Subject Name: **Operating System**

Subject Code: **22CS005**

Session: **2022-23**

Department: **DCSE**

CHITKARA
UNIVERSITY

**Submitted By:**
Bhuvesh Mittal
2210991450
G5-E

**Submitted To:**
Dr. Rajwinder Kaur

# Index

# EXPERIMENT 1

## PART 1:

## **Installation**: Configuration & Customizations of Linux

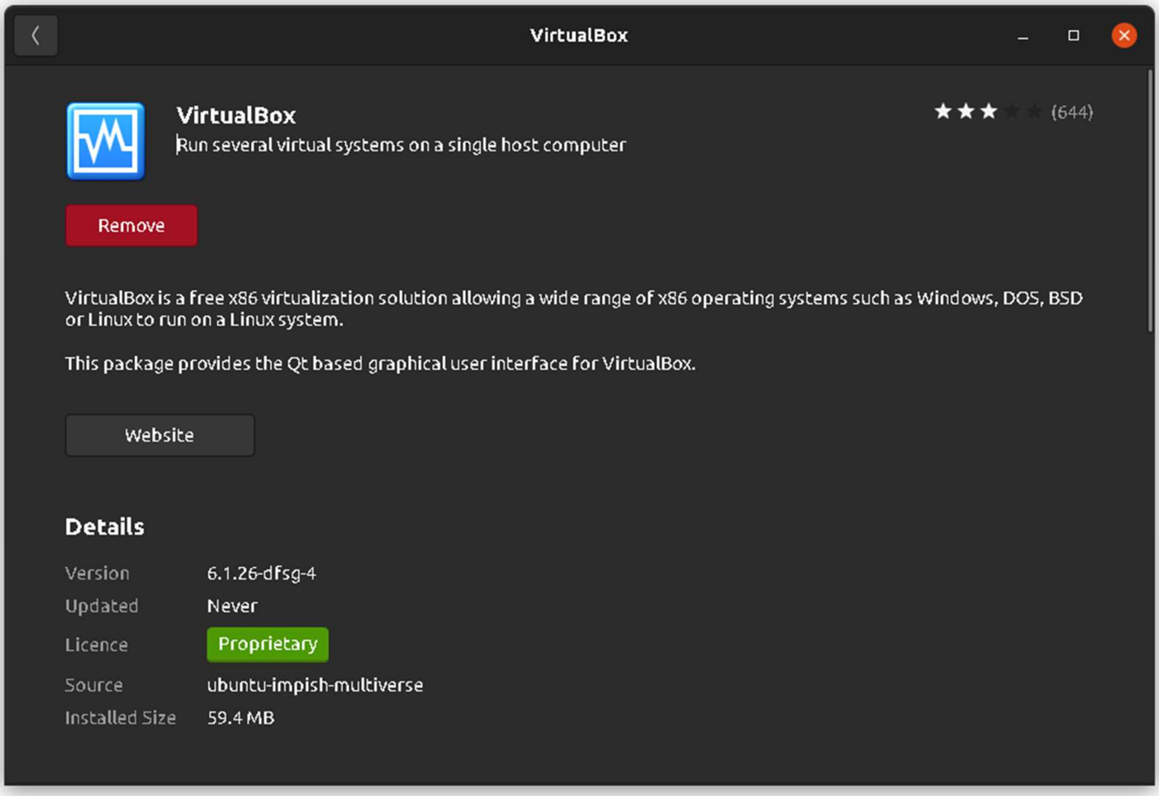
First Download an Ubuntu Image

• You can download an Ubuntu image

• https://ubuntu.com/download/desktop • Make sure to save it to a memorable location on your PC! For this tutorial, we will use the Ubuntu 20.04 LTS release.

• On Mac OS or Windows, you can download VirtualBox from the downloads page https://www.virtualbox.org/wiki/Downloads

• This page also includes instructions to download VirtualBox for Linux. However, on Ubuntu, you can find VirtualBox by simply searching for it in the Ubuntu Software app.



VirtualBox

Run several virtual systems on a single host computer

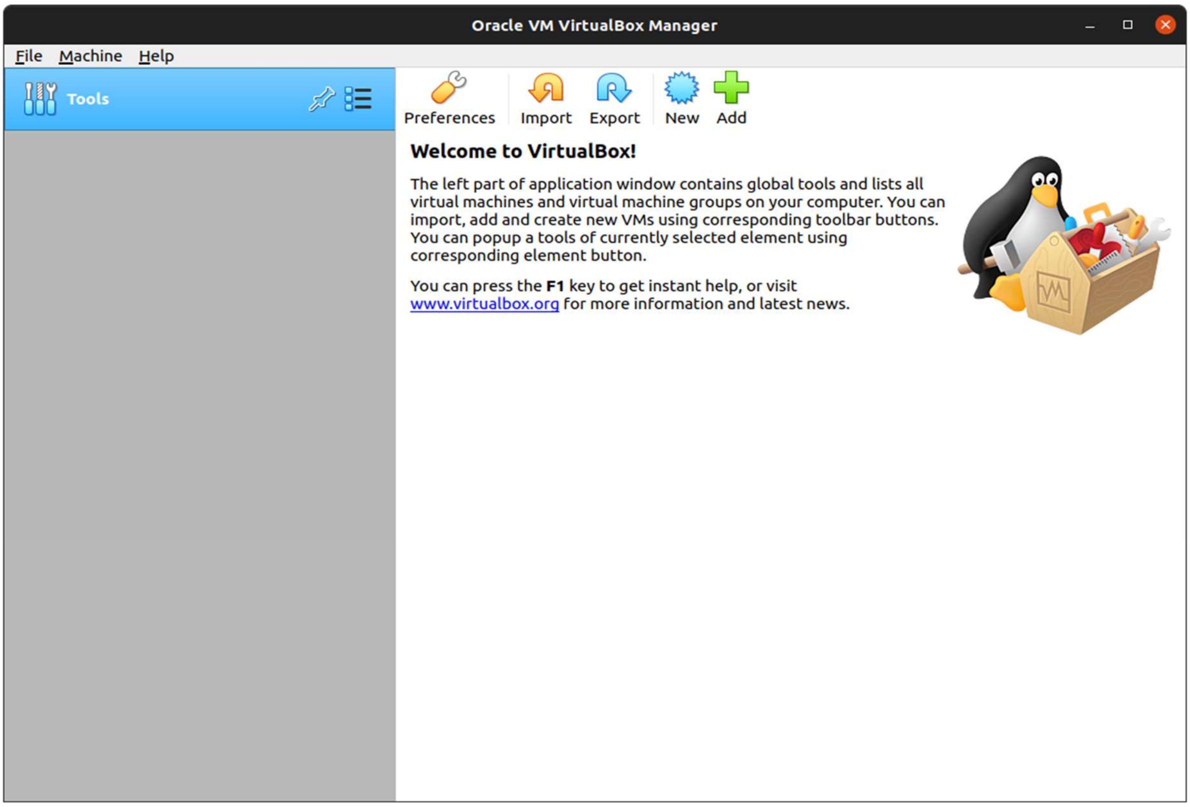
VirtualBox is a free x86 virtualization solution allowing a wide range of x86 operating systems such as Windows, DOS, BSD or Linux to run on a Linux system.

This package provides the Qt based graphical user interface for VirtualBox.

Details

| | |
|---|---|
| Version | 6.1.26-dfsg-4 |
| Updated | Never |
| Licence | Proprietary |
| Source | ubuntu-impish-multiverse |
| Installed Size | 59.4 MB |

- Once you have completed the installation, go ahead and run VirtualBox.



**Welcome to VirtualBox!**

The left part of application window contains global tools and lists all virtual machines and virtual machine groups on your computer. You can import, add and create new VMs using corresponding toolbar buttons. You can popup a tools of currently selected element using corresponding element button.

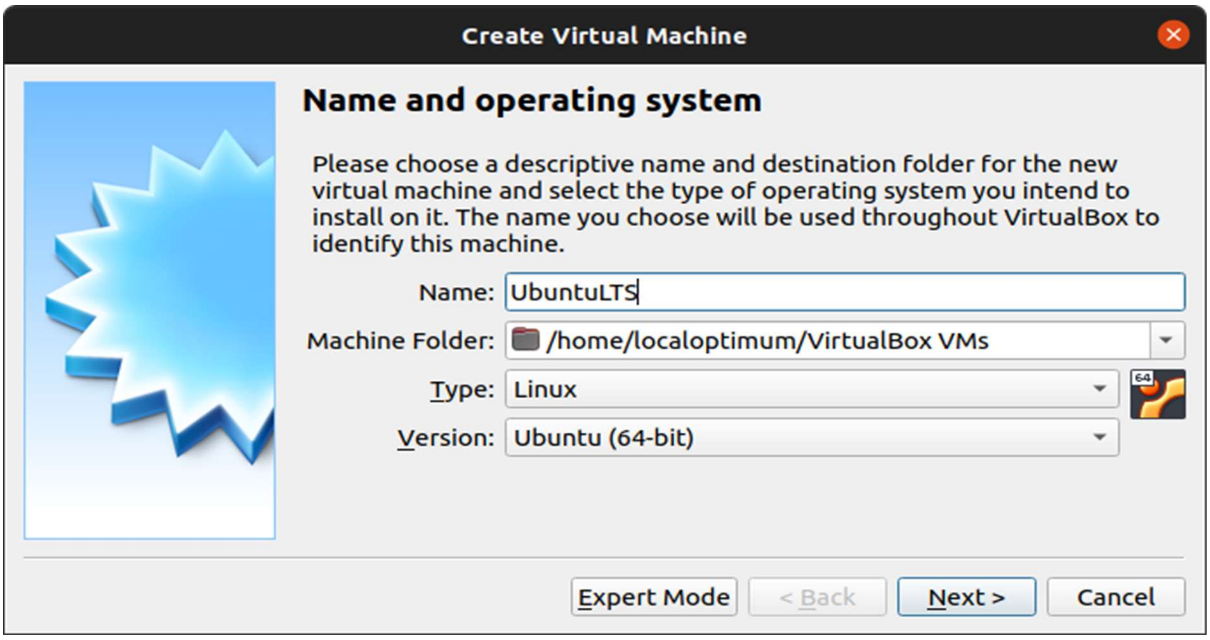
You can press the **F1** key to get instant help, or visit www.virtualbox.org for more information and latest news.

Step 2. Create a new virtual machine

- Click New to create a new virtual machine. Fill in the appropriate details
- Name: If you include the word Ubuntu in your name the Type and Version will

auto-update.

- Machine Folder: This is where your virtual machines will be stored so you can resume working on them whenever you like.
- Type: Linux
- Version: Ubuntu (64-bit)



**Name and operating system**

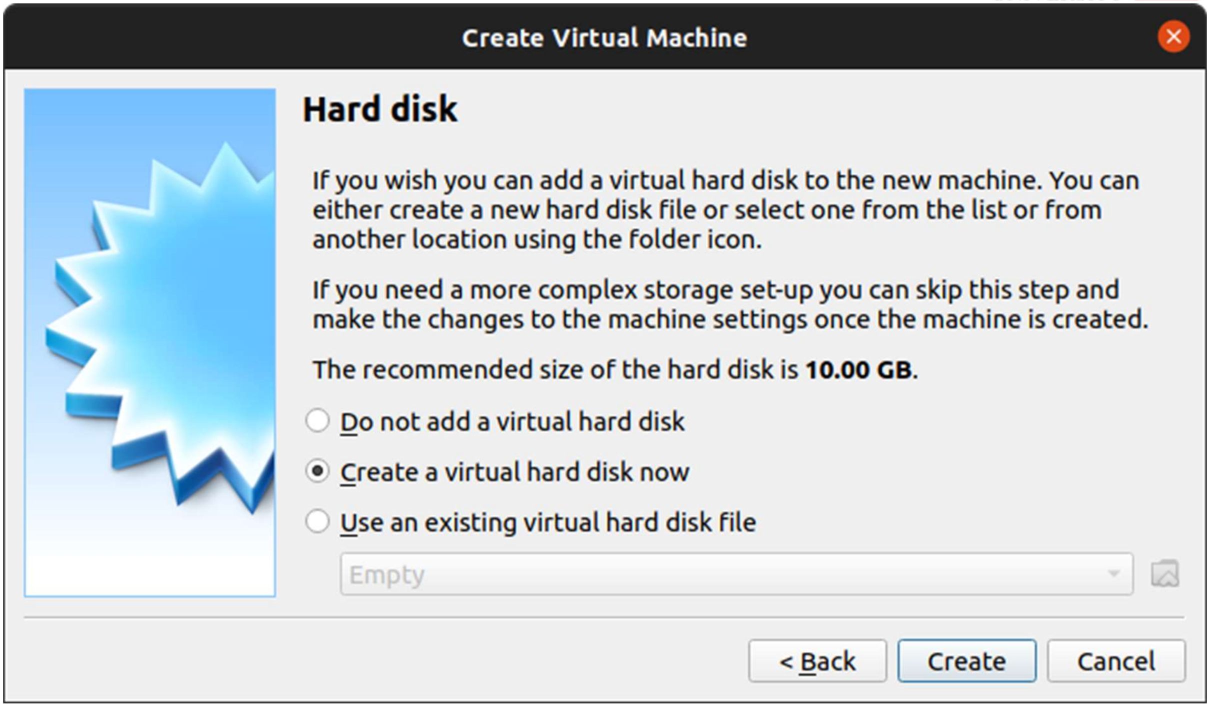
Please choose a descriptive name and destination folder for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name: UbuntuLTS

Machine Folder: /home/localoptimum/VirtualBox VMs

Type: Linux

Version: Ubuntu (64-bit)



**Hard disk**

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

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

The recommended size of the hard disk is **10.00 GB**.

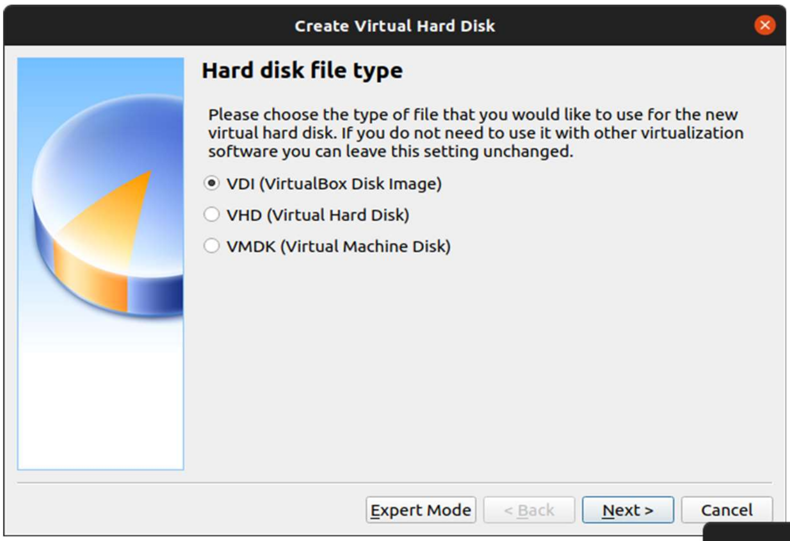
○ Do not add a virtual hard disk

● Create a virtual hard disk now

○ Use an existing virtual hard disk file

Empty

Then you can choose whether the hard disk is dynamically allocated (up to the limit we will set on the next screen), filling up as the VM requires it. Otherwise, we can tell it to allocate the full amount of memory right from the start. This will improve performance but may take up unnecessary space. We’ll leave it as dynamically allocated for this installation.



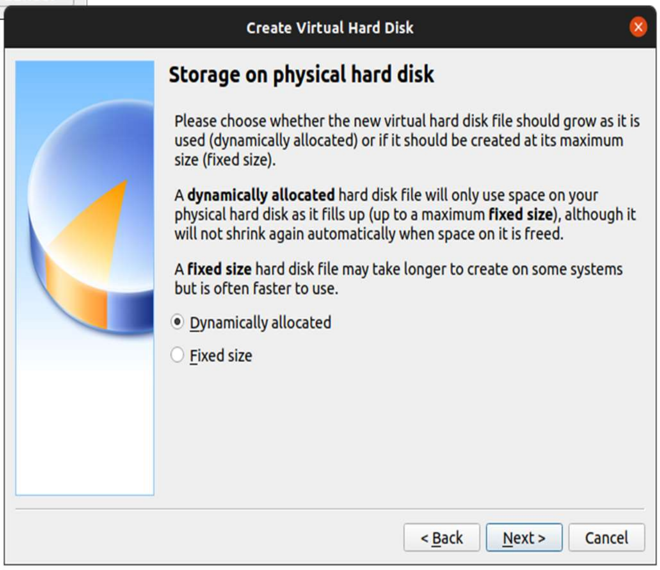
**Hard disk file type**

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

● VDI (VirtualBox Disk Image)

○ VHD (Virtual Hard Disk)

○ VMDK (Virtual Machine Disk)



**Storage on physical hard disk**

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

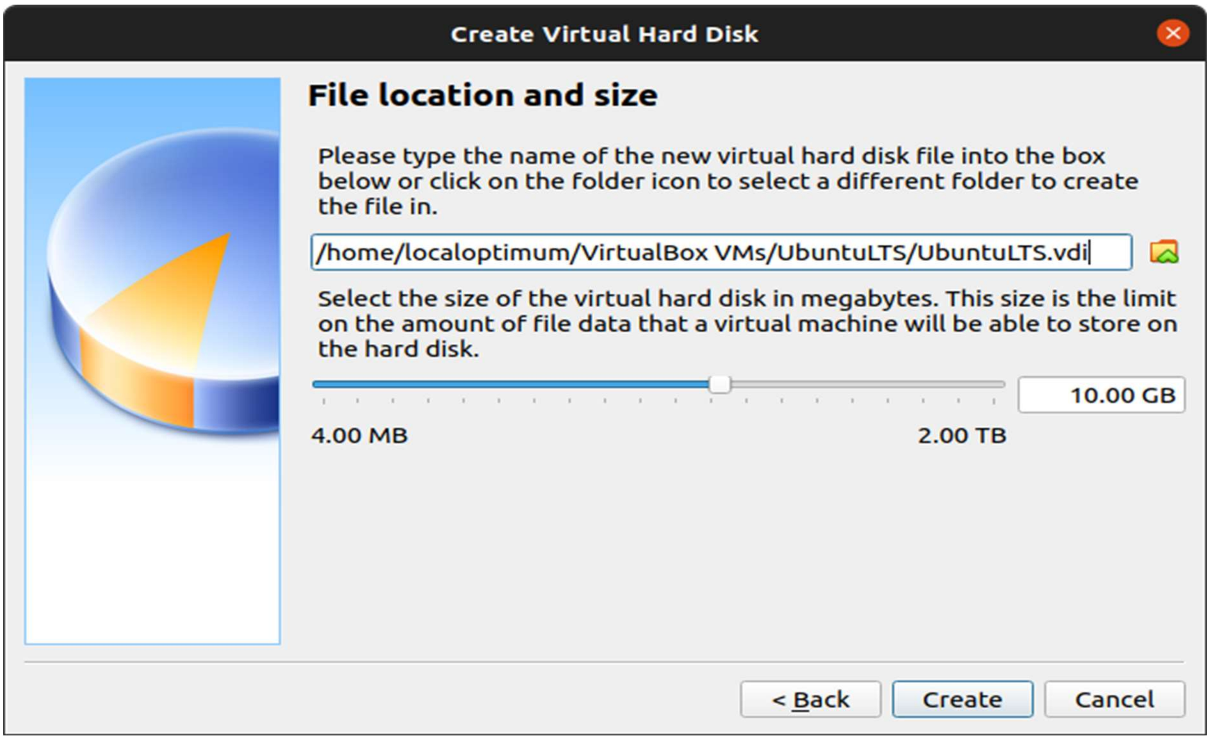
A **dynamically allocated** hard disk file will only use space on your physical hard disk as it fills up (up to a maximum **fixed size**), although it will not shrink again automatically when space on it is freed.

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

● Dynamically allocated

○ Fixed size

- Finally, you can set the maximum amount of memory your VM can access.



Create Virtual Hard Disk

**File location and size**

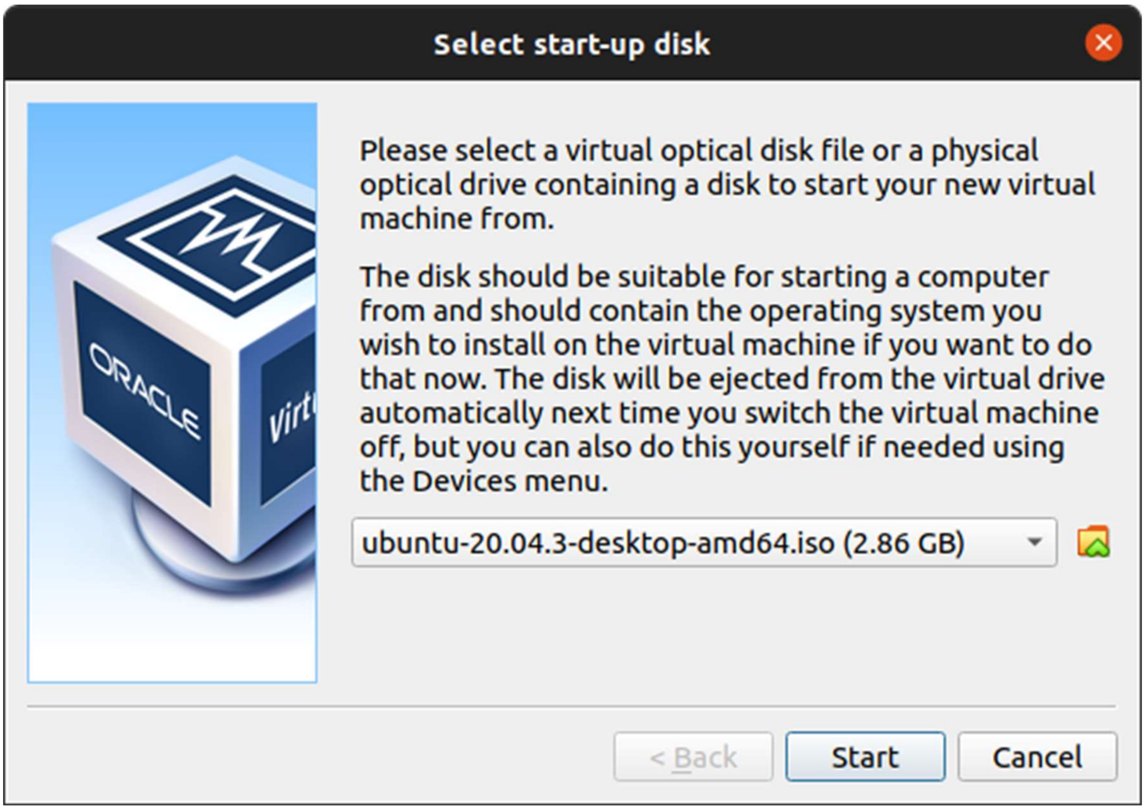
Please type the name of the new virtual hard disk file into the box below or click on the folder icon to select a different folder to create the file in.

/home/localoptimum/VirtualBox VMs/UbuntuLTS/UbuntuLTS.vdi

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

10.00 GB

- Click Start to launch the virtual machine. You will be prompted to select the start-up disk. Use the file icon to open the Optical disc selector and click Add to find your .iso file

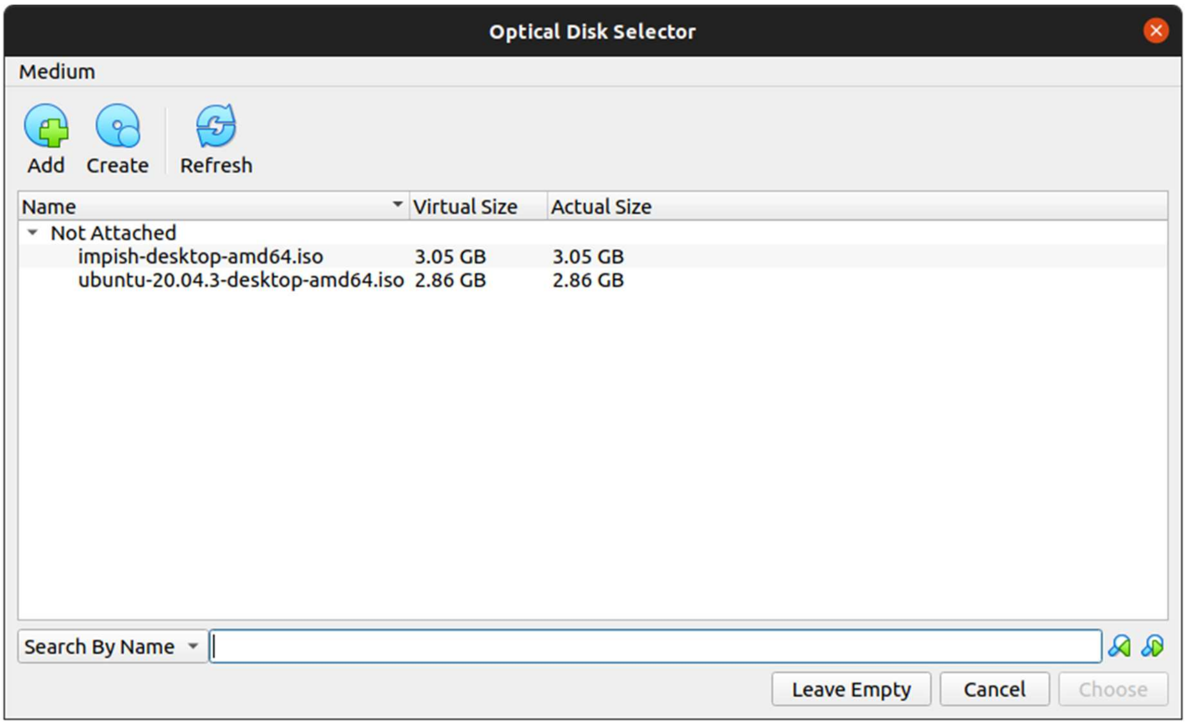


Select start-up disk

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

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

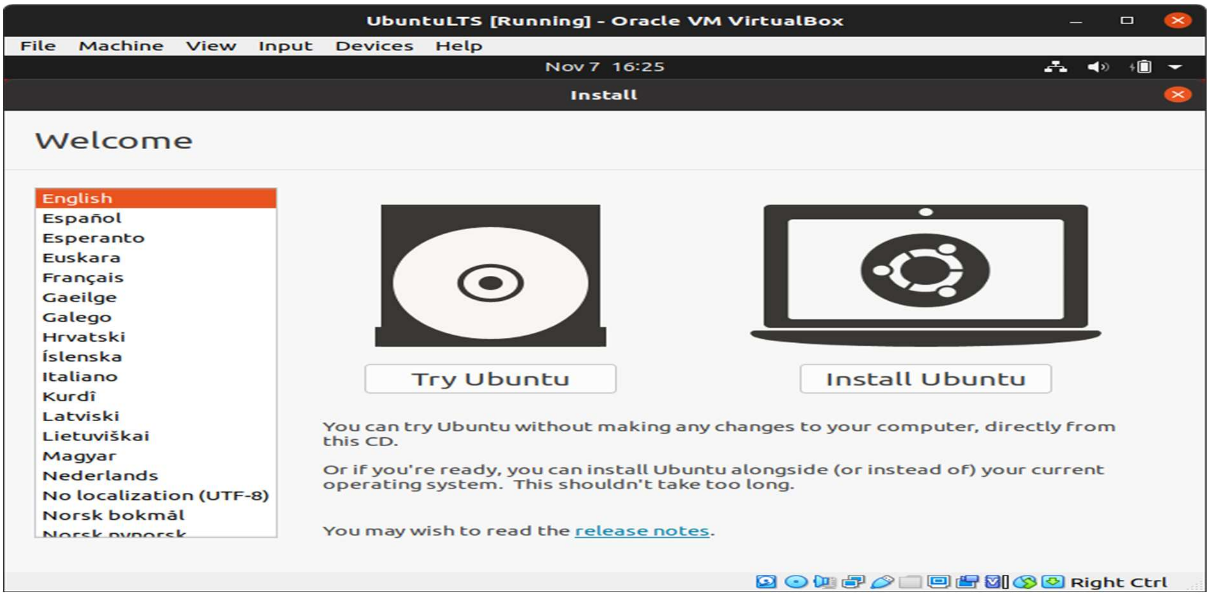
ubuntu-20.04.3-desktop-amd64.iso (2.86 GB)



| Name | Virtual Size | Actual Size |
|---|---|---|
| Not Attached | | |
| impish-desktop-amd64.iso | 3.05 GB | 3.05 GB |
| ubuntu-20.04.3-desktop-amd64.iso | 2.86 GB | 2.86 GB |

Choose the disc image you want to use, then click Start on the start-up disc window.
Note: If you close this window before selecting an image you can still do so from the Devices menu at the top of the VM window. Select Devices > Optical Drives > Choose/Create a disc image…

- Ubuntu desktop should now boot and display the installation menu.



You can try Ubuntu without making any changes to your computer, directly from this CD.

Or if you're ready, you can install Ubuntu alongside (or instead of) your current operating system. This shouldn't take too long.

You may wish to read the release notes.

Now follow instructions to complete installation.

PART 2:

**Introduction to GCC compiler:** Basics of GCC, Compilation of program, Execution of program, Time stamping, Automating the execution using Make file.

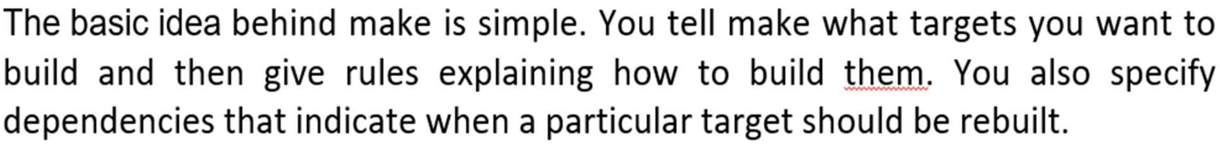
### Major features of GCC

- First of all, GCC is a portable compiler—it runs on most platforms available today.
- GCC is not only a native compiler—it can also cross-compile any program, producing executable files for different system.
- GCC has a modular design, allowing support for new languages.
- Most importantly, GCC is free software.

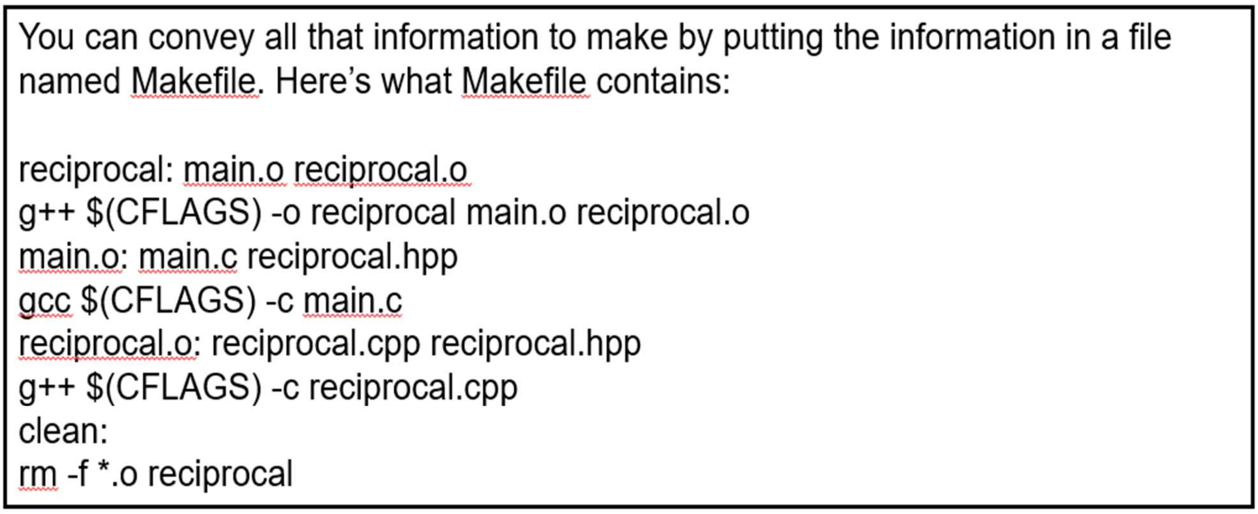
### Compiling a C program

- There are two ways of compiling a C program
- 1). $ gcc bad.c // compiling the C program.
  $ ./a.out // executing the object file.
- 2). $ gcc bad.c -o bad // compilation with different object file name.
  $. /bad // executing the object file.

### Make File

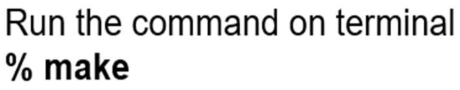


The basic idea behind make is simple. You tell make what targets you want to build and then give rules explaining how to build them. You also specify dependencies that indicate when a particular target should be rebuilt.



You can convey all that information to make by putting the information in a file named Makefile. Here's what Makefile contains:

```
reciprocal: main.o reciprocal.o
g++ $(CFLAGS) -o reciprocal main.o reciprocal.o
main.o: main.c reciprocal.hpp
gcc $(CFLAGS) -c main.c
reciprocal.o: reciprocal.cpp reciprocal.hpp
g++ $(CFLAGS) -c reciprocal.cpp
clean:
rm -f *.o reciprocal
```



Run the command on terminal

**% make**

## Experiment 2

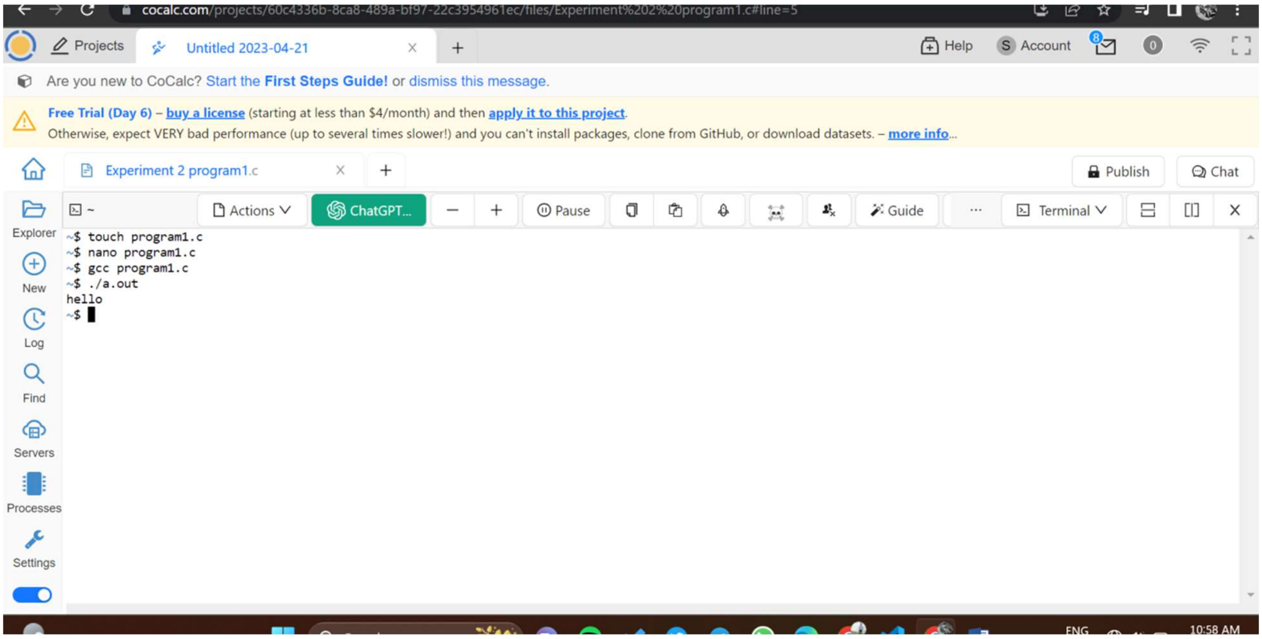
Implement Process concepts using C language by Printing process Id, Execute Linux command as sub process, Creating and executing process using fork and exec system calls.

**Program 1 :**To write some data on the standard output device.

Code :

```
#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<sys/wait.h>

#include<fcntl.h>

#include<unistd.h>

int main()

{

write(1,"hello\n",6);

}
```

**Output :**



cocalc.com/projects/60c4336b-8ca8-489a-bf97-22c3954961ec/files/Experiment%202%20program1.c#line=5

Are you new to CoCalc? Start the First Steps Guide! or dismiss this message.

Free Trial (Day 6) – buy a license (starting at less than $4/month) and then apply it to this project.
Otherwise, expect VERY bad performance (up to several times slower!) and you can't install packages, clone from GitHub, or download datasets. – more info...

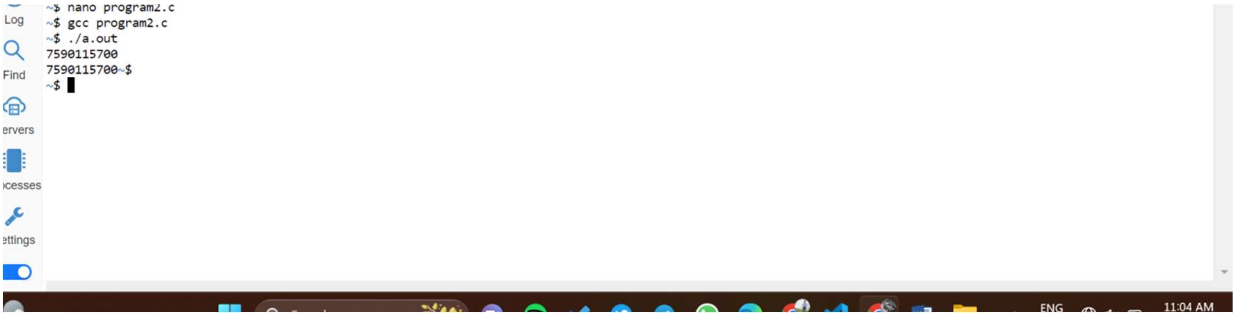
```
~$ touch program1.c
~$ nano program1.c
~$ gcc program1.c
~$ ./a.out
hello
~$ █
```

**Program 2:** To read data from the standard input device and write it on the screen.

Code :

```
#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<sys/wait.h>

#include<fcntl.h>

#include<unistd.h>

int main()

{

char buff[20];

read(0,buff,10);

write(1,buff,10);

}
```

**Output :**



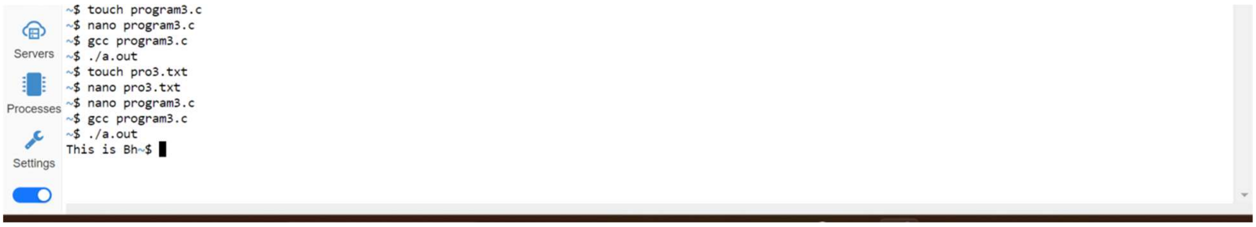
```
~$ nano program2.c
~$ gcc program2.c
~$ ./a.out
7590115700
7590115700~$
~$
```

**Program 3:** Write a program using open () system call to read the first 10 characters of an existing file “test.txt” and print them on screen.

Code :

```
#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

int main()

{

int n,fd;

char buff[50];

fd=open("test.txt",O_RDONLY);

n=read(fd,buff,10);

write(1,buff,n);

}
```

**Output:**



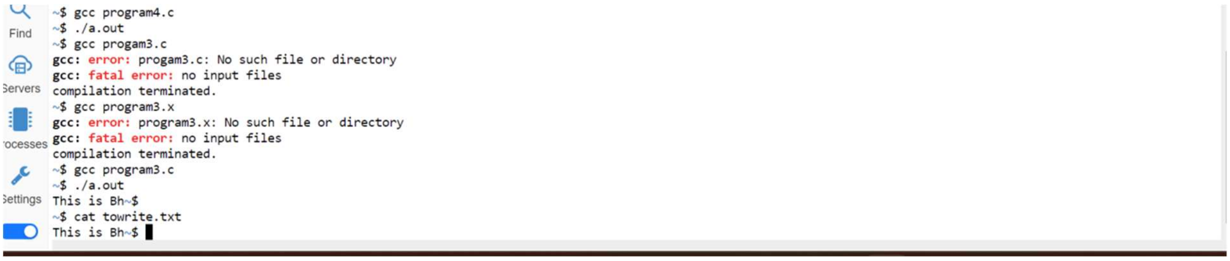
```
~$ touch program3.c
~$ nano program3.c
~$ gcc program3.c
~$ ./a.out
~$ touch pro3.txt
~$ nano pro3.txt
~$ nano program3.c
~$ gcc program3.c
~$ ./a.out
This is Bh~$ █
```

**Program 4:** To read 10 characters from file “test.txt” and write them into non-existing file “towrite.txt”.

Code :

```
#include<unistd.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

int main()

{

int n,fd,fd1;

char buff[50];

fd=open("test.txt",O_RDONLY);

n=read(fd,buff,10);

fd1=open("towrite.txt",O_WRONLY|O_CREAT,0642);

write(fd1,buff,n);

}
```

**Output :**

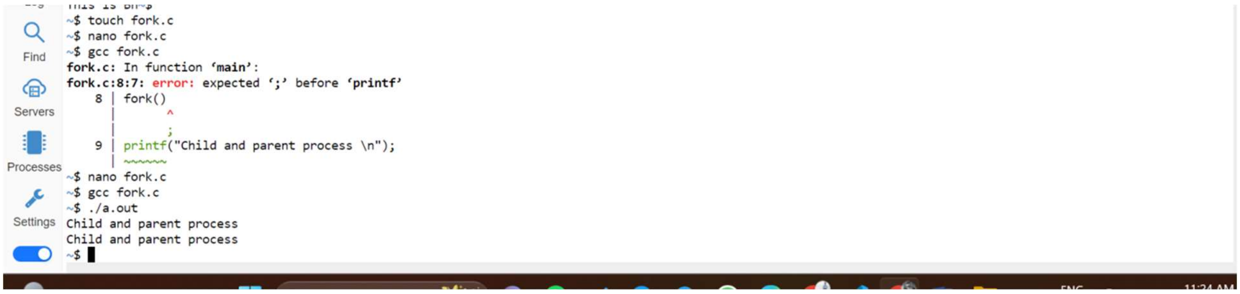


```
~$ gcc program4.c
~$ ./a.out
~$ gcc progam3.c
gcc: error: progam3.c: No such file or directory
gcc: fatal error: no input files
compilation terminated.
~$ gcc program3.x
gcc: error: program3.x: No such file or directory
gcc: fatal error: no input files
compilation terminated.
~$ gcc program3.c
~$ ./a.out
This is Bh~$
~$ cat towrite.txt
This is Bh~$ █
```

**Program 5:** fork () Command

```
#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<sys/wait.h>

#include<fcntl.h>

int main(){

fork();

printf("Child and parent process \n");

}
```

**Output :**



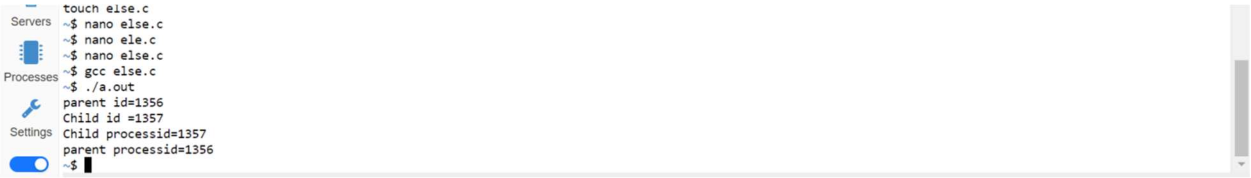
```
This is BH~$
~$ touch fork.c
~$ nano fork.c
~$ gcc fork.c
fork.c: In function ‘main’:
fork.c:8:7: error: expected ‘;’ before ‘printf’
    8 | fork()
      |       ^
      |       ;
    9 | printf("Child and parent process \n");
      | ~~~~~~
~$ nano fork.c
~$ gcc fork.c
~$ ./a.out
Child and parent process
Child and parent process
~$ 
```

**Program 6:** Printing Parent and child process id through if and else block.

Code :

```
#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<sys/wait.h>

#include<fcntl.h>

void main(){

pid_t q;

q=fork();

if(q==-1){

printf("error");}

if(q==0){

printf("Child processid=%d \n",getpid());

printf("parent processid=%d\n",getppid());}

else{

printf("parent id=%d \n",getpid());

printf("Child id =%d \n" ,q);

}

}
```

Output :



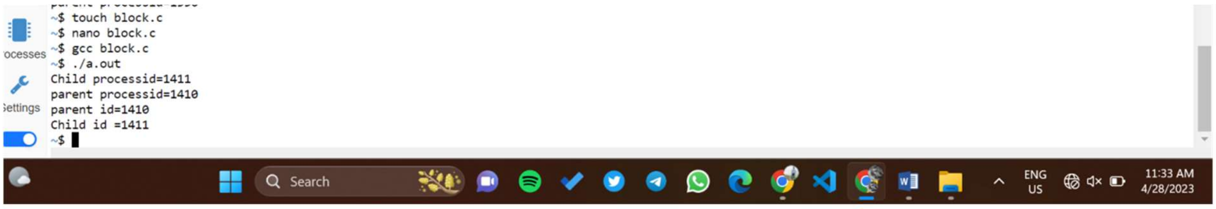
```
touch else.c
~$ nano else.c
~$ nano ele.c
~$ nano else.c
~$ gcc else.c
~$ ./a.out
parent id=1356
Child id =1357
Child processid=1357
parent processid=1356
~$ 
```

**Program 7 :** Printing Parent and child process id through keeping else block on wait().

Code :

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<sys/wait.h>
#include<fcntl.h>
void main(){
pid_t q;
q=fork();
if(q==-1){
printf("error");}
if(q==0){
printf("Child processid=%d \n",getpid());
printf("parent processid=%d\n",getppid());}
else{
wait(NULL);
printf("parent id=%d \n",getpid());
printf("Child id =%d \n" ,q);
}
}
```

Output :



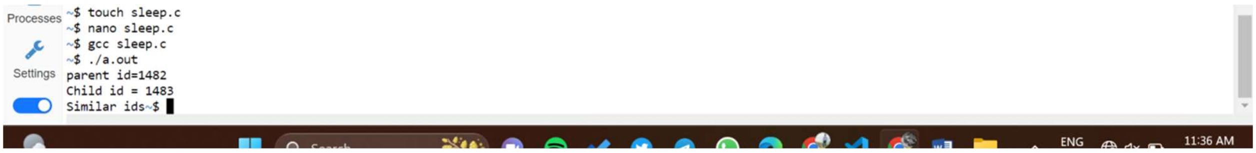
```
~$ touch block.c
~$ nano block.c
~$ gcc block.c
~$ ./a.out
Child processid=1411
parent processid=1410
parent id=1410
Child id =1411
~$
```

**Program 8:** Printing Parent and child process id through keeping if block on sleep().

Code :

```
#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<sys/wait.h>

#include<fcntl.h>

void main(){

pid_t q;

q=fork();

if(q==-1){

printf("error");}

if(q==0){

sleep(10);

printf("Child processid=%d \n",getpid());

printf("parent processid=%d\n",getppid());}

else{

printf("parent id=%d \n",getpid());

printf("Child id = %d \n",q);}

printf("Similar ids");

}
```

Output :



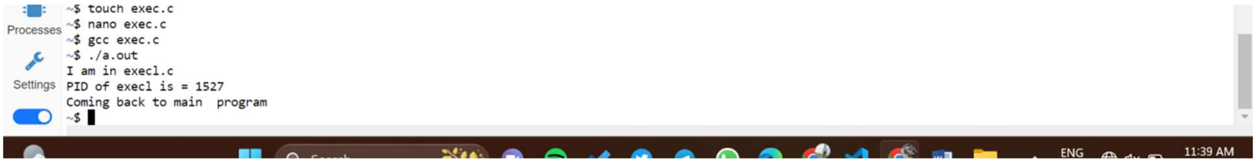
```
~$ touch sleep.c
~$ nano sleep.c
~$ gcc sleep.c
~$ ./a.out
parent id=1482
Child id = 1483
Similar ids~$
```

**Program 9 :** Creating and executing process using fork and exec system calls.

Code :

```
#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<sys/wait.h>

#include<fcntl.h>

int main()

{

printf("I am in execl.c \n");

printf("PID of execl is = %d \n" ,getpid());

char *args[]={"./Hello",NULL};

execv(args[0],args);

printf("Coming back to main  program  \n");

return 0;

}
```

Output :



```
~$ touch exec.c
~$ nano exec.c
~$ gcc exec.c
~$ ./a.out
I am in execl.c
PID of execl is = 1527
Coming back to main  program
~$ 
```