudo apt install build-essential
```

```
romana@ubuntu:~$ sudo apt install build-essential
[sudo] password for romana:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
```

5. Check the version of g++ compiler by following command.

```
$ g++ --version
```

```
romana@ubuntu:~/Desktop/Codes
File Edit View Search Terminal Help
romana@ubuntu:~/Desktop/Codes$ g++ --version
g++ (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Installing VSCode on Ubuntu

6. Install snap inorder to download latest version of vs code by typing.

```
$ sudo apt-get install snap
```

```
romana@ubuntu:~$ sudo apt-get install snap
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  snap
0 upgraded, 1 newly installed, 0 to remove and 580 not upgraded.
Need to get 375 kB of archives.
After this operation, 2,714 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu bionic/universe amd64 snap amd64 2013-11-29-8 [375 kB]
Fetched 375 kB in 3s (120 kB/s)
Selecting previously unselected package snap.
(Reading database ... 117109 files and directories currently installed.)
Preparing to unpack .../snap_2013-11-29-8_amd64.deb ...
Unpacking snap (2013-11-29-8) ...
Setting up snap (2013-11-29-8) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
```

7. Install VS code by following command.

```
$ sudo snap install --classic code
```

```
romana@ubuntu:~$ sudo snap install --classic code
code c722ca6c from Visual Studio Code (vscode✓) installed
```

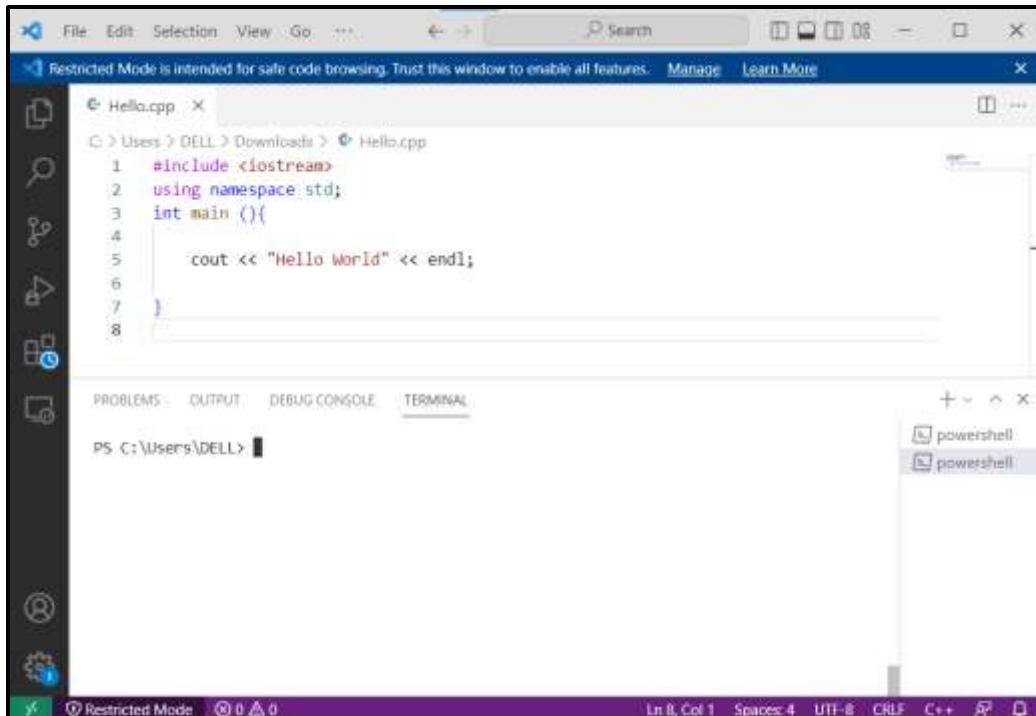
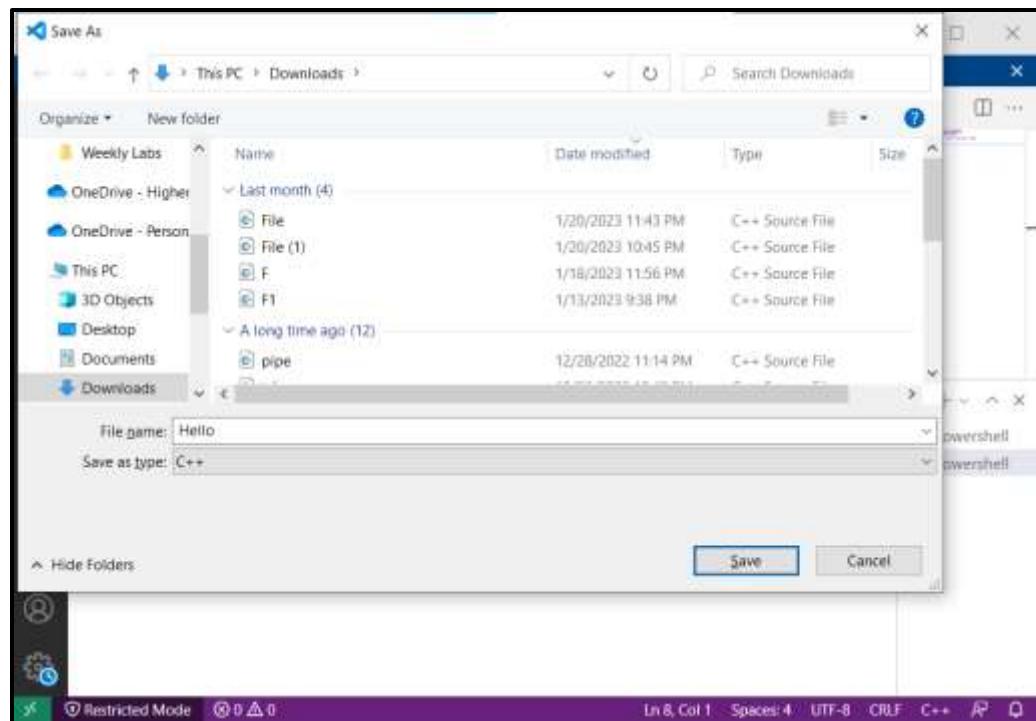
8. Let's make a directory **Codes** on Desktop. Make and open a new file of VS Code in the Codes directory by following command

```
$ code .
```

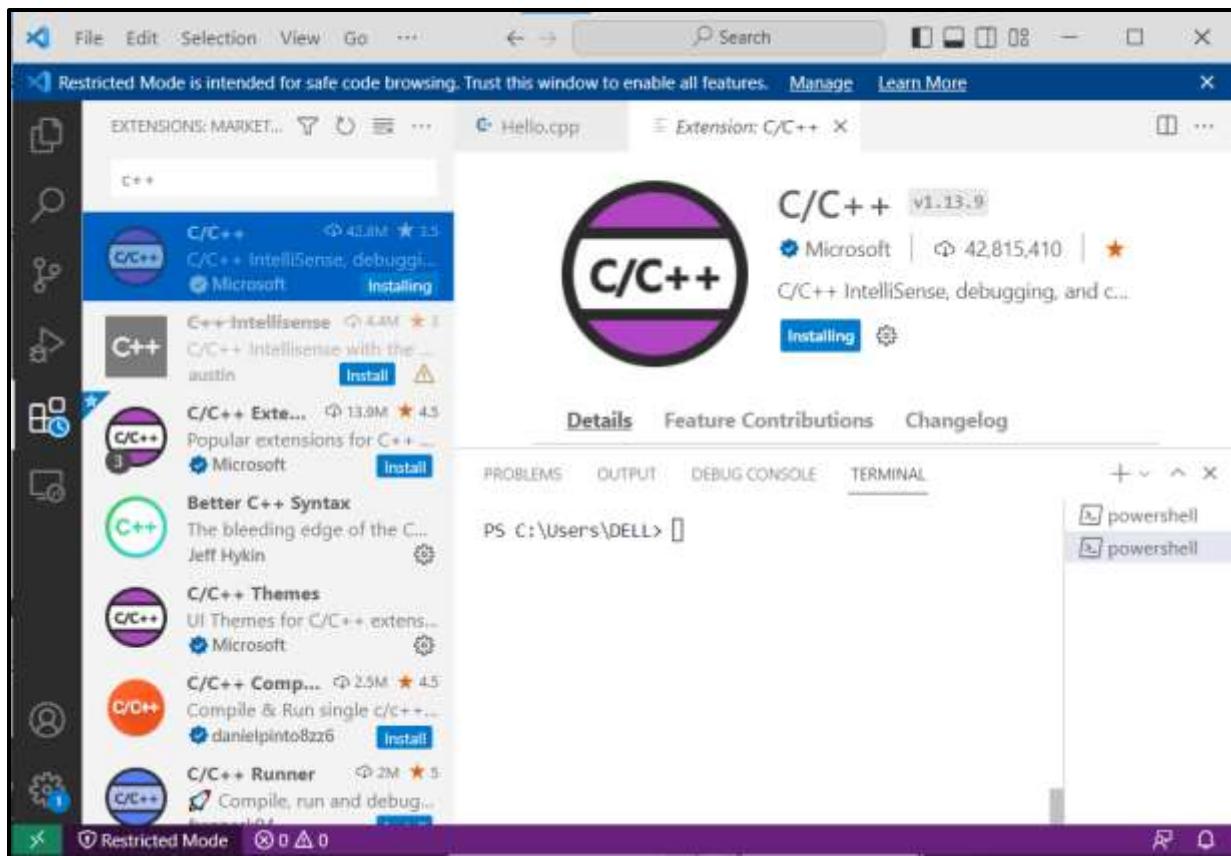
```
romana@ubuntu: ~/Desktop/Codes
```

```
File Edit View Search Terminal Help
romana@ubuntu:~/Desktop$ cd Codes
romana@ubuntu:~/Desktop/Codes$ code .
romana@ubuntu:~/Desktop/Codes$
```

9. In Code directory, write a simple hello world program and save the file as .cpp file.



10. Install the C++ extensions and run the program using g++ compiler on terminal present in vs code.



11. In order to compile, code open ubuntu terminal. Write following in the vs code terminal and press enter. Ubuntu console will be opened in the vscode. As shown in the picture.

wsl

```

File Edit Selection View Go Run Terminal Help
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
Hello.cpp X
C: > Users > DELL > Downloads > Hello.cpp
8

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\DELL> wsl
Windows Subsystem for Linux is now available in the Microsoft Store!
You can upgrade by running 'wsl.exe --update' or by visiting https://aka.ms/wslstorepage
Installing WSL from the Microsoft Store will give you the latest WSL updates, faster.
For more information please visit https://aka.ms/wslstoreinfo

Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 4.4.0-19041-Microsoft x86_64)

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

System information as of Sun Feb 19 11:28:23 PKT 2023

System load: 0.52      Users logged in:  0
Usage of /home: unknown  IPv4 address for eth2: 192.168.67.1
Memory usage: 69%       IPv4 address for eth3: 192.168.73.1
Swap usage:  4%         IPv4 address for wifi2: 192.168.10.11
Processes:   7

98 updates can be applied immediately.
49 of these updates are standard 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 a day. To disable it please create the
/home/romanaali/.hushlogin file.
romanaali@DESKTOP-EUCC41L:/mnt/c/Users/DELL$ 
  
```

12. Hello.cpp is present in downloads in windows whose address is “/mnt/c/users/Dell/downloads”. We will change directories using `cd`.

The screenshot shows a code editor window with the following details:

- Title Bar:** File Edit Selection View Go ... Search Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
- File Explorer:** Shows the file path: C:\Users\DELL\Downloads\Hello.cpp
- Code Editor:** Displays the following C++ code:

```
1 #include <iostream>
2 using namespace std;
3 int main (){
4
5     cout << "Hello World" << endl;
6
7 }
```
- Terminal:** Shows the command-line interface output:

```
romanaali@DESKTOP-EUCC41L:/mnt/c/users/Dell/downloads$ cd mnt/c/users/Dell/downloads
romanaali@DESKTOP-EUCC41L:/mnt/c/users/Dell/downloads$
```
- Sidebar:** Includes icons for file operations like Open, Save, Find, and Copy/Paste.
- Status Bar:** Shows Restricted Mode, 0 errors, 0 warnings, Line 8, Column 1 (5 selected), Spaces: 4, UTF-8, CRLF, C++, and a file icon.

13. Use following command to compile the hello.cpp.

g++ Hello.cpp -o h

This command will create a object file of Hello.cpp named h. use following to run this object file/ executable. As shown in the figure.

./h



The screenshot shows a code editor with a terminal window integrated. The code editor has a tab for 'Hello.cpp' with the following content:

```
C:\> Users > DELL > Downloads > Hello.cpp
1 #include <iostream>
2 using namespace std;
3 int main (){
4
5     cout << "Hello World" << endl;
6
7
8 }
```

Below the code editor is a terminal window with the following session:

```
romanaali@DESKTOP-EUCC41L:/mnt/c/users/Dell/downloads$ g++ Hello.cpp -o h
romanaali@DESKTOP-EUCC41L:/mnt/c/users/Dell/downloads$ ./h
Hello World
romanaali@DESKTOP-EUCC41L:/mnt/c/users/Dell/downloads$
```

A sidebar on the right shows a list of open terminals, with one labeled 'powershell' being selected.

OPERATING SYSTEM LABORATORY MANUAL



UNIVERSITY OF THE PUNJAB

**FACULTY OF COMPUTING & INFORMATION TECHNOLOGY, LAHORE
DEPARTMENT OF COMPUTER SCIENCE**

Course:	Operating System Lab	Date:
Course Code:	CC-217-3L	Max Marks: 40
Faculty/Instructor's Name & Email:	Dr. Ahmad Hassan Butt (ahmad.hassan@pucit.edu.pk)	

**LAB MANUAL # 3
(SPRING 2023)**

Name: _____ Enroll No: _____

Objective(s) :

To study and execute the commands in Linux.

Lab Tasks :

Task 1 : Execute the Date Commands and write the output.

Task 2: Execute the below mentioned LINUX Commands and generate output.

Task 3 : Execute the below File Commands and write the output.

Task 4 : Execute FILTERS AND PIPES commands and write the output

Lab Grading Sheet :

Task	Max Marks	Obtained Marks	Comments(<i>if any</i>)
1.	10		
2.	10		
3.	10		
4.	10		
Total	40		Signature

Note : Attempt all tasks and get them checked by your Instructor

Lab 03: LINUX Commands

Objective(s):

- To study and execute the commands in Linux.

Tool(s) used:

Ubuntu

General Purpose utility LINUX Commands

Task 01 Execute the Date Commands and write the output.

This command is used to display the current date and time.

Syntax: \$date

Output:

Options:

a = Abbreviated weekday.
A = Full weekday.
b = Abbreviated month.
B = Full month.
c = Current day and time.
C = Display the century as a decimal number.
d = Day of the month.
D = Day in “mm/dd/yy” format
h = Abbreviated month day.
H = Display the hour.
m = Month of the year.
M = Minute.
P = Display AM or PM
S = Seconds
T = HH:MM:SS format
y = Display the year in 2 digit.
Y = Display the full year.
Z = Time zone.

To change the format:

Syntax: \$date +%H-%M-%S

Output:

Calendar Command

This command is used to display the calendar of the year or the particular month of calendar year.

Syntax

\$cal year

\$cal month year

Here the first syntax gives the entire calendar for given year & the second Syntax gives the calendar of reserved month of that year.

Output:

Task 02 Execute the below mentioned LINUX Commands and generate output.

Echo Command

This command is used to print the arguments on the screen.

Syntax: \$echo text

Output:

Banner Command

It is used to display the arguments in „#“ symbol.

Syntax: \$banner <arguments>

Output:

‘ who’ Command

It is used to display who are the users connected to our computer currently.

Syntax: \$who – option’s

Options

- H–Display the output with headers.
- b–Display the last booting date or time or when the system was lastly rebooted.

Output:

'whoami' Command

Display the details of the current working directory.

Syntax: \$whoami

Output:

'Binary' Calculator Command

It will change the „\$“ mode and in the new mode, arithmetic operations such as +,- ,*,/,%,n,sqrt(),length(),=, etc can be performed. This command is used to go to the binary calculus mode.

Syntax: \$bc operations ^d

- 1 base – input base
- 0 base – output base are used for base conversions.
- Base: Decimal = 1 Binary = 2 Octal = 8 Hexa = 16

Output:

'CLEAR' Command

It is used to clear the screen.

Syntax: \$clear

Task 03 Execute the below File Commands and write the output.

Create a File

To create a new file in the current directory we use CAT command.

Syntax:

\$cat > filename.

The > symbol is re-directory we use cat command.

Output:

Display A File

To display the content of file mentioned we use CAT command without “>” operator.

Syntax:

\$cat <filename.

Options -s = to neglect the warning /error message.

Output:

Copying Contents

To copy the content of one file with another. If file does not exist, a new file is created and if the file exists with some data then it is appended.

Syntax:

\$ cat source filename >> destination filename it is to avoid overwriting.

Options: -n content of file with numbers included with blank lines.

Syntax: \$cat -n filename

Output:

Copying Contents From One File To Another

To copy the contents from source to destination file. so that both contents are same.

Syntax

\$cp source filename destination filename

Output:

MOVE Command

To completely move the contents from source file to destination file and to remove the source file.

Syntax: \$ mv source filename destination filename

Output:

REMOVE Command

To permanently remove the file we use this command.

Syntax: \$rm filename

Output:

WORD Command

To list the content count of no of lines, words, characters.

Syntax: \$wc filename

Options:

- -c – to display no of characters. -l – to display only the lines.
- -w – to display the no of words.

Output:

PAGE Command

This command is used to display the contents of the file page wise & next page can be viewed by pressing the enter key.

Syntax: \$pg filename

Output:

Task 04 Execute FILTERS AND PIPES commands and write the output

HEAD

It is used to display the top ten lines of file.

Syntax: \$head filename

Output:

TAIL

This command is used to display the last ten lines of file.

Syntax: \$tail filename

Output:

SORT

This command is used to sort the data's in some order.

Syntax: \$sort filename

Output:

PIPE

It is a mechanism by which the output of one command can be channeled into the input of another command.

Syntax: echo 1+1|bc

Output:

TR

The tr filter is used to translate one set of characters from the standard inputs to another.

Syntax: \$tr “[a-z]” “[A-Z]”

Output:

OPERATING SYSTEM LABORATORY MANUAL



UNIVERSITY OF THE PUNJAB

**FACULTY OF COMPUTING & INFORMATION TECHNOLOGY, LAHORE
DEPARTMENT OF COMPUTER SCIENCE**

Course:	Operating System Lab	Date:
Course Code:	CC-217-3L	Max Marks: 40
Faculty/Instructor's Name & Email:	Dr. Ahmad Hassan Butt (ahmad.hassan@pucit.edu.pk)	

**LAB MANUAL # 4
(SPRING 2023)**

Name: _____ Enroll No: _____

Objective(s) :

To understand working with VIM Editor.

Lab Tasks :

Task 1 : Familiarity with Vi Editor.

Task 2: Compiling and executing a C++ program in VIM

Task 3 : You can convert temperature from degrees Celsius to degrees Fahrenheit by multiplying by 9/5 and adding 32. Write a program that allows the user to enter a floating-point number representing degrees Celsius, and then displays the corresponding degrees Fahrenheit.

Task 4 : Write and run a program that simulates a simple calculator. It reads two integers and a character. If the character is a +, the sum is printed; if it is a -, the difference is printed; if it is a *, the product is printed; if it is a /, the quotient is printed; and if it is a %, the remainder is printed.

Lab Grading Sheet :

Task	Max Marks	Obtained Marks	Comments(<i>if any</i>)
1.	10		
2.	10		
3.	10		
4.	10		
Total	40		Signature

Note : Attempt all tasks and get them checked by your Instructor

Lab 04: Editor Commands

Objective(s):

To understand working with VIM Editor.

Tool(s) used:

Ubuntu, VIM Editor

VI Editor

The Vi editor is a visual editor used to create and edit text, files, documents and programs. It displays the content of files on the screen and allows a user to add, delete or change part of text. There are three modes available in the Vi editor, they are

1. Command mode
2. Input (or) insert mode.

Task 01 Familiarity with Vi Editor.Starting Vi

The Vi editor is invoked by giving the following commands in LINUX prompt.

Syntax: \$vi <filename> (or) \$vi

This command would open a display screen with 25 lines and with tilt (~) symbol at the start of each line. The first syntax would save the file in the filename mentioned and for the next the filename must be mentioned at the end.

Options: vi +n <filename> - this would point at the nth line (cursor pos).

Inserting and Replacing Commands

To move editor from command mode to edit mode, you have to press the <ESC> key. For inserting and replacing the following commands are used.

ESC a Command

This command is used to move the edit mode and start to append after the current character.

Syntax: <ESC>

ESC A Command

This command is also used to append the file, but this command append at the end of current line.

Syntax: <ESC> A

ESC i Command

This command is used to insert the text before the current cursor position.

Syntax: <ESC> i

ESC I Command

This command is used to insert at the beginning of the current line.

Syntax: <ESC> I

ESC o Command

This command is insert a blank line below the current line & allow insertion of contents.

Syntax: <ESC> o

ESC O Command

This command is used to insert a blank line above & allow insertion of contents.

Syntax: <ESC> O**ESC r Command**

This command is to replace the particular character with the given characters.

Syntax: <ESC> rx Where x is the new character.**ESC R Command**

This command is used to replace the particular text with a given text.

Syntax: <ESC> R text**<ESC> S Command**

This command is used to replace a current line with group of characters.

Syntax: <ESC> S**Cursor Movement in Vi****<ESC> h**

This command is used to move to the previous character typed. It is used to move to left of the text. It can also use to move character by character (or) a number of characters.

Syntax:

- <ESC> h - to move one character to left.
- <ESC> nh - to move “n” character to left.

<ESC> l

This command is used to move to the right of the cursor (i.e.) to the next character. It can also be used to move the cursor for a number of characters.

Syntax:

- <ESC> l – single character to right.

- <ESC> nl – “n” characters to right.

<ESC> j

This command is used to move down a single line or a number of lines.

Syntax:

- <ESC> j – single down movement.
- <ESC> nj – “n” times down movement.

<ESC> k

This command is used to move up a single line or a number of lines.

Syntax:

- <ESC> k – single line above.
- <ESC> nk – “n” lines above.

Enter (OR) N Enter

This command will move the cursor to the starting of next lines or a group of lines mentioned.

Syntax:

- <ESC> enter
- <ESC> n enter

<ESC> + Command

This command is used to move to the beginning of the next line.

Syntax:

- <ESC> +
- <ESC> n+

<ESC> - Command

This command is used to move to the beginning of the previous line.

Syntax:

- <ESC> -
- <ESC> n-

<ESC> 0

This command will bring the cursor to the beginning of the same current line.

Syntax: <ESC> 0

<ESC> \$

This command will bring the cursor to the end of the current line.

Syntax: <ESC> \$

<ESC> ^

This command is used to move to first character of first lines.

Syntax: <ESC> ^

<ESC> b Command

This command is used to move back to the previous word (or) a number of words.

Syntax:

- <ESC>b
- <ESC>nb

<ESC> e Command

This command is used to move towards and replace the cursor at last character of the word (or) no of words.

Syntax:

- <ESC> e
- <ESC>ne

<ESC> w Command

This command is used to move forward by a single word or a group of words.

Syntax:

- <ESC> w
- <ESC> nw

Deleting The Text From Vi

<ESC> x Command

To delete a character to right of current cursor positions, this command is used.

Syntax:

- <ESC> x
- <ESC> nx

<ESC> X Command

To delete a character to left of current cursor positions, this command is used.

Syntax:

- <ESC> X
- <ESC> nX

<ESC> dw Command

This command is to delete a single word or number of words to right of current cursor position.

Syntax:

- <ESC> dw
- <ESC> ndw

db Command

This command is to delete a single word to the left of the current cursor position.

Syntax:

- <ESC> db
- <ESC> ndb

<ESC> dd Command

This command is used to delete the current line (or) a number of lines below the current line.

Syntax:

- <ESC> dd
- <ESC> ndd

<ESC> d\$ Command

This command is used to delete the text from current cursor position to last character of current line.

Syntax: <ESC> d\$

SAVING AND QUITING FROM Vi

<ESC> w Command

To save the given text present in the file.

Syntax: <ESC>w

<ESC> q! Command

To quit the given text without saving.

Syntax: <ESC>:q!

<ESC> wq Command

This command quits the vi editor after saving the text in the mentioned file.

Syntax: <ESC>:wq

<ESC> x Command

This command is same as “wq” command it saves and quit.

Syntax: <ESC>x

<ESC> q Command

This command would quit the window but it would ask for again to save the file.

Syntax: <ESC>: q

Task 2 Compiling and executing a C++ program in VIM

Write and save the program in Vi:

Open a simple text editor Vi, command line code editor. Create new File – hello.cpp (.cpp extension is used to indicate that it's a c++ program). Then write a simple HELLO WORLD program and save it.

```
#include<iostream>
void main(){
    cout<<"Hello World!\n";
}
```

Compile the program

Install g++ Compiler from Synaptic manager or by writing the following commands in terminal.

```
sudo apt-get install g++
g++ hello.cpp
```

If there is no syntax/semantic error in your program then the compiler will successfully generate an executable file, otherwise fix the problem in your code.

Execute the program

To execute the program, you need to run –

```
./a.out
```

Task 3 You can convert temperature from degrees Celsius to degrees Fahrenheit by multiplying by 9/5 and adding 32. Write a program that allows the user to enter a floating-point number representing degrees Celsius, and then displays the corresponding degrees Fahrenheit.

Task 4 Write and run a program that simulates a simple calculator. It reads two integers and a character. If the character is a +, the sum is printed; if it is a -, the difference is printed; if it is a *, the product is printed; if it is a /, the quotient is printed; and if it is a %, the remainder is printed.

OPERATING SYSTEM LABORATORY MANUAL



UNIVERSITY OF THE PUNJAB

**FACULTY OF COMPUTING & INFORMATION TECHNOLOGY, LAHORE
DEPARTMENT OF COMPUTER SCIENCE**

Course:	Operating System Lab	Date:
Course Code:	CC-217-3L	Max Marks: 40
Faculty/Instructor's Name & Email:	Dr. Ahmad Hassan Butt (ahmad.hassan@pucit.edu.pk)	

**LAB MANUAL # 5
(SPRING 2023)**

Name: _____ Enroll No: _____

Objective(s) :

To study and execute the Linux Shell Programming.

Lab Tasks :

Task 1 : Write the output of the programs given.

Task 2: Write the output of the arithmetic programs provided in the lab.

Task 3 : Write the output of the following Control Structures.

Task 4 : Write a program to enter the numbers till the user wants and at the end it should display the maximum and minimum number entered.

Lab Grading Sheet :

Task	Max Marks	Obtained Marks	Comments(<i>if any</i>)
1.	10		
2.	10		
3.	10		
4.	10		
Total	40		Signature

Note : Attempt all tasks and get them checked by your Instructor

Lab 05: LINUX Shell Programming - I

Objective(s):

To understand VIM Editor.

Tool(s) used:

Ubuntu, VIM Editor

Introduction

Shell programming is a group of commands grouped together under single filename. After logging onto the system a prompt for input appears which is generated by a Command String Interpreter program called the shell. The shell interprets the input, takes appropriate action, and finally prompts for more input. The shell can be used either interactively - enter commands at the command prompt, or as an interpreter to execute a shell script. Shell scripts are dynamically interpreted, NOT compiled.

Common Shells**C-Shell - csh**

The default on teaching systems Good for interactive systems Inferior programmable features.

Bourne Shell

bash or sh - also restricted shell – bsh (The Bourne Again Shell) It was written by Steve Bourne. Over the years the original Bourne Shell has been expanded, but it remains the basic shell provided in many commercial versions of Linux.

Korn Shell

It was written by David Korn This shell extended many features of Bourne Again Shell and added many new features.

Thomas C-Shell - tcsh

The TC Shell performs the same functions as Bourne Again Shell. It is an interactive command line interpreter as well as for high level programming languages.

Shell Keywords

echo, read, if fi, else, case, esac, for , while , do , done, until , set, unset, readonly, shift, export, break, continue, exit, return, trap , wait, eval ,exec, ulimit , umask.

General Shell Terminologies

The shebang line or hashbang #!

The “shbang” line is the very first line of the script and lets the kernel know what shell will be interpreting the lines in the script. The shbang line consists of #! Followed by the full pathname to the shell, and can be followed by options to control the behavior of shell.

Example

```
#!/bin/bash
```

Comments

Comments are descriptive material preceded by a # sign. They are ineffect until the end of a line and can be started anywhere on the line.

Example

```
#This text is not interpreted by the shell.
```

FIRST “HELLO WORLD” SHELL PROGRAM

Step 1 Open VIM with the filename.sh extension.

Step 2 Write the Hello World Program

```
#!/bin/bash
#YOUR FIRST HELLO WORLD PROGRAM
echo 'Hello World!'
```

Step 3 Execution of SHELL Script:

By using change mode

command

```
$ chmod u +x hello.sh
$ ./hello.sh
```

Shell Variables

A variable is something to which we assign a value. The value assigned could be a number, text, or any data type.

Rules

- A variable name is any combination of alphabets, digits and an underscore.
- No commas or blanks are allowed within a variable name.
- The first character of a variable name must either be an alphabet or an underscore.
- Variable names should be of any reasonable length.

Shell Variables

1. **PATH** - Directory paths to search for commands.
2. **HOSTNAME** - The name of the computer.
3. **USER** - The user id of the user running this shell.
4. **SHELL** - The shell currently is used.
5. **TERM** - The type of terminal being used.
6. **PS1** - The prompt to print when the shell is ready for another command.

Task 1 Write the output of the below programs.

i) `#!/bin/bash
#Variable Assignment & Accessing
echo "Variable Name : "
Name="Operating System"
echo $Name`

OUTPUT

ECHO Statement

Similar to the output statement. To print output to the screen, the echo command is used.

Syntax: Echo “String” (or) echo \$ b (for variable).

Example: echo "What is your name?"

READ Statement

To get the input from the user.

Syntax: read x y (no need of commas between variables)

Reading user input: The read command takes a line of input from the user and assigns it to a variable(s) on the right-hand side. The read command can accept multiple variable names. Each variable will be assigned a word.

```
ii)#!/bin/bash
#Input from user
echo "Enter your name"
read NAME
echo "Enter your age"
read AGE
echo "Enter your enrollment"
read ENROLLMENT
echo "Hello $NAME, Your age is : $AGE Your enrollment is :
$ENROLLMENT"
```

OUTPUT

```
iii) #!/bin/bash
      #readonly Variables
      echo " Readonly Variables"
      Name="David"
      readonly Name
      Name='John'
```

OUTPUT

```
iv)   #!/bin/bash
      echo "Unset Variables : "
      Name="John"
      unset Name
      echo $Name
```

OUTPUT

Wildcards

There are some characters that are evaluated by the shell in a special way. They are called shell meta characters or “Wildcards. These characters are neither number nor letters.

Example

* , ?, [], \$

\$\$echo \$\$ -- It represents process ID Number, or PID of the current shell

The following table shows a number of special variables that can be used in shell scripts.

Variable	Description
\$0	The filename of the current script
\$n	These variables correspond to the arguments with which a script is invoked.
\$#	The number of arguments supplied to a script.
\$*	All arguments are double quoted. If a script receives two arguments \$* is equivalent to \$1,\$2
\$@	All arguments are individually double quoted, equivalent to \$1,\$2
\$\$	Shows the number of current shell.
\$!	The process number of last background command.

```
v) #!/bin/bash
#Unix Special Commands
echo "File Name = $0"
echo "First Parameter = $1"
echo "Second Parameter = $2"
echo "Quoted Values = $@"
echo "Quoted Values = $*"
echo "Total number of parameters = $#"
```

OUTPUT

```
vi) #!/bin/bash
#Special Commands
echo "File Name = $0"
echo "First Parameter = $1"
echo "Second Parameter = $2"
echo "Quoted Values = $@"
echo "Quoted Values = $*"
echo "Total number of parameters = $#"
echo $?
```

OUTPUT

EXPRESSION Command

Arithmetic Operations

To perform all arithmetic operations. The Bourne shell does not support arithmetic. LINUX/Linux commands must be used to perform calculations.

Task 2 Write the output of the following arithmetic programs

```
#!/bin/bash
var1=10
var2=20
add=$(expr $var1 + $var2)
sub=$(expr $var1 - $var2)
multi=$(expr $var1 \* $var2)
div=$(expr $var1 / $var2)
mod=$(expr $var1 % $var2)
echo "Addition : $add"
echo "Subtraction : $sub"
echo "Multiplication : $multi"
echo "Division : $div"
echo "Modulus : $mod"
```

OUTPUT

Operators

The Bourne shell uses the built-in test command operators to test numbers and strings.

Example

Equality:

=	<i>string</i>
!=	<i>string</i>
-eq	<i>number</i>
-ne	<i>number</i>

Logical:

-a	<i>and</i>
-o	<i>or</i>
!	<i>not</i>

Relational:

-gt	<i>greater than</i>
-ge	<i>greater than, equal to</i>
-lt	<i>less than</i>
-le	<i>less than, equal to</i>

Arithmetic:

+,-,*,/,%

Control Structures

Unix Shell supports conditional statements which are used to perform different actions based on different conditions. Two decision making statements are mentioned below:

- if...else statements
- case...esac statements

if..fi Statements

The if construct is followed by a command. If an expression is to be tested, it is enclosed in square brackets. The then keyword is placed after the closing parenthesis. An if must end with a fi.

Syntax:

```
if [ expression ]
```

```
then
    Statements to be executed if true
fi
```

Task 3 Write the output of the following Control Structures.

i)

```
#!/bin/bash
a=10
b=20
if [ $a == $b ]
then
    echo "a is equal to b"
fi
if [ $a != $b ]
then
    echo "a is not equal to b"
fi
```

OUTPUT

if..else..Statements

The if...else...fi statements is the next form of control statements that allow shell to execute statements in more controlled way and making decision in two ways.

Syntax

```
if [ expression ]
then
    Statement(s) to be executed if true
else
    Statement(s) to be executed if true

ii)      #!/bin/bash
a=10
b=20
if [ $a == $b ]
then
    echo "a is equal to b"
else
    echo "a is not equal to b"
fi
```

OUTPUT

if...else...elif...fi Statement

The if...elif...fi statement is to make correct decision out of several conditions.

Syntax

```
if [ expression 1 ]
then
    Statement(s) to be executed if expression 1 is true
elif [ expression 2]
    Statement(s) to be executed if expression 2 is true
elif [ expression 3]
    Statement(s) to be executed if expression 3 is true
else
    Statement(s) to be executed if no expression is true
fi
```

```
iii)  #!/bin/sh
a=10
b=20
if [ $a == $b ]
then
    echo "a is equal to b"
elif [ $a -gt $b ]
then
    echo "a is greater than b"
elif [ $a -lt $b]
then
    echo "a is less than b"
else
    echo "None of the conditions met."
fi
```

OUTPUT

case...esac Statement

You can handle multiple if...elif statements to perform a multiway branch. However, this is not the best solution, especially when all the branches depend the value of single variable.

Syntax

```
case word in
    pattern1)
        Statement(s) to be executed if pattern1 matches;;
    pattern2)
        Statement(s) to be executed if pattern1 matches;;
    pattern3)
        Statement(s) to be executed if pattern1 matches;;
esac

iv)  #!/bin/bash
     echo "Enter fruit name"
     read Fruit
     case "$Fruit" in
         "apple") echo "Apple pie";;
         "banana") echo "I like banana";;
         "kiwi") echo "New Zealand famous for kiwi";;
     esac
```

OUTPUT

LOOPS

There are three types of loops: while, until and for. The while loop is followed by a command or an expression enclosed in square brackets, a do keyword, a block of statements, and terminated with the done keyword. As long as the expression is true, the body of statements between do and done will be executed.

The until loop is just like the while loop, except the body of the loop will be executed as long as the expression is false.

The for loop used to iterate through a list of words, processing a word and then shifting it off, to process the next word. When all words have been shifted from the list, it ends. The for loop is followed by a variable name, the in keyword, and a list of words then a block of statements, and terminates with the done keyword.

The loop control commands are break and continue.

Syntax

While Loop

```
while command  
do  
    block of statements  
done
```

For Loop

```
for var in word1.....word  
do  
    Statement(s) to be executed for every word  
done
```

Until Loop

Until command
do
 Statement(s) to be executed until command is true
done

Example

```
for var in 0 1 2 3 4 5 6 7 8 9
do
    echo $var
done
```

Task 4 Write a program to enter the numbers till the user wants and at the end it should display the maximum and minimum number entered.

OPERATING SYSTEM LABORATORY MANUAL



UNIVERSITY OF THE PUNJAB

**FACULTY OF COMPUTING & INFORMATION TECHNOLOGY, LAHORE
DEPARTMENT OF COMPUTER SCIENCE**

Course:	Operating System Lab	Date:
Course Code:	CC-217-3L	Max Marks: 40
Faculty/Instructor's Name & Email:	Dr. Ahmad Hassan Butt (ahmad.hassan@pucit.edu.pk)	

**LAB MANUAL # 6
(SPRING 2023)**

Name: _____ Enroll No: _____

Objective(s) :

To study about the Linux Shell Programming. To write a shell program to compare/concatenate the two strings. To find greatest of three numbers and to perform the arithmetic operations using case.

Lab Tasks :

Task 1 : Write the output of the following array program.

Task 2: Write the output for concatenation of two strings.

Task 3 : Write the output of program for maximum of three numbers.

Task 4 : Write a program for implementing arithmetic operations using case.

Lab Grading Sheet :

Task	Max Marks	Obtained Marks	Comments(<i>if any</i>)
1.	10		
2.	10		
3.	10		
4.	10		
Total	40		Signature

Note : Attempt all tasks and get them checked by your Instructor

Lab 06 Shell Programming - II

Objective(s):

- To study about the Linux Shell Programming.
- To write a shell program to compare(concatenate) the two strings.
- To find greatest of three numbers and to perform the arithmetic operations using case.

Tool(s) used:

Ubuntu, VIM Editor

Arrays

Shell variable is capable enough to hold a single value. Shell supports a different type of variable called an array variable that can hold multiple values at the same time. Arrays provide a method of grouping a set of variables. Instead of creating a new name for each variable that is required, you can use a single array variable that stores all the other variables.

Variables are assigned as,

Name1 = “Zara”

Name2 = “Sarah”

Name3 = “Ali”

Name4 = “Ayesha”

We can use single array to store all the above mentioned names. This could be achieved by array

array_name[index]= value

Here *array_name* is the name of the array, *index* is the index of the item in the array that you want to set, and *value* is the value you want to set for that item.

Task 1 Write the output of the following array program.

```
#!/bin/bash
arr_name=("SARAH" "ALI" "AHMED")
echo "First Index : " ${arr_name[0]}
echo "Second Index : " ${arr_name[1]}
echo "Third Index : " ${arr_name[2]}
```

OUTPUT

Task 2 Write the output for concatenation of two strings.

Algorithm

Step 1 Enter into the vi editor and go to the insert mode for entering the code

Step 2 Read the first string.

Step 3 Read the second string

Step 4 Concatenate the two strings

Step 5 Enter into the escape mode for the execution of the result and verify the output.

Program

```
echo "enter the first
string" read str1
echo "enter the second
string" read str2
echo "the concatenated string is" $str1$str2
```

OUTPUT

Task 2.1 Write the output for the comparison of two strings.

Algorithm

Step 1 Enter into the vi editor and go to the insert mode for entering the code

Step 2 Read the first string.

Step 3 Read the second string

Step 4 Compare the two strings using the if loop

Step 5 If the condition satisfies then print that two strings are equal else print two strings are not equal.

Step 6 Enter into the escape mode for the execution of the result and verify the output

Program

```
echo "enter the first string" read str1
echo "enter the second string" read str2
if [ $str1 = $str2 ] then
    echo "strings are equal" else
    echo "strings are unequal" fi
```

OUTPUT

Task 3 Write the output of program for maximum of three numbers.

Algorithm

Step 1 Declare the three variables.

Step 2 Check if A is greater than B and C.

Step 3 If so print A is greater.

Step 4 Else check if B is greater than C.

Step 5 If so print B is greater.

Step 6 Else print C is greater.

Program

```
echo "enter A" read a
echo "enter B" read b
echo "enter C" read c
if [ $a -gt $b -a $a -gt $c ] then
    echo "A is greater"
elif [ $b -gt $a -a $b -gt $c ] then
    echo "B is greater" else
    echo "C is greater" fi
```

Sample I/P

Sample O/P

Task 4 Write a program for implementing arithmetic operations using case.

Algorithm

Step 1 Read the input variables and assign the value

Step 2 Print the various arithmetic operations which we are going to perform

Step 3 Using the case operator assign the various functions for the arithmetic operators.

Step 4 Check the values for all the corresponding operations.

Step 5 Print the result and stop the execution.

OUTPUT

OPERATING SYSTEM LABORATORY MANUAL



UNIVERSITY OF THE PUNJAB

**FACULTY OF COMPUTING & INFORMATION TECHNOLOGY, LAHORE
DEPARTMENT OF COMPUTER SCIENCE**

Course:	Operating System Lab	Date:
Course Code:	CC-217-3L	Max Marks: 40
Faculty/Instructor's Name & Email:	Dr. Ahmad Hassan Butt (ahmad.hassan@pucit.edu.pk)	

**LAB MANUAL # 7
(SPRING 2023)**

Name: _____ Enroll No: _____

Objective(s) :

To write a C program to implement the CPU scheduling algorithm for FIRST COME FIRST SERVE. To write a C program to implement the CPU scheduling algorithm for Shortest job first.

Lab Tasks :

Task 1 : Calculate the Average Time using FCFS Algorithm.

Task 2: Write the output of the program for First Come First Serve.

Task 3 : Calculate the Average Time using SJF Algorithm.

Task 4 : Write the output a program for Shortest Job First.

Lab Grading Sheet :

Task	Max Marks	Obtained Marks	Comments(<i>if any</i>)
1.	10		
2.	10		
3.	10		
4.	10		
Total	40		Signature

Note : Attempt all tasks and get them checked by your Instructor

Lab 7: Scheduling

Objective(s):

- To write a C program to implement the CPU scheduling algorithm for FIRST COME FIRST SERVE.
- To write a C program to implement the CPU scheduling algorithm for Shortest job first.

Tool(s) used:

Ubuntu, VIM Editor

First Come First Serve

CPU scheduler will decide which process should be given to the CPU for its execution. For this it uses different algorithms to choose among the process. One among that algorithm is FCFS algorithm. In this algorithm, the process which arrive first is given to the CPU after finishing its request only it will allow CPU to execute other process.

Task 1 : Calculate the Average Time using FCFS Algorithm.

Process	Duration	Order	Arrival Time
P1	24	1	0
P2	3	2	0
P3	4	3	0

Task 2 : Write the output of the program for First Come First Serve.

Step 1 : Create the number of process.

Step 2 : Get the ID and Service time for each process.

Step 3 : Initially, Waiting time of first process is zero and Total time for the first process is the starting time of that process.

Step 4 : Calculate the Total time and Processing time for the remaining processes.

Step 5 : Waiting time of one process is the Total time of the previous process.

Step 6 : Total time of process is calculated by adding Waiting time and Service time.

Step 7 : Total waiting time is calculated by adding the waiting time for each process.

Step 8 : Total turnaround time is calculated by adding all total time of each process.

Step 9 : Calculate Average waiting time by dividing the total waiting time by total number of process.

Step 10 : Calculate Average turnaround time by dividing the total time by the number of process.

Step 11 : Display the result.

Program

```

#include<stdio.h>
struct process{
    int id,wait,ser,tottime;
}p[20];

int main(){

    int i,n,j,totalwait=0,totalser=0,avwait;
    printf("enter number of process");
    scanf("%d",&n);

    for(i=1;i<=n;i++){
        printf("enter process_id");
        scanf("%d",&p[i].id);
        printf("enter process service time");
        scanf("%d",&p[i].ser);
    }
    p[1].wait=0;
    p[1].tottime=p[1].ser;

    for(i=2;i<=n;i++){
        for(j=1;j<i;j++){
            p[i].wait=p[i].wait+p[j].ser;
        }

        totalwait=totalwait+p[i].wait;
        p[i].tottime=p[i].wait+p[i].ser;
        totalser=totalser+p[i].tottime;
    }

    avwait=totalwait/n;
    printf("Id\tservice\twait\ttotal");

    for(i=1;i<=n;i++){

        printf("\n%d\t%d\t%d\t%d\n",p[i].id,p[i].ser,p[i].wait,p[i].tottime);
    }

    printf("average waiting time %d\n",avwait);

    return 0;
}

```

OUTPUT

Shortest Job First

CPU scheduler will decide which process should be given to the CPU for its execution. For this it uses different algorithms to choose among the processes. One among that algorithm is Shortest Job First. In this algorithm the process which has less service time given the CPU after finishing its request only it will allow CPU to execute next other process.

Task 3 : Calculate the Average Time using SJF Algorithm.

Process	Duration	Order	Arrival Time
P1	6	1	0
P2	8	2	0
P3	7	3	0
P4	3	4	0

Task 4 Write the output a program for Shortest Job First.

Step 1 : Get the number of process.

Step 2 : Get the id and service time for each process.

Step 3 : Initially the waiting time of first short process as 0 and total time of first short is process the service time of that process.

Step 4 : Calculate the total time and waiting time of remaining process.

Step 5 : Waiting time of one process is the total time of the previous process.

Step 6 : Total time of process is calculated by adding the waiting time and service time of each process.

Step 7 : Total waiting time calculated by adding the waiting time of each process.

Step 8 : Total turnaround time calculated by adding all total time of each process.

Step 9 : Calculate average waiting time by dividing the total waiting time by total number of process.

Step 10 : Calculate average turnaround time by dividing the total waiting time by total number of process.

Step 11 : Display the result.

Program

```

#include<stdio.h>
struct ff{
    int pid,ser,wait;
}p[20];

struct ff tmp;
int main(){
    int i,n,j,tot=0,await,totwait=0,tturn=0,aturn;
    printf("Enter the number of process");
    scanf("%d",&n);
    for(i=0;i<n;i++){
        printf("Enter process id");
        scanf("%d",&p[i]);
        printf("Enter service time");
        scanf("%d",&p[i].ser);
        p[i].wait=0;
    }

    for(i=0;i<n-1;i++){
        for(j=i+1;j<n;j++){
            if(p[i].ser>p[j].ser){
                tmp=p[i];
                p[i]=p[j];
                p[j]=tmp;
            }
        }
    }

    printf("PID\tSER\tWAIT\tTOT\n");

    for(i=0;i<n;i++){
        tot=tot+p[i].ser;
        p[i+1].wait=tot;
    }
}

```

```
totwait=totwait+p[i].wait;

printf("%d\t%d\t%d\t%d\n",p[i].pid,p[i].ser,p[i].wait,tot);
}

avwait=totwait/n;

printf("TOTAL WAITING TIME:%d\n",totwait);
printf("AVERAGE WAITING TIME: %d\n",avwait);

return 0;
}
```

OUTPUT

OPERATING SYSTEM LABORATORY MANUAL



UNIVERSITY OF THE PUNJAB

**FACULTY OF COMPUTING & INFORMATION TECHNOLOGY, LAHORE
DEPARTMENT OF COMPUTER SCIENCE**

Course:	Operating System Lab	Date:
Course Code:	CC-217-3L	Max Marks: 40
Faculty/Instructor's Name & Email:	Dr. Ahmad Hassan Butt (ahmad.hassan@pucit.edu.pk)	

**LAB MANUAL # 8
(SPRING 2023)**

Name: _____ Enroll No: _____

Objective(s) :

To write a C program to implement CPU scheduling algorithm for Priority Scheduling.

Lab Tasks :

Task 01: Calculate the Average Time using Priority Scheduling.

Task 02: Write the algorithm for Priority Scheduling Algorithm.

Task 03 + 04: Write the output for program of Priority Scheduling.

Lab Grading Sheet :

Task	Max Marks	Obtained Marks	Comments(<i>if any</i>)
1.	10		
2.	10		
3.	10		
4.	10		
Total	40		Signature

Note : Attempt all tasks and get them checked by your Instructor

Lab 08: Priority Scheduling

Objective(s):

To write a C program to implement CPU scheduling algorithm for Priority Scheduling.

Tool(s) used:

Ubuntu, VIM Editor

CPU scheduler will decide which process should be given the CPU for its execution. For this it uses different algorithm to choose among the processes. One among that algorithm is FCFS algorithm. In this algorithm the process which arrives first is given the CPU after finishing its request only it will allow CPU to execute other process. In priority scheduling algorithm each process has a priority associated with it and as each process hits the queue, it is stored in based on its priority so that process with higher priority are dealt with first. It should be noted that equal priority processes are scheduled in FCFS order.

Task 01: Calculate the Average Time using Priority Scheduling.

Process	CPU Burst Time	Priority
P1	9	5
P2	4	3
P3	5	1
P4	7	2
P5	3	4
Total	28	

Task 02: Write the algorithm for Priority Scheduling Algorithm.

Task 03+04: Write the output for program of Priority Scheduling.

```
#include <stdio.h>
int main()
{
    int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
    printf("Enter Total Number of Process:");
    scanf("%d",&n);
    printf("\nEnter Burst Time and Priority\n");
    for(i=0;i<n;i++)
    {
        printf("\nP[%d]\n",i+1);
        printf("Burst Time:");
        scanf("%d",&bt[i]);
        printf("Priority:");
        scanf("%d",&pr[i]);
        p[i]=i+1;           //contains process number
    }
    //sorting burst time, priority and process number in ascending order
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++){
            if(pr[j]<pr[pos])
                pos=j;
        }
        temp=pr[i];
        pr[i]=pr[pos];
        pr[pos]=temp;
        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;
        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }
    wt[0]=0;      //waiting time for first process is zero
```

```
//calculate waiting time
for(i=1;i<n;i++){
    wt[i]=0;
    for(j=0;j<i;j++)
        wt[i]+=bt[j];
    total+=wt[i];
}
avg_wt=total/n;      //average waiting time
printf("\nProcess\t\t Burst Time\t\t Waiting Time ");
for(i=0;i<n;i++){
    printf("\nP[%d]\t\t %d\t\t %d",p[i],bt[i],wt[i]);
}
printf("\n\nAverage Waiting Time=%d",avg_wt);

return 0;
}
```

OUTPUT

OPERATING SYSTEM LABORATORY MANUAL



UNIVERSITY OF THE PUNJAB

**FACULTY OF COMPUTING & INFORMATION TECHNOLOGY, LAHORE
DEPARTMENT OF COMPUTER SCIENCE**

Course:	Operating System Lab	Date:
Course Code:	CC-217-3L	Max Marks: 40
Faculty/Instructor's Name & Email:	Dr. Ahmad Hassan Butt (ahmad.hassan@pucit.edu.pk)	

**LAB MANUAL # 9
(SPRING 2023)**

Name: _____ Enroll No: _____

Objective(s) :

To write a program to create a process in LINUX. To create child with sleep command. To understand getpid() and getppid().

Lab Tasks :

Task 1 : Write the output of program for process creation using fork command.

Task 2: Write the output of a program for execution of ls command using exec.

Task 3 : Write the output of a program illustrating the sleep command during process creation.

Task 4 : Write the output of the program for getting the pid and ppid while using the sleep command.

Lab Grading Sheet :

Task	Max Marks	Obtained Marks	Comments(<i>if any</i>)
1.	10		
2.	10		
3.	10		
4.	10		
Total	40		Signature

Note : Attempt all tasks and get them checked by your Instructor

Lab 09: Processes

Objective(s):

- To write a program to create a process in LINUX.
- To create child with sleep command.
- To understand getpid() and getppid().

Tool(s) used:

Ubuntu, VIM Editor

Task 1 Write the output of a program for process creation using fork command.

Algorithm

STEP 1: Start the program.

STEP 2: Declare pid as integer.

STEP 3: Create the process using Fork command.

STEP 4: Check pid is less than 0 then print error else if pid is equal to 0 then execute command else parent process wait for child process.

STEP 5: Stop the program.

Program

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(){
    int id;
    id=fork();
    if(id<0){
        printf ("Cannot Create the file");
        exit(-1);
    }
    if(id==0){
        printf ("Child Process");
```

```
        exit(0);
    }
else{
    printf ("Parent Process");
    exit(1);
}
return 0;
}
```

Program Execution

```
$gcc pc.c -o pc  
$./pc
```

OUTPUT

Task 2 Write the output of a program for execution of ls command using exec.

Algorithm

STEP 1: Start the program.

STEP 2: Execute the command in the shell program using exec ls.

STEP 3: Stop the execution.

Program

```
echo Program for executing LINUX command using Shell Programming
echo Welcome
ps
exec ls
```

OUTPUT

Task 3 Write the output of a program illustrating the sleep command during process creation.

Algorithm

STEP 1: Start the program.

STEP 2: Create process using fork and assign into a variable.

STEP 3: If the value of variable is < zero print not create and > 0 process create and else print child create.

STEP 4: Create child with sleep of 2.

STEP 5: Stop the program.

Program

```
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
int main( ){
    pid_t id;
    id=fork( );
    if (id== -1){
        printf ("Cannot Create the file");
        exit(1);
    }
    if (id==0){
        sleep(20);
        printf ("This is child Process");
    }
    else{
        printf ("Parent Process");
        exit(1);
    }
    return 0;
}
```

OUTPUT

- Task 4** Write the output of the program for getting the pid and ppid while using the sleep command.

Algorithm

STEP 1: Start the execution and create a process using fork() command.

STEP 2: Make the parent process to sleep for 10 seconds.

STEP 3: In the child process print it pid and its corresponding pid.

STEP 4: Make the child process to sleep for 5 seconds.

STEP 5: Again print its pid and its parent pid.

STEP 6: After making the sleep for the parent process for 10 seconds print its pid.

STEP 7: Stop the execution.

Program

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(){
    pid_t pid;
    pid=fork();
    if(pid==0){
        printf("\nChild Process");
        printf("\nChild Process ID is %d", getpid());
        printf("\nIts Parent Process ID is %d", getppid());
        sleep(5);
        printf("\nChild Process after sleep=5");
        printf("\nChild Process ID is %d", getpid());
        printf("\nParent Process ID is %d", getppid());
    }
    else{
        printf("\n\nParent Process\n");
        sleep(5);
    }
}
```

```
    printf("\nChild Process ID is %d", getpid());
    printf("\nIts Parent Process ID is %d", getppid());
    printf("\nParent Terminates\n");
}
return 0;
}
```

OUTPUT

OPERATING SYSTEM LABORATORY MANUAL



UNIVERSITY OF THE PUNJAB

**FACULTY OF COMPUTING & INFORMATION TECHNOLOGY, LAHORE
DEPARTMENT OF COMPUTER SCIENCE**

Course:	Operating System Lab	Date:
Course Code:	CC-217-3L	Max Marks: 40
Faculty/Instructor's Name & Email:	Dr. Ahmad Hassan Butt (ahmad.hassan@pucit.edu.pk)	

**LAB MANUAL # 10
(SPRING 2023)**

Name: _____ Enroll No: _____

Objective(s) :

To write a program for signal handling in LINUX. To use wait command using c program. To use wait command using c program and creating a separate process using exec(). To create a Zombie Process.

Lab Tasks :

Task 1 : Write the output of the program illustrating the Kill Command.

Task 2: Write the output of the program implementing the Wait signal.

Task 3 : Write the output of the program for wait signal using exec().

Task 4 : Write the output of the program for a Zombie Process.

Lab Grading Sheet :

Task	Max Marks	Obtained Marks	Comments(<i>if any</i>)
1.	10		
2.	10		
3.	10		
4.	10		
Total	40		Signature

Note : Attempt all tasks and get them checked by your Instructor

Lab 10: Signal Handling

Objective(s):

- To write a program for signal handling in LINUX,
- To use wait command using c program,
- To use wait command using c program and creating a separate process using exec()
- To create a Zombie Process.

Tool(s) used:

Ubuntu, VIM Editor

Task 1 Write the output of the program illustrating the Kill Command.

Algorithm

STEP 1: Start the program

STEP 2: Read the value of pid.

STEP 3: Kill the command surely using kill-9 pid.

STEP 4: Stop the program.

Program

```
echo program for performing KILL operations
ps

echo enter the pid
read pid

kill -9 $pid
```

OUTPUT

Task 2 Write the output of the program implementing the Wait signal.

Algorithm

STEP 1: Start the execution

STEP 2: Create process using fork and assign it to a variable

STEP 3: Check for the condition pid is equal to 0

STEP 4: If it is true print the value of i and terminate the child process

STEP 5: If it is not a parent process has to wait until the child terminate

STEP 6: Stop the execution

Program

```
main( ){
    pid_t pid;
    int i=10;
    pid=fork();
    if(pid==0){
        printf("initial value of i %d \n ",i);
        i+=10;
        printf("value of i %d\n ",i);
        printf("child terminated \n");
    }
    else{
        wait(NULL);
        printf("value of i in parent process %d",i);
    }
    return 0;
}
```

OUTPUT

Task 3 Write the output of the program for a Zombie Process.

Algorithm

STEP 1: Start the execution

STEP 2: Create process using fork and assign it to a variable

STEP 3: Check for the condition pid is equal to 0

STEP 4: If it is true print Child Process

STEP 5: If it is not a parent process has to wait until the child terminates

STEP 6: Parent process should print Parent Process.

STEP 7: Stop the execution

Program

```
int main( ){
    pid_t pid;
    pid=fork( );
    if(pid==0){
        printf("Child Process");
    }
    else{
        sleep(10);
        wait(NULL);
        printf("Parent Process");
    }
    return 0; }
```

OUTPUT

Task 4 Write the output of the program for wait signal using exec()

Algorithm

STEP 1: Start the execution

STEP 2: Create process using fork and assign it to a variable

STEP 3: Check for the condition pid is equal to 0

STEP 4: If it is true print the commands to be executed by ls.

STEP 5: If it is not a parent process has to wait until the child terminates

STEP 6: Parent process should print Child Process completed.

STEP 7: Stop the execution

Program

```
int main( ){
    pid_t pid;
    if(pid<0){
        fprintf(stderr, "Fork Failed");
        return 1;
    } else if(pid ==0){
        execlp("/bin/ls", "ls",NULL);
    } else{
        wait(NULL);
        printf("Child Completes");
    }
    return 0;
}
```

OUTPUT

OPERATING SYSTEM LABORATORY MANUAL



UNIVERSITY OF THE PUNJAB

**FACULTY OF COMPUTING & INFORMATION TECHNOLOGY, LAHORE
DEPARTMENT OF COMPUTER SCIENCE**

Course:	Operating System Lab	Date:
Course Code:	CC-217-3L	Max Marks: 40
Faculty/Instructor's Name & Email:	Dr. Ahmad Hassan Butt (ahmad.hassan@pucit.edu.pk)	

**LAB MANUAL # 11
(SPRING 2023)**

Name: _____ Enroll No: _____

Objective(s) :

To understand Mutex and Thread Synchronization.

Lab Tasks :

Task 1 : Write the output of Hello World program using Threads.

Task 2: Write the output of program for Mutex Hello World.

Task 3 Identify the output error in Thread Synchronization Problem.

Task 4 : Write the output for Mutex for Thread Synchronization.

Lab Grading Sheet :

Task	Max Marks	Obtained Marks	Comments(<i>if any</i>)
1.	10		
2.	10		
3.	10		
4.	10		
Total	40		Signature

Note : Attempt all tasks and get them checked by your Instructor

Lab 11: Thread Synchronization

Objective(s):

To write a C program to understand Mutex and Synchronization.

Tool(s) used:

Ubuntu, VIM Editor

Thread synchronization is defined as a mechanism which ensures that two or more concurrent processes or threads do not simultaneously execute some particular program segment known as critical section. Processes' access to critical section is controlled by using synchronization techniques. When one thread starts executing the critical section (serialized segment of the program) the other thread should wait until the first thread finishes. If proper synchronization techniques are not applied, it may cause a race condition where the values of variables may be unpredictable and vary depending on the timings of context switches of the processes or threads.

Task 01: Write the output of Hello World program using Threads.

```
#include <pthread.h>
#include <stdio.h>
void* compute_thread (void*);  
  
int main( ){
    pthread_t tid;
    pthread_attr_t attr;
    char hello[ ] = {"Hello, "};
    char thread[ ] = {"thread"};
    pthread_attr_init(&attr);
    pthread_create(&tid, &attr, compute_thread, thread);
    printf(hello);
    sleep(1);
    printf("\n");
    exit(0);
}
void* compute_thread(void* dummy){
    printf(dummy);
    return 0;}  

```

OUTPUT:

Task 02: Write the output of program for Mutex Hello World.

```
#include <pthread.h>
#include <stdio.h>
void* compute_thread (void* );
pthread_mutex_t my_sync;

main( ){
    pthread_t tid;
    pthread_attr_t attr;
    char hello[ ] = {"Hello, "};
    char thread[ ] = {"thread"};
    pthread_attr_init (&attr);
    pthread_mutex_init (&my_sync,NULL);
    pthread_create(&tid, &attr, compute_thread, hello);
    sleep(1);
    pthread_mutex_lock(&my_sync);
    printf(thread);
    printf("\n");
    pthread_mutex_unlock(&my_sync);
    exit(0);}

void* compute_thread(void* dummy){
    pthread_mutex_lock(&my_sync);
    printf(dummy);
    pthread_mutex_unlock(&my_sync);
    sleep(1);

return;
}
```

OUTPUT

Task 03: Identify the output error in Thread Synchronization Problem.

```
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
pthread_t tid[2];
int counter;
void* trythis(void *arg){
    unsigned long i = 0;
    counter += 1;
    printf("\n Job %d has started\n", counter);
    for(i=0; i<(0xFFFFFFFF);i++);
    printf("\n Job %d has finished\n", counter);
    return NULL;
}

int main(void){
    int i = 0;
    int error;
    while(i < 2){
        error = pthread_create(&(tid[i]), NULL, &trythis, NULL);
        if (error != 0)
            printf("\nThread can't be created : [%s]", strerror(error));
        i++;
    }
    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);
    return 0;
}
```

OUTPUT

Task 04: Write the output for Mutex for Thread Synchronization.

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

pthread_t tid[2];
int counter;
pthread_mutex_t lock;
void* trythis(void *arg){
    pthread_mutex_lock(&lock);
    unsigned long i = 0;
    counter += 1;
    printf("\n Job %d has started\n", counter);
    for(i=0; i<(0xFFFFFFFF);i++);
    printf("\n Job %d has finished\n", counter);
    pthread_mutex_unlock(&lock);
    return NULL;
}

int main(void){
    int i = 0;
    int error;
    if (pthread_mutex_init(&lock, NULL) != 0){
        printf("\n mutex init has failed\n");
        return 1;
    }
    while(i < 2) {
        error = pthread_create(&(tid[i]), NULL, &trythis, NULL);
        if (error != 0)
            printf("\nThread can't be created :[%s]", strerror(error));
        i++;
    }
    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);
    pthread_mutex_destroy(&lock);
    return 0;
}
```

OUTPUT:

OPERATING SYSTEM LABORATORY MANUAL



UNIVERSITY OF THE PUNJAB

**FACULTY OF COMPUTING & INFORMATION TECHNOLOGY, LAHORE
DEPARTMENT OF COMPUTER SCIENCE**

Course:	Operating System Lab	Date:
Course Code:	CC-217-3L	Max Marks: 40
Faculty/Instructor's Name & Email:	Dr. Ahmad Hassan Butt (ahmad.hassan@pucit.edu.pk)	

**LAB MANUAL # 12
(SPRING 2023)**

Name: _____ Enroll No: _____

Objective(s) :

To implement Producer & Consumer Problem (Semaphore).

Lab Tasks :

Task 01: Write the code to initialize Semaphore Mutex.

Task 02: Write the code according the requirement if it's a consumer.

Task 03: Write the code according to the requirement if it's a producer.

Task 04: Display the result.

Lab Grading Sheet :

Task	Max Marks	Obtained Marks	Comments(<i>if any</i>)
1.	10		
2.	10		
3.	10		
4.	10		
Total	40		Signature

Note : Attempt all tasks and get them checked by your Instructor

Lab 12: Semaphore

Objective(s):

To implement Producer & Consumer Problem (Semaphore)

Tool(s) used:

Ubuntu, VIM Editor

A **semaphore**, in its most basic form, is a protected integer variable that can facilitate and restrict access to shared resources in a multi-processing environment. The producer-consumer problem is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer used as a queue. The producer's job is to generate data, put it into the buffer, and start again. At the same time, the consumer is consuming the data (i.e., removing it from the buffer), one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

Task 01: Write the code for the Semaphore mutex, full & empty initialized.

Task 02: Write the code in the case of producer process.

- i) Produce an item in to temporary variable.
- ii) If there is empty space in the buffer check the mutex value for enter into the critical section.
- iii) If the mutex value is 0, allow the producer to add value in the temporary variable to the buffer.

Task 03: Write the code in the case of consumer process.

- i) It should wait if the buffer is empty
- ii) If there is any item in the buffer check for mutex value, if the mutex==0, remove item from buffer
- iii) Signal the mutex value and reduce the empty value by 1.
- iv) Consume the item.

Task 04: Print the result.

Program

```
#define BUFFERSIZE 10
int
mutex,n,empty,full=0,item,item
1;
int buffer[20];
int
in=0,out=0,mutex=1;
void wait(int s)
{
    while(s<0)
    {
        printf("\nCannot add an
item\n"); exit(0);
    }
    s--;
}
void signal(int s)
{
    s++;
}
void producer( )
{
    Do
    {
        wait (empty);
        wait (mutex);
        printf("\nEnter an
item:");
    }
}
```

```
scanf("%d", &item);
buffer[in]=item;
in=in+1;
signal(mutex);
signal(full);
}

while(in<n);
}

void consumer( )
{
Do
{
    wait(full);
    wait(mutex);
    item1=buffer[out]; printf("\nConsumed item
=%d", item1);
    out=out+1;
    signal(mutex);
    signal(empty);
}
while(out<n);
}

void main( )
{
    printf("Enter the value of n:");
    scanf("%d ", &n);
    empty=n;
    while(in<n)
        producer( );
}
```

```
while(in!=out)
    consumer();
}
```

OUTPUT

OPERATING SYSTEM LABORATORY MANUAL



UNIVERSITY OF THE PUNJAB

**FACULTY OF COMPUTING & INFORMATION TECHNOLOGY, LAHORE
DEPARTMENT OF COMPUTER SCIENCE**

Course:	Operating System Lab	Date:
Course Code:	CC-217-3L	Max Marks: 40
Faculty/Instructor's Name & Email:	Dr. Ahmad Hassan Butt (ahmad.hassan@pucit.edu.pk)	

**LAB MANUAL # 13
(SPRING 2023)**

Name: _____ Enroll No: _____

Objective(s) :

To understand memory management using paging.

Lab Tasks :

Task 01: Explain memory management.

Task 02: Explain the process of paging.

Task 03: Differentiate between logical and physical address.

Task 04: Write the output of the program given in Lab Manual.

Lab Grading Sheet :

Task	Max Marks	Obtained Marks	Comments(<i>if any</i>)
1.	10		
2.	10		
3.	10		
4.	10		
Total	40		Signature

Note : Attempt all tasks and get them checked by your Instructor

Lab 13: Memory Management Scheme-Paging

Objective(s):

To write a C program to implement memory management using paging technique.

Tool(s) used:

Ubuntu, VIM Editor

Paging:

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non – contiguous.

- Logical Address or Virtual Address (represented in bits): An address generated by the CPU
- Logical Address Space or Virtual Address Space(represented in words or bytes): The set of all logical addresses generated by a program
- Physical Address (represented in bits): An address actually available on memory unit
- Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses

Read the base address, page size, number of pages and memory unit. If the memory limit is less than the base address display the memory limit is less than limit. Create the page table with the number of pages and page address. Read the page number and displacement value. If the page number and displacement value is valid, add the displacement value with the address corresponding to the page number and display the result. Display the page is not found or displacement should be less than page size. Stop the program.

PROGRAM

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

    int b[20],n,i,pa,p,a,d;
    printf("\nProgram for
paging"); scanf("%d",&n);
    printf("\nEnter the base
address:");
    for(i=0;i<n;i++){
        scanf("%d",&b[i]);
    }
    printf("\nEnter the logical
address:"); scanf("%d",&p);
    for(i=0;i<n;i++){
        if(i==p){
            pa=b[i]+d;
            a=b[i];
            printf("\n\tPageNo.\t BaseAdd. PhysicalAdd. \n\t %d \t
            %d \t %d \t
            ",p,a,pa);
        }
    }
    printf("\nInvalid page");
}
```

Sample Input 1

Sample Output 1

Sample Input 2

Sample Output 2

OPERATING SYSTEM LABORATORY MANUAL



UNIVERSITY OF THE PUNJAB

**FACULTY OF COMPUTING & INFORMATION TECHNOLOGY, LAHORE
DEPARTMENT OF COMPUTER SCIENCE**

Course:	Operating System Lab	Date:
Course Code:	CC-217-3L	Max Marks: 40
Faculty/Instructor's Name & Email:	Dr. Ahmad Hassan Butt (ahmad.hassan@pucit.edu.pk)	

**LAB MANUAL # 14
(SPRING 2023)**

Name: _____ Enroll No: _____

Objective(s) :

To understand concepts of Deadlocks and Banker's Algorithm.

Lab Tasks :

Task 1: What is a deadlock and Banker's Algorithm?

Task 2: Which data structures is being used in Banker's Algorithm?

Task 3 & 4: Write and analyze the program to illustrate the Banker's Algorithm.

Lab Grading Sheet :

Task	Max Marks	Obtained Marks	Comments(<i>if any</i>)
1.	10		
2.	10		
3.	10		
4.	10		
Total	40		Signature

Note : Attempt all tasks and get them checked by your Instructor

Lab 14: Deadlocks and Banker's Algorithm

Objective(s):

To understand concepts of Deadlocks and Banker's Algorithm.

Task 1: What is a deadlock and Banker's Algorithm?

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

Task 2: Which data structures is being used in Banker's Algorithm?

Following **Data structures** are used to implement the Banker's Algorithm:

Let '**n**' be the number of processes in the system and '**m**' be the number of resources types.

Available:

- It is a 1-d array of size '**m**' indicating the number of available resources of each type.
- Available [**j**] = **k** means there are '**k**' instances of resource type R_j

Max:

- It is a 2-d array of size '**n*m**' that defines the maximum demand of each process in a system.
- Max[**i, j**] = **k** means process P_i may request at most '**k**' instances of resource type R_j .

Allocation:

- It is a 2-d array of size '**n*m**' that defines the number of resources of each type currently allocated to each process.
- Allocation[**i, j**] = **k** means process P_i is currently allocated '**k**' instances of resource type R_j

Need:

- It is a 2-d array of size '**n*m**' that indicates the remaining resource need of each process.
- Need [**i, j**] = **k** means process P_i currently allocated '**k**' instances of resource type R_j
- Need [**i, j**] = Max [**i, j**] – Allocation [**i, j**]

Allocation_i specifies the resources currently allocated to process P_i and **Need_i** specifies the additional resources that process P_i may still request to complete its task.

Task 3 & 4: Write and analyze the program to illustrate the Banker's Algorithm.

PROGRAM:

```
// C++ program to illustrate Banker's Algorithm

#include <iostream>

using namespace std;

// Number of processes
const int P = 5;

// Number of resources
const int R = 3;

// Function to find the need of each process
void calculateNeed(int need[P][R], int maxm[P][R],
                    int allot[P][R])
{
    // Calculating Need of each P
    for (int i = 0 ; i < P ; i++)
        for (int j = 0 ; j < R ; j++)
            // Need of instance = maxm instance -
            //                     allocated instance
            need[i][j] = maxm[i][j] - allot[i][j];
}
```

```
}

// Function to find the system is in safe state or not
bool isSafe(int processes[], int avail[], int maxm[][][R],
            int allot[][][R])

{
    int need[P][R];

    // Function to calculate need matrix
    calculateNeed(need, maxm, allot);

    // Mark all processes as infinish
    bool finish[P] = {0};

    // To store safe sequence
    int safeSeq[P];

    // Make a copy of available resources
    int work[R];
    for (int i = 0; i < R; i++)
        work[i] = avail[i];

    // While all processes are not finished
    // or system is not in safe state.

    int count = 0;
```

```
while (count < P)

{
    // Find a process which is not finish and
    // whose needs can be satisfied with current
    // work[] resources.

    bool found = false;

    for (int p = 0; p < P; p++)

    {
        // First check if a process is finished,
        // if no, go for next condition

        if (finish[p] == 0)

        {
            // Check if for all resources of
            // current P need is less
            // than work

            int j;

            for (j = 0; j < R; j++)

                if (need[p][j] > work[j])

                    break;

            // If all needs of p were satisfied.

            if (j == R)

            {
                // Add the allocated resources of
```

```
// current P to the available/work  
  
// resources i.e. free the resources  
  
for (int k = 0 ; k < R ; k++)  
  
    work[k] += allot[p][k];  
  
  
// Add this process to safe sequence.  
  
safeSeq[count++] = p;  
  
  
// Mark this p as finished  
  
finish[p] = 1;  
  
  
found = true;  
  
}  
  
}  
  
}  
  
// If we could not find a next process in safe  
  
// sequence.  
  
if (found == false)  
  
{  
  
    cout << "System is not in safe state";  
  
    return false;  
  
}  
  
}
```

```
// If system is in safe state then  
  
// safe sequence will be as below  
  
cout << "System is in safe state.\nSafe"  
      " sequence is: ";  
  
for (int i = 0; i < P ; i++)  
  
    cout << safeSeq[i] << " ";  
  
  
return true;  
}  
  
// Driver code  
  
int main()  
  
{  
  
    int processes[] = {0, 1, 2, 3, 4};  
  
  
    // Available instances of resources  
  
    int avail[] = {3, 3, 2};  
  
  
    // Maximum R that can be allocated  
    // to processes  
  
    int maxm[][][R] = {{7, 5, 3},  
                      {3, 2, 2},  
                      {9, 0, 2},  
                      {2, 2, 2},
```

```
{ 4, 3, 3 } } ;  
  
// Resources allocated to processes  
  
int allot[][][R] = {{0, 1, 0},  
                    {2, 0, 0},  
                    {3, 0, 2},  
                    {2, 1, 1},  
                    {0, 0, 2}};  
  
// Check system is in safe state or not  
isSafe(processes, avail, maxm, allot);  
  
return 0;  
}
```

OPERATING SYSTEM LABORATORY MANUAL



UNIVERSITY OF THE PUNJAB

**FACULTY OF COMPUTING & INFORMATION TECHNOLOGY, LAHORE
DEPARTMENT OF COMPUTER SCIENCE**

Course:	Operating System Lab	Date:
Course Code:	CC-217-3L	Max Marks: 40
Faculty/Instructor's Name & Email:	Dr. Ahmad Hassan Butt (ahmad.hassan@pucit.edu.pk)	

**LAB MANUAL # 15
(SPRING 2023)**

Name: _____ Enroll No: _____

Objective(s) :

To understand the concepts of socket programming with threads using the C programming language in a Linux environment. The lab includes an introduction, objectives, equipment/software requirements, code examples, and exercises.

Lab Tasks :

Task 1: What is Sockets Programming?

Task 2: Write and analyze code for ‘client.c’

Task 3: Write and analyze code for ‘server.c’

Task 4: Implement threads to support multiple client requests simultaneously.

Lab Grading Sheet :

Task	Max Marks	Obtained Marks	Comments(<i>if any</i>)
1.	10		
2.	10		
3.	10		
4.	10		
Total	40		Signature

Note : Attempt all tasks and get them checked by your Instructor

Lab 15: Socket Programming using Threads in Linux

Objective(s):

To understand the concepts of socket programming using the C programming language in a Linux environment. The lab includes an introduction, objectives, equipment/software requirements, code examples, and exercises. This lab will introduce you to socket programming in a Linux environment using the C programming language. You will learn how to create a simple server that listens for incoming connections and a client that connects to the server. Additionally, you will implement threads to handle multiple client connections concurrently. These skills are fundamental for developing networked applications and understanding the basics of concurrent programming.

Socket Programming:

It is a fundamental aspect of network programming that allows processes to communicate over a network. This lab focuses on creating a simple server-client application using sockets in the Linux environment. Additionally, threads are employed to handle multiple client connections concurrently.

Implementations:

- Understand the basics of socket programming.
- Implement a simple server that listens for incoming connections.
- Develop a client program to connect to the server.
- Use threads to handle multiple client connections simultaneously.

Code Files:

- '**server.c**' : Code for the server application.
- '**client.c**' : Code for the client application.

Program Code for 'server.c'

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <pthread.h>

#define PORT 8080
#define MAX_CLIENTS 5

void *handleClient(void *socket_desc)
{
    int client_socket = *(int *)socket_desc;
    char buffer[1024] = {0};
    char *message = "Hello from server!\n";

    // Send a welcome message to the client
    send(client_socket, message, strlen(message), 0);

    // Receive data from the client
    recv(client_socket, buffer, sizeof(buffer), 0);
    printf("Client message: %s\n", buffer);

    // Close the socket and free the thread's resources
    close(client_socket);
    free(socket_desc);

    return NULL;
}

int main()
{
    int server_fd, client_socket;
    struct sockaddr_in server_addr, client_addr;
    pthread_t thread_id;

    // Create a socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
    {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

```

```

server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = INADDR_ANY;
server_addr.sin_port = htons(PORT);

// Bind the socket to the specified address and port
if (bind(server_fd, (struct sockaddr *)&server_addr,
          sizeof(server_addr)) < 0)
{
    perror("Bind failed");
    exit(EXIT_FAILURE);
}

// Listen for incoming connections
if (listen(server_fd, MAX_CLIENTS) < 0)
{
    perror("Listen failed");
    exit(EXIT_FAILURE);
}

printf("Server listening on port %d... \n", PORT);

while (1)
{
    int addr_len = sizeof(client_addr);

    // Accept a new connection
    if ((client_socket = accept(server_fd,
                                (struct sockaddr *)&client_addr,
                                (socklen_t *)&addr_len)) < 0)
    {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }

    // Create a new thread to handle the client
    int *new_socket = malloc(sizeof(int));
    *new_socket = client_socket;

    if (pthread_create(&thread_id,
                      NULL, handleClient, (void *)new_socket) < 0)
    {
        perror("Thread creation failed");
        exit(EXIT_FAILURE);
    }
}

```

```

    // Detach the thread to allow it to run independently
    pthread_detach(thread_id);
}

return 0;
}

```

Program Code for 'client.c'

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080

int main()
{
    int sock = 0;
    struct sockaddr_in server_addr;
    char buffer[1024] = {0};

    // Create a socket
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0)
    {
        perror("Invalid address/ Address not supported");
        exit(EXIT_FAILURE);
    }
}

```

```

// Connect to the server
if (connect(sock, (struct sockaddr *)&server_addr,
            sizeof(server_addr)) < 0)
{
    perror("Connection failed");
    exit(EXIT_FAILURE);
}

// Receive a welcome message from the server
recv(sock, buffer, sizeof(buffer), 0);
printf("%s", buffer);

// Send a message to the server
char *message = "Hello from client!";
send(sock, message, strlen(message), 0);

close(sock);

return 0;
}

```

Tasks and Exercises:**Compile and Run:**

- Compile both server.c and client.c using GCC.
- Run the server and client in separate terminal windows.
- Observe the interaction between the client and server.

Multiple Clients:

- Modify the server code to handle multiple clients concurrently using threads.
- Test the server by connecting multiple clients simultaneously.

Enhancements:

- Add error handling to both the server and client code.
- Implement message exchange between the server and multiple clients.

Protocol Design:

- Design a simple protocol for communication between the server and clients.
- Modify the server and client code to adhere to the protocol.