**Practical File**

of

Operating System 24CSE0105

### Submitted

***in partial fulfillment for the award of the degree of***

**BACHELEOR OF ENGINEERING**

***in***

COMPUTER SCIENCE & ENGINEERING



**CHITKARA UNIVERSITY CHANDIGARH-PATIALA NATIONAL HIGHWAY**

**RAJPURA (PATIALA) PUNJAB-140401 (INDIA)**

Chitkara University,Punjab February, 2025

# Submitted To: Submitted By:

**Faculty Name: Dr Teena Narang Designation: Assistant Professor**

**Name: Jiya Manchanda**

**Roll No: 2410990506**

## TABLE OF CONTENTS:

|  |  |  |
| --- | --- | --- |
| S.No. | Title of Experiment | Page No. |
| 1. | a) Installation: Configuration & Customizations of Linux  b) Introduction to GCC compiler: Basics of GCC, Compilation of program, Execution of program  c) Time stamping,  d) Automating the execution using make file. | 3-5 |
| 2. | Implement the commands that is used for Creating and Manipulating files: cat, cp, mv, rm, ls and its options, touch and their options, which is, where is, what is | 6-7 |
| 3. | Implement Directory oriented commands: cd, pwd, mkdir, rmdir | 8-10 |
| 4. | Implement the basic and user status commands like: su, sudo, man, help, history, who, whoami, id, uname, uptime, free, tty, cal, date, hostname, reboot, clear | 11-12 |
| 5. | Implement Process concepts using C language by Printing process Id. | 13-14 |
| 6. | Implement FCFS, SJF scheduling algorithms in C language |  |
| 7. | Implement priority scheduling, and RR scheduling algorithms in C language |  |

**EXPERIMENT 1:**

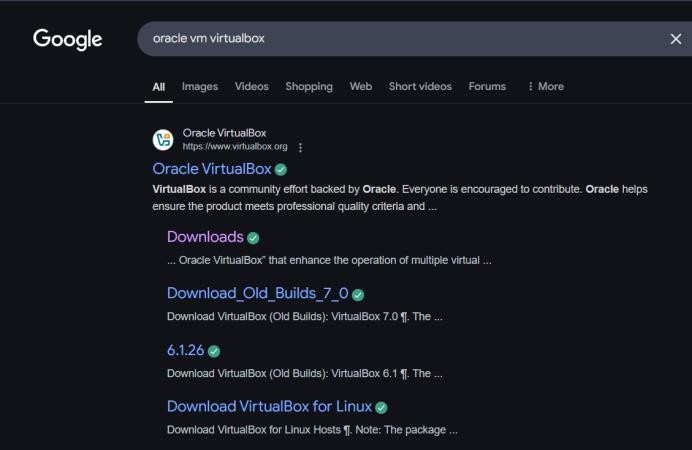
**AIM: Installation:** Configuration & Customizations of Linux

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

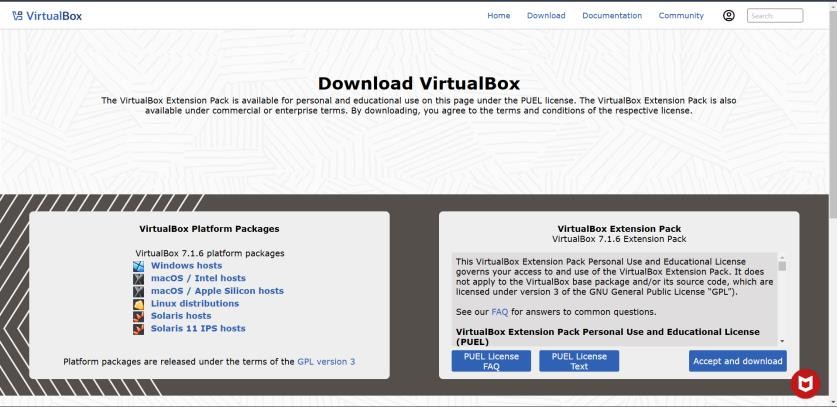
**THEORY: Introduction to Linux:** Linux is a Unix-like open-source operating system for computers, servers, mobile phones, and embedded devices. A program that deals with the hardware and resources of a system directly, e.g., CPU, memory, storage, is referred to as an operating system (OS). An OS works as a mediator between software and hardware to allow both of them to communicate effectively with each other. Initially released by Linus Torvalds on 17 September 1991, Linux was intended for personal computers but has later been ported to other systems such as servers, mainframes, and supercomputers. It is today also widely employed in embedded systems such as routers, automation controllers, televisions, digital video recorders, gaming consoles, and smartwatches. Android, which is among the greatest successes of Linux, is an operating system employing the Linux kernel. It is typically provided in a set of distributions known as Linux distributions, which combine the OS with other software and tools.

**PROCEDURE: Installation of Virtual Box:**

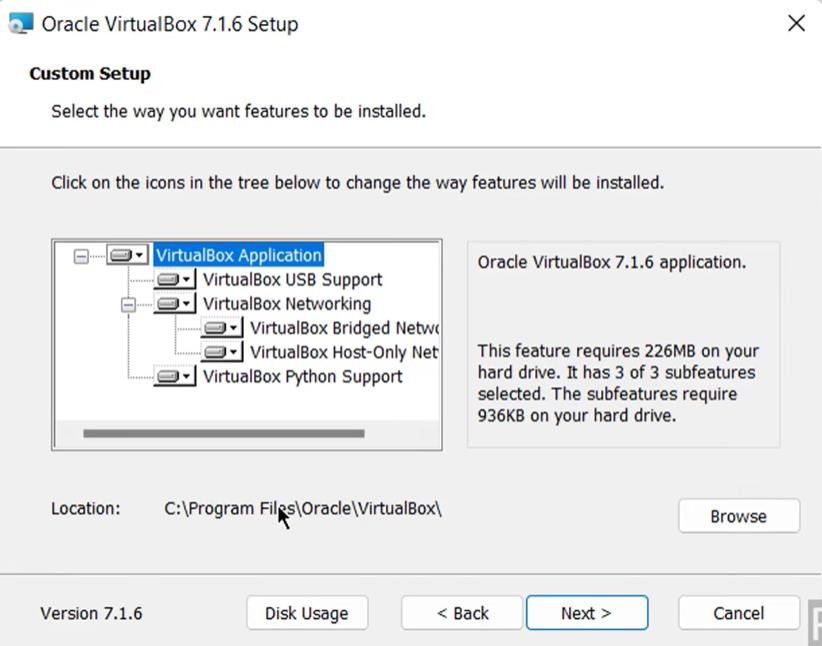
1. Go to Google chrome and click on the ‘Oracle VM Virtual box’.



1. Then, this page will be displayed, click on ‘Windows hosts’ and version 7.1.6 will get downloaded.

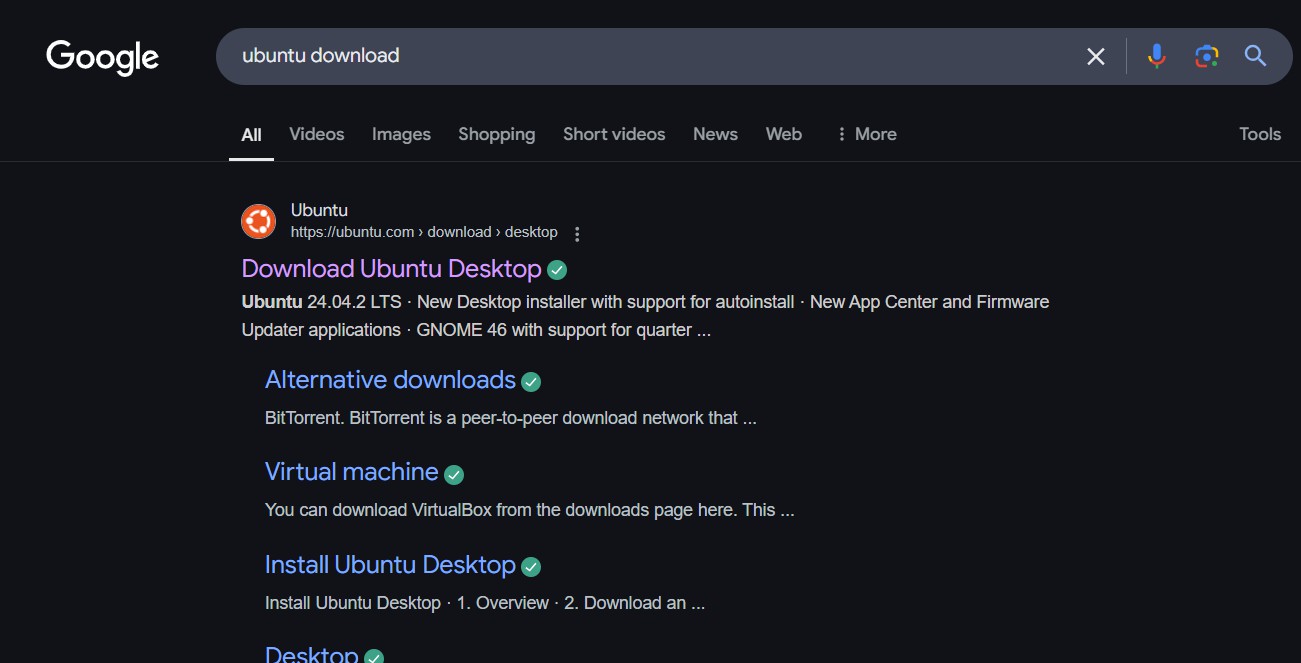


1. After that go to the downloaded file and install it on your system



**Installation of Ubuntu**:

1. Go to Google chrome and click on the ‘Download Ubuntu Desktop’.



1. Now open VirtualBox and click “New”



1. Now, open Virtual Box and add Ubuntu. After all the setup is done, you’re ready to go.
2. You will get the following screen.

**b. GCC compiler and installation of GCC compiler:**

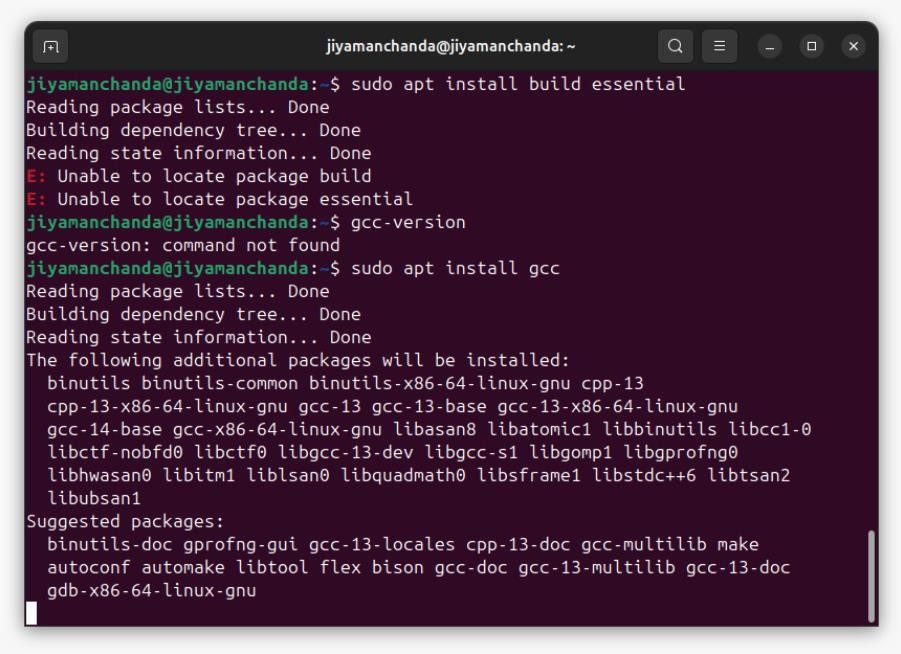
The term compiler refers to a piece of software that converts our source code from a high-level programming language to a low-level programming language (machine

-level code) to build an executable program file and in Linux Operating Systems and compile C program in Linux, we'll need to install the GCC Compiler.

1. First open the terminal.
2. Install GCC Compiler.

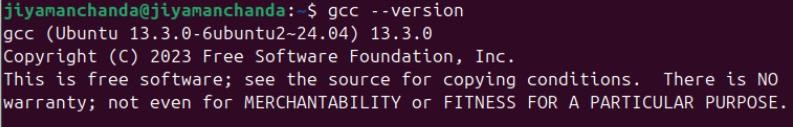
**sudo apt update**: updates the package list.

**sudo apt install gcc** : installs the GNU Compiler Collection.

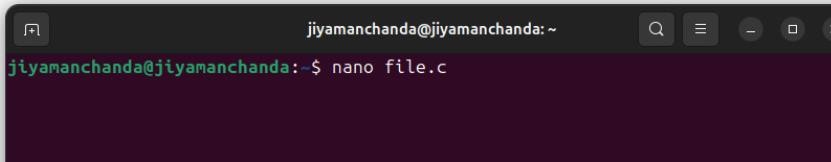


The gcc compiler is installed successfully:

**gcc –version :** checks the version



1. Use a text editor like nano to create a new file.



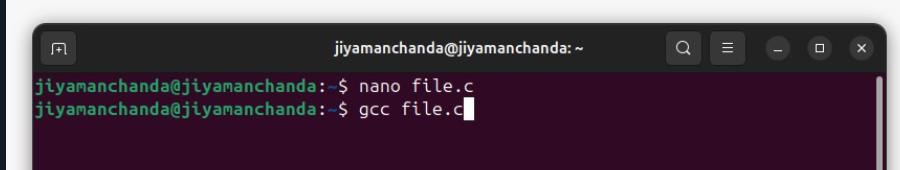
1. In the nano editor, type the following C code:



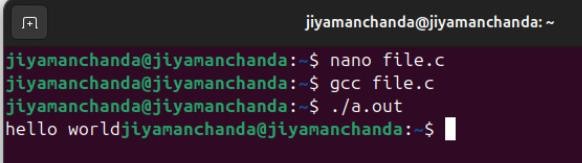
1. Save the file by pressing Ctrl + O, then press Enter. Exit nano by pressing Ctrl + X and

then press Ctrl + Y.

1. Compile the Program.



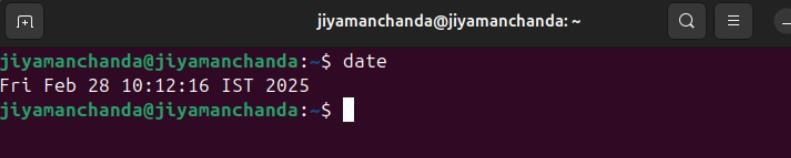
1. To generate the output of the compiled C program, type “./a.out” in the terminal.



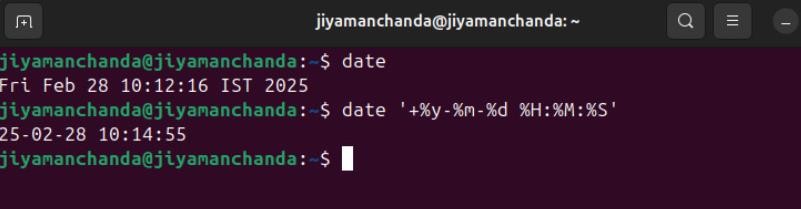
# c.Time stamping

In Linux, you can use the date command to get the current timestamp. Here’s how you can do it:

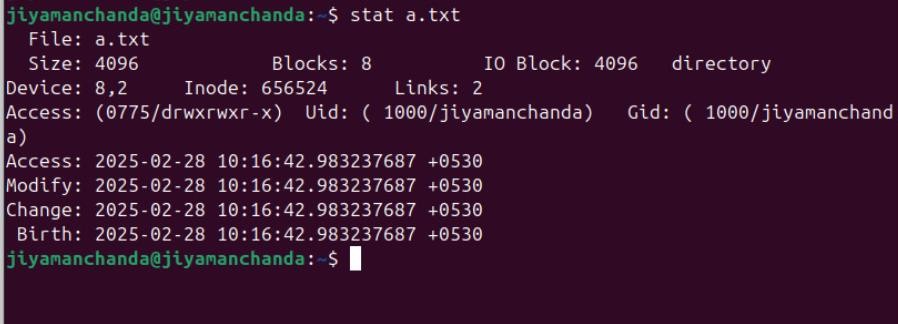
1. **Current Date and Time:** This command will display the current date and time.



1. **Custom Format:** This command will display the date in the format Year-Month-Day Hour:Minute:Second.



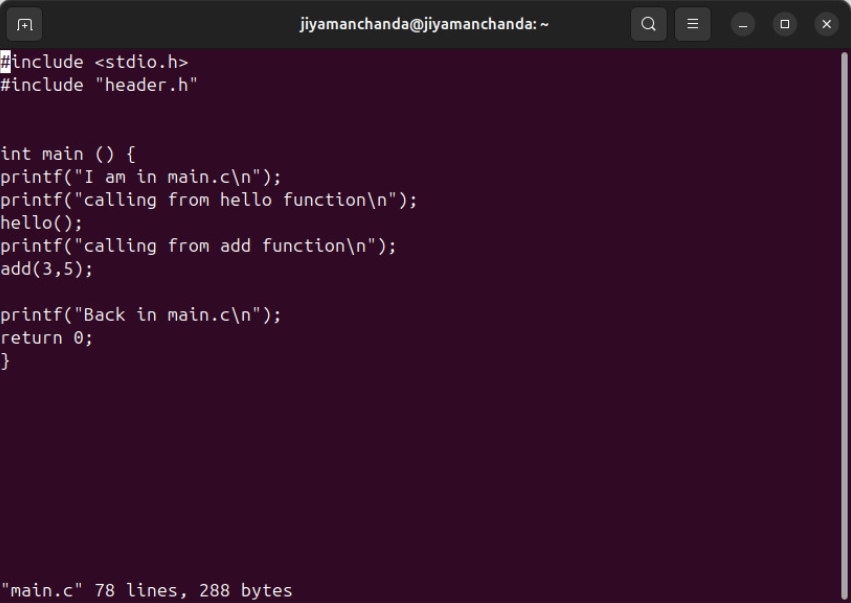
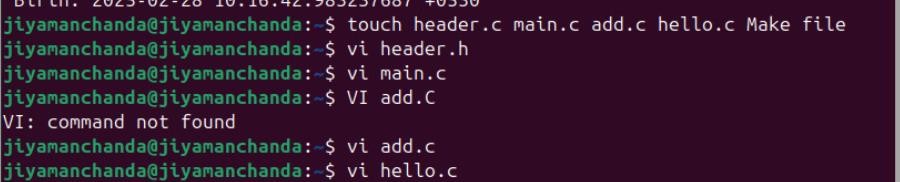
1. **File Timestamps:** This command is used to view file’s timestamps, such as Access, Modify, and Change timestamps.



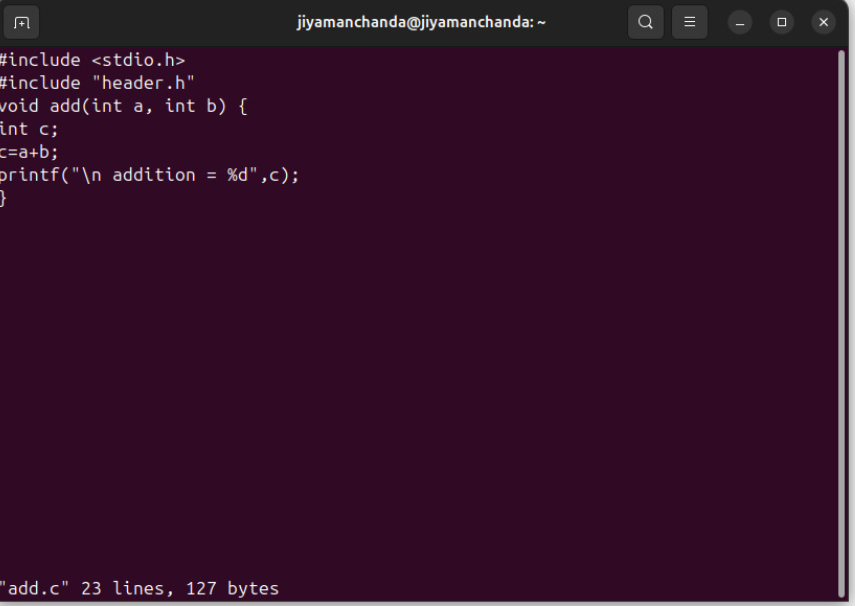
# 

# d) Automating the execution using Make file.

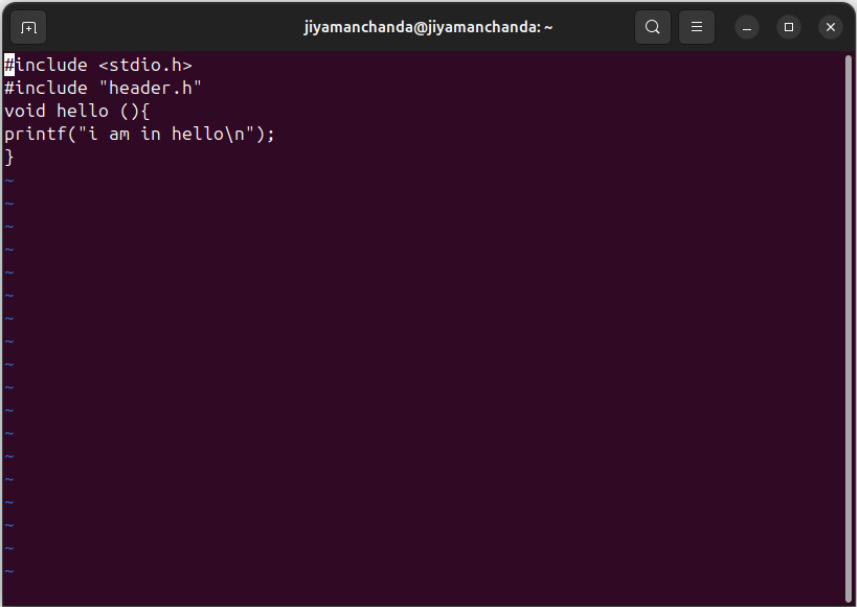
* 1. We create four files header.h, main.c, add.c and hello.c by using touch command and write code by using VI editor.



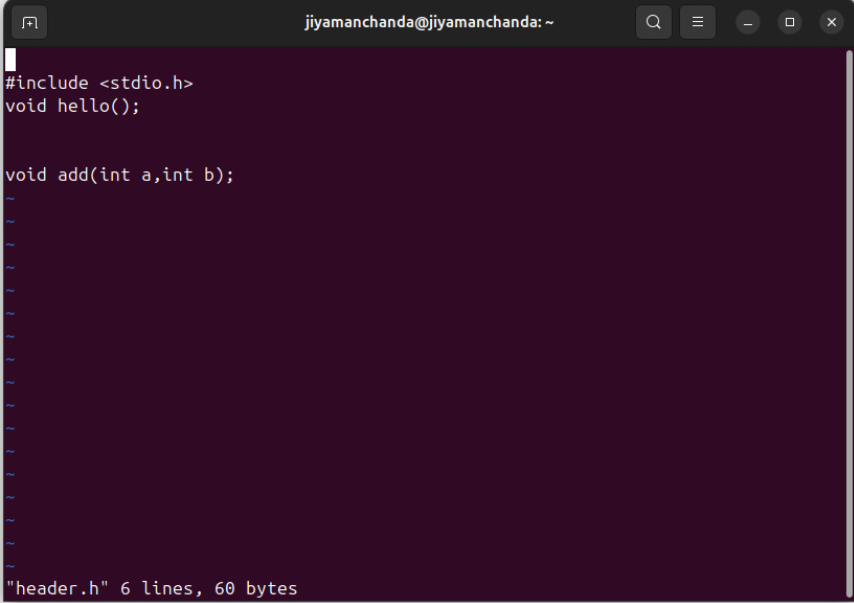
Main .c



Add.c

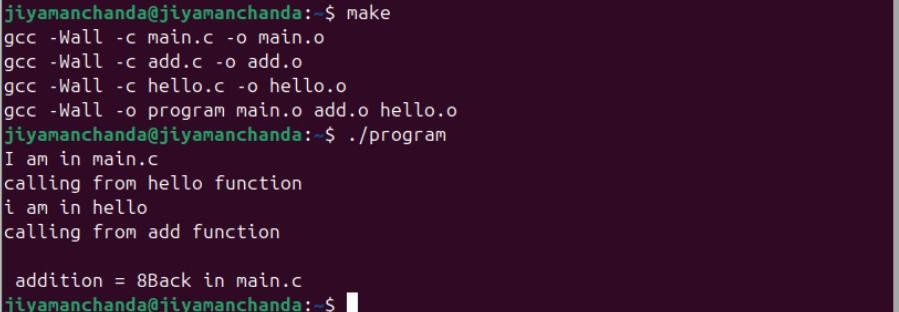
.

hello.c



header.h

* + 1. Type make command in terminal for compilation of files and using ./”file name” for execution.



## PROGRAM 2

**PROGRAM 2:-** Implement the commands that is used for creating and manipulating files: cat, cp, mv, rm, ls and its options, touch and their options, whichis, whereis, whatis.

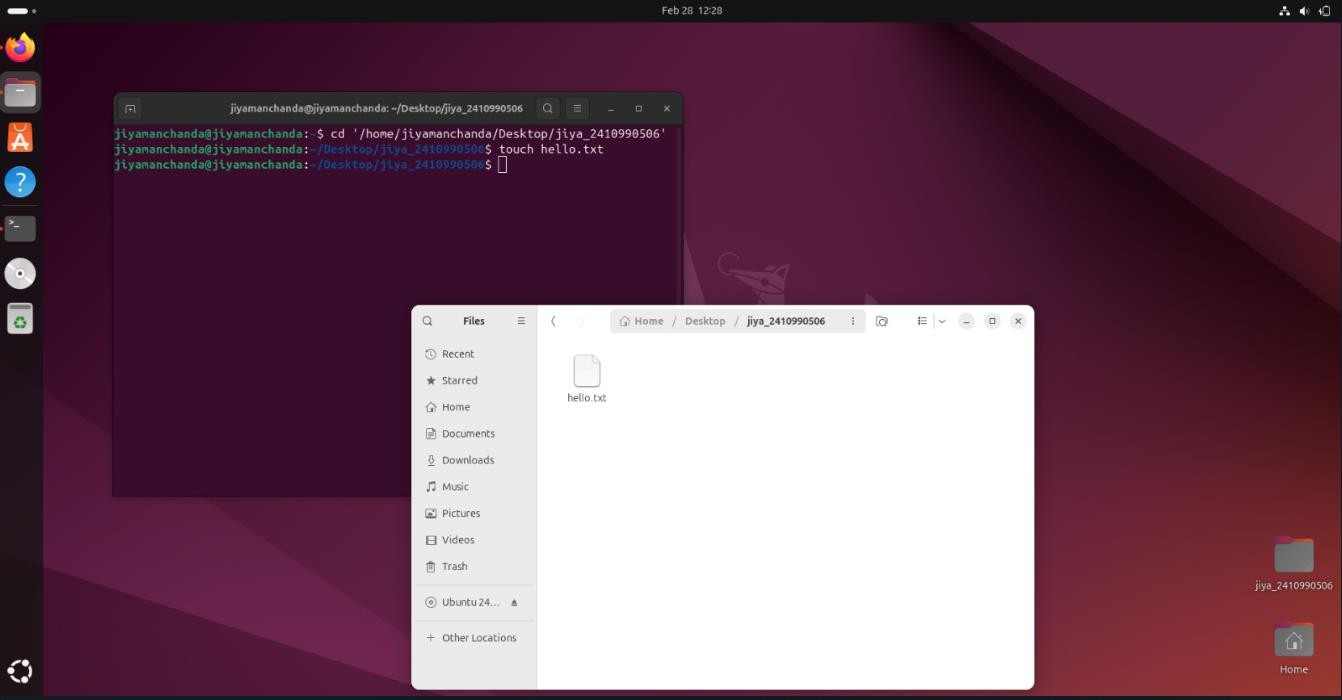
### Solution:-

1. **touch:** This command is used to create empty files or update the access and modification times of existing files.

Usage: touch [OPTIONS] FILE

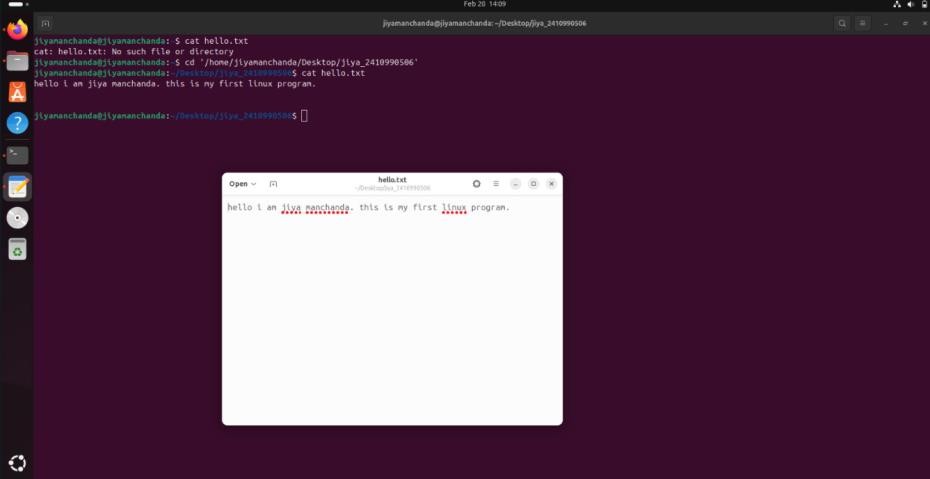
Common options:

* + 1. -a: Change only the access time.
    2. -m: Change only the modification time.

Example: touch hello.txt

1. **cat:** This command is used to concatenate and display the content of files.

Usage: cat [OPTIONS][FILE]

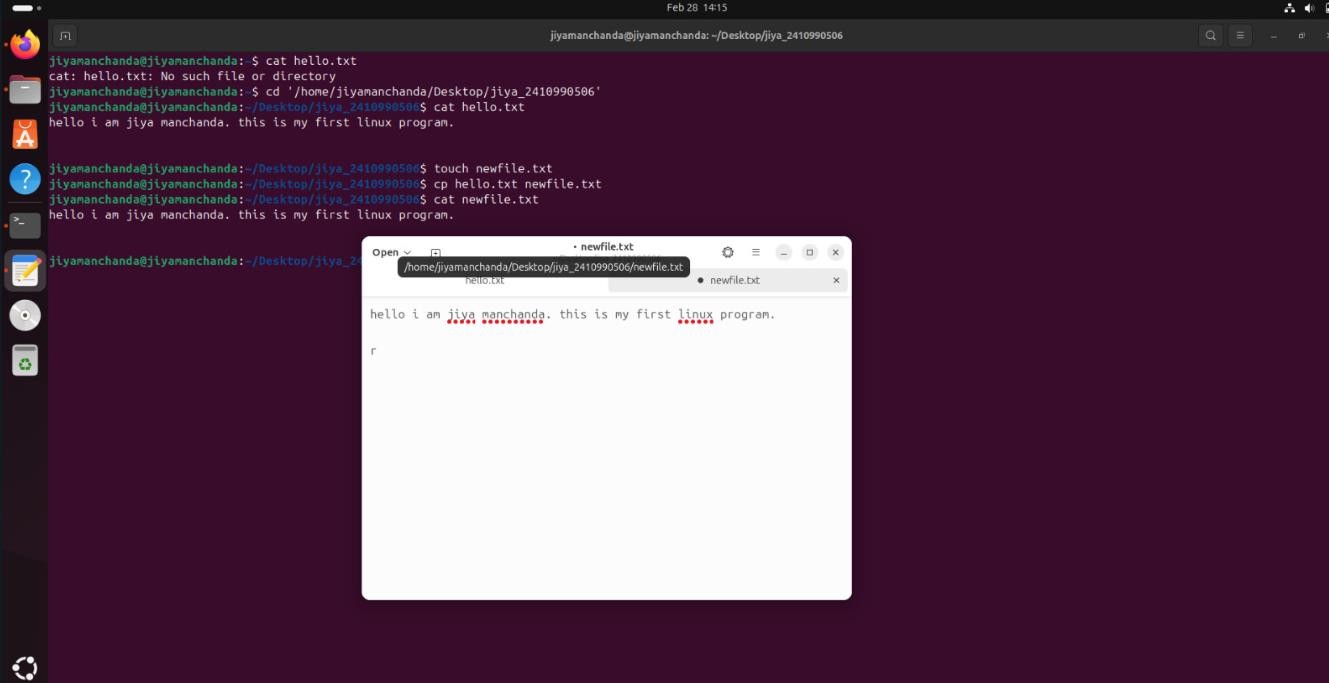
Example: cat hello.txt

1. **cp:** This command is used to copy files or directories.

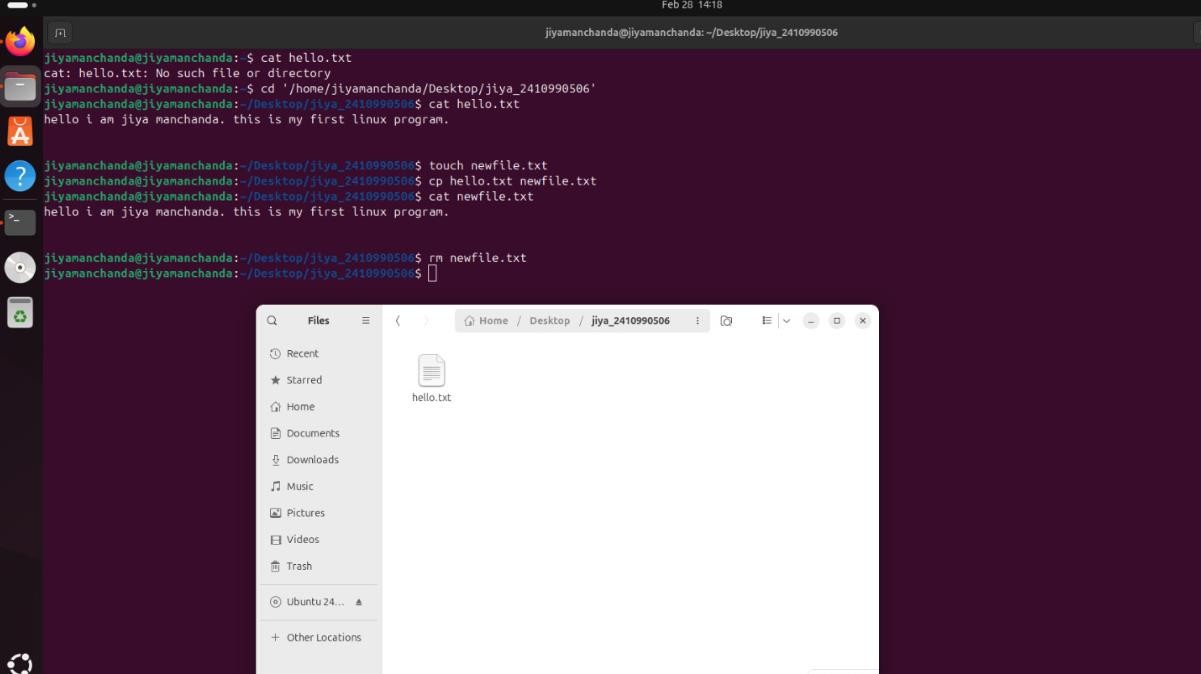
Usage: cp [OPTIONS] SOURCE DESTINATION

Example: cp hello.txt newfile.txt

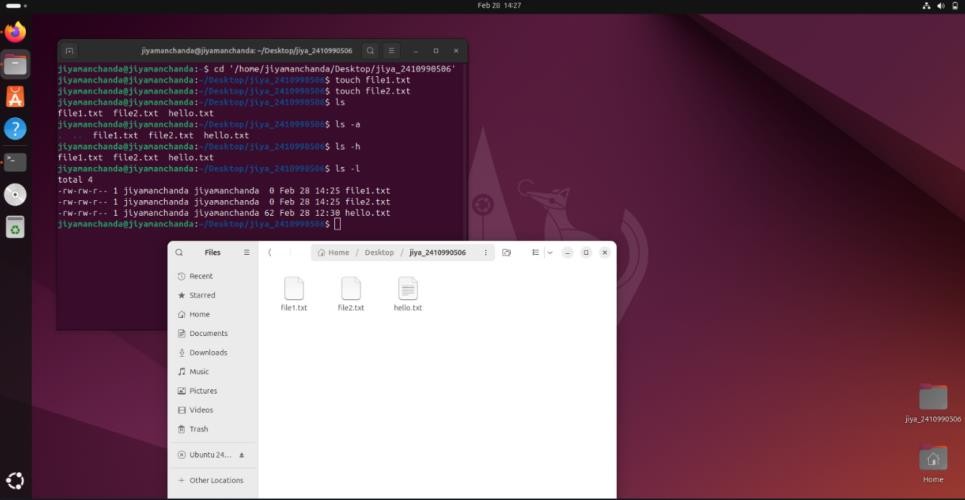
The content of file.txt will get copied in file2.txt



* 1. **rm:** This command is used to remove files or directories.
  2. Usage: rm [OPTIONS] FILE
  3. Example: rm newfile.txt

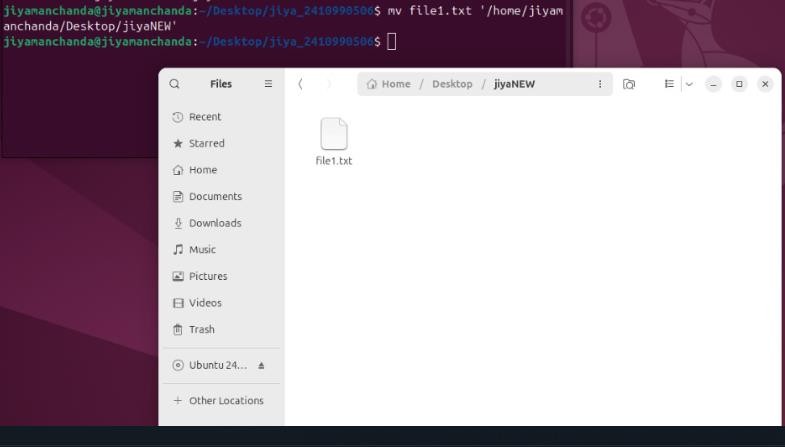


1. **ls:** This command is used to list files and directories.
   1. Usage: ls [OPTIONS] [FILE]
   2. Common options:
      1. -l: Long listing format, displaying additional information like permissions, owner, size, etc.
      2. -a: Include hidden files and directories (those starting with a dot).
      3. -h: Print sizes in human-readable format.



### Touch command will all options.

1. **mv:** This command is used to move files or directories.
   1. Usage: mv [OPTIONS] SOURCE DESTINATION
   2. Example: mv file.txt ‘/home/Tanishq/Desktop/New’



1. **which:** This command is used to locate the binary file associated with a given command.
   1. Usage: which COMMAND
   2. Example: which ls



1. **whereis:** This command is used to locate the binary, course, and manual page files for a command.
   1. Usage: whereis COMMAND
   2. Example: whereis ls



1. **whatis:** This command is used to display a short description of a command.
   1. Usage: whatis COMMAND
   2. Example: whatis ls



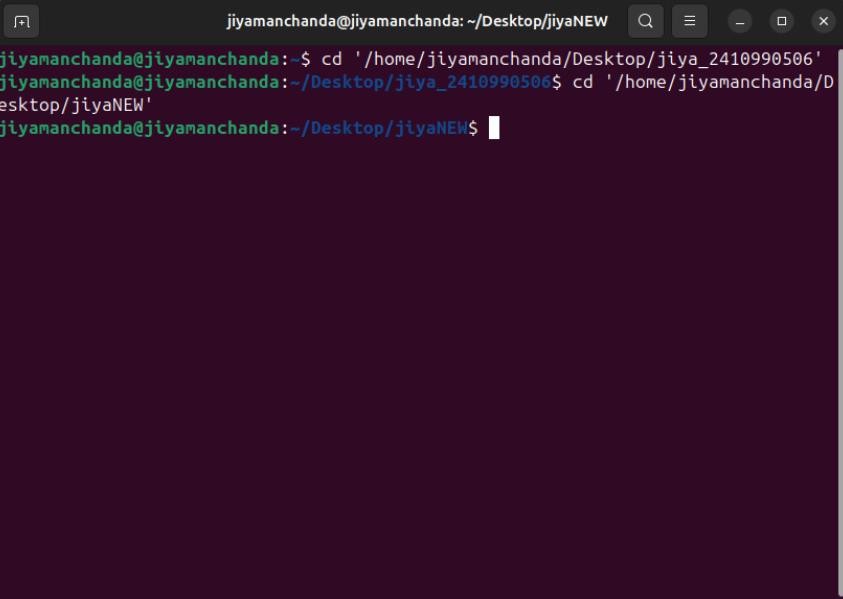
PROGRAM 3

### Program 3: Implement directory oriented commands: cd, pwd, mkdir, rmdir. Solution:

1. **cd:** This command is used to change the current wordking directory.

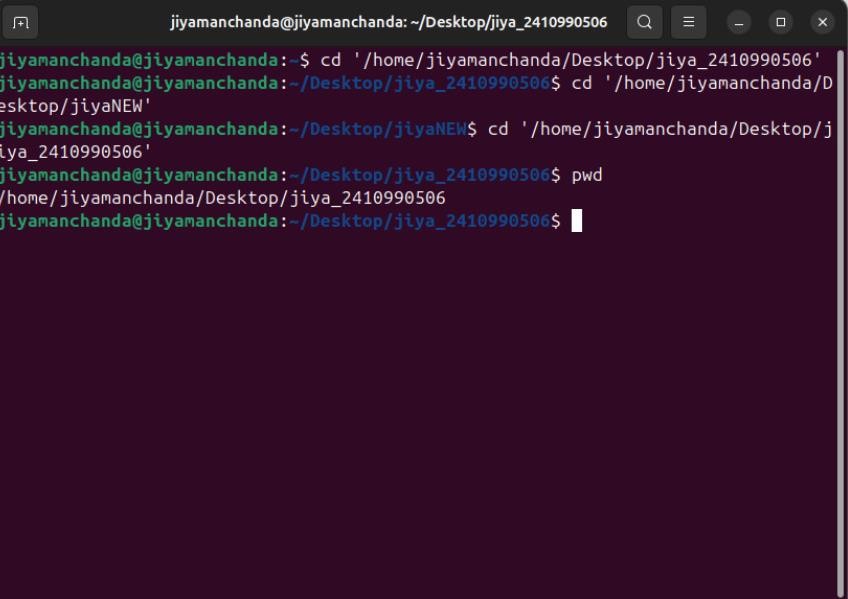
Usage: cd [DIRECTORY]

Example: cd ‘/home/jiyamanchanda/Desktop/jiya\_2410990506’



1. **pwd:** This command displays the current working directory.

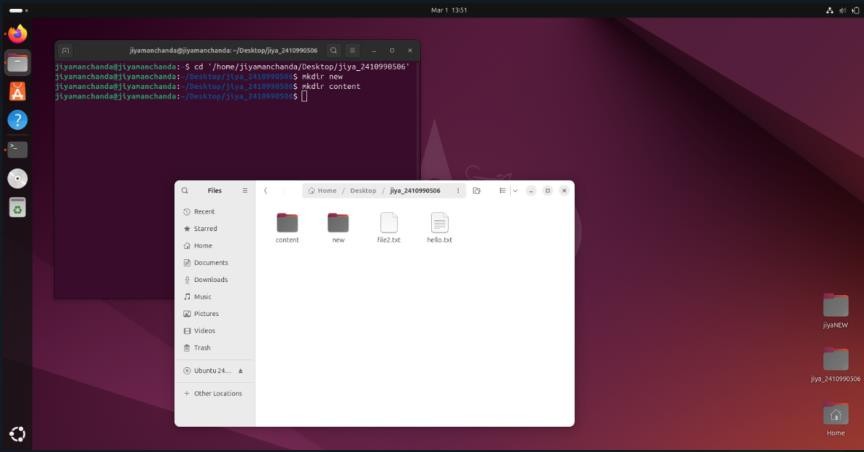
Example: pwd



1. **mkdir:** This command makes a new directory, or multiple directories at once.

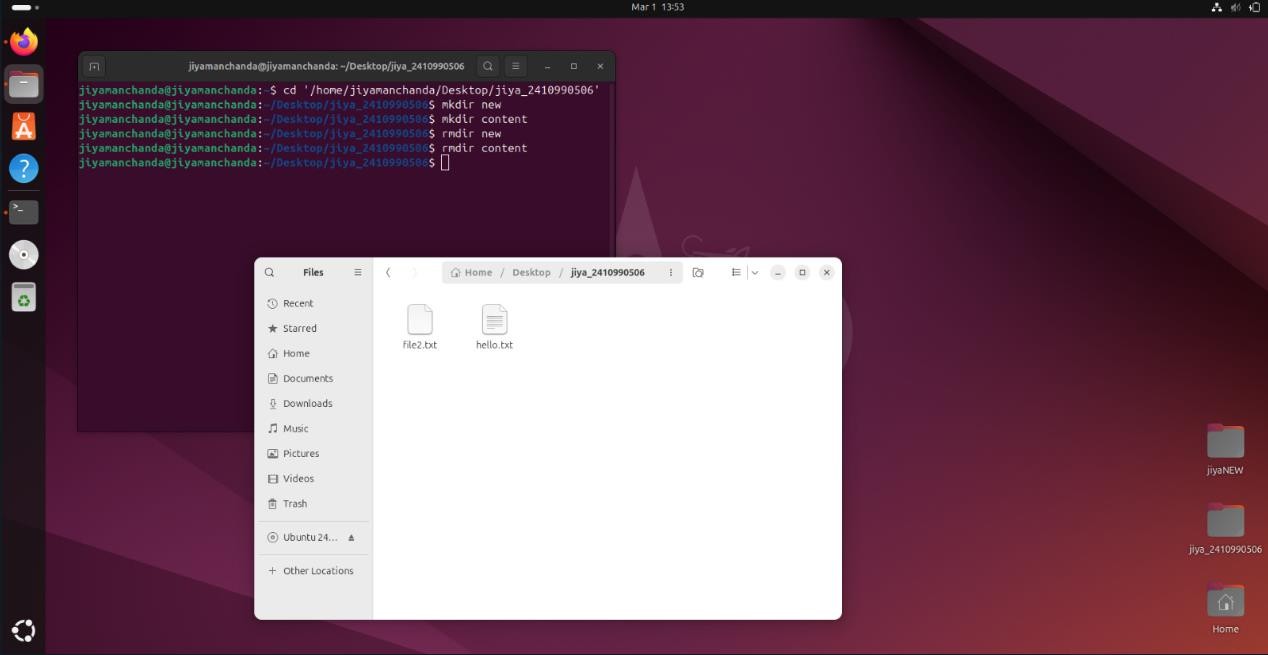
Usage: mkdir [OPTIONS] [DIRECTORY\_NAME]

Example: mkdir content



1. **rmdir:** This command removes directories.

Usage: rmdir [OPTIONS] [DIRECTORY]

Example: rmdir content

**PROGRAM 5**

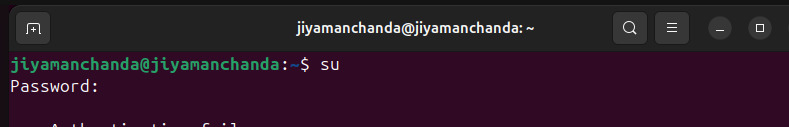
**PROGRAM 5:- Implement the basic and user status commands like:**

su, sudo, man, help, history, who, whoami, id, uname, uptime, free, tty, cal, date, hostname, reboot, clear

**Solution:=**

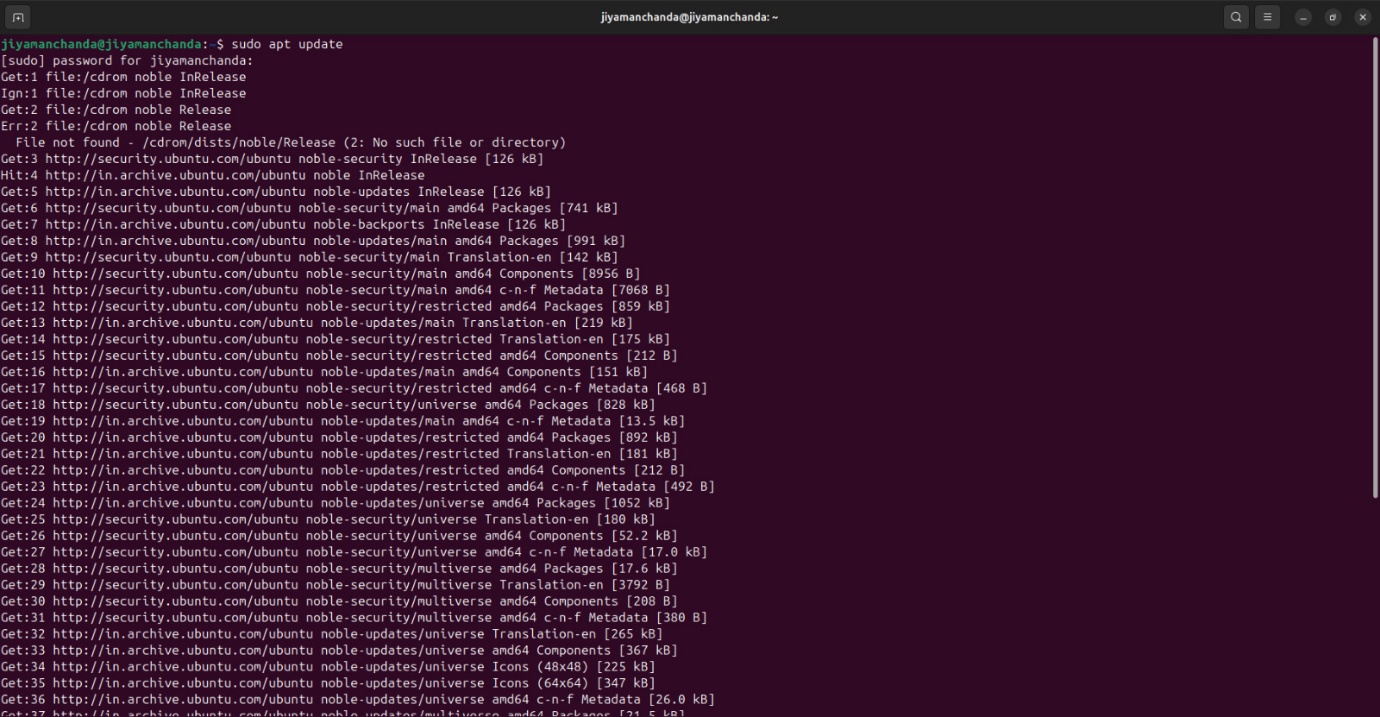
**1. su**

**Switch user to root or another account**  
✔ **Usage:** su [username]  
✔ **Example:** su root



**2. sudo**

**Execute a command as superuser**  
✔ **Usage:** sudo [command]  
✔ **Example:** sudo apt update



**3. man**

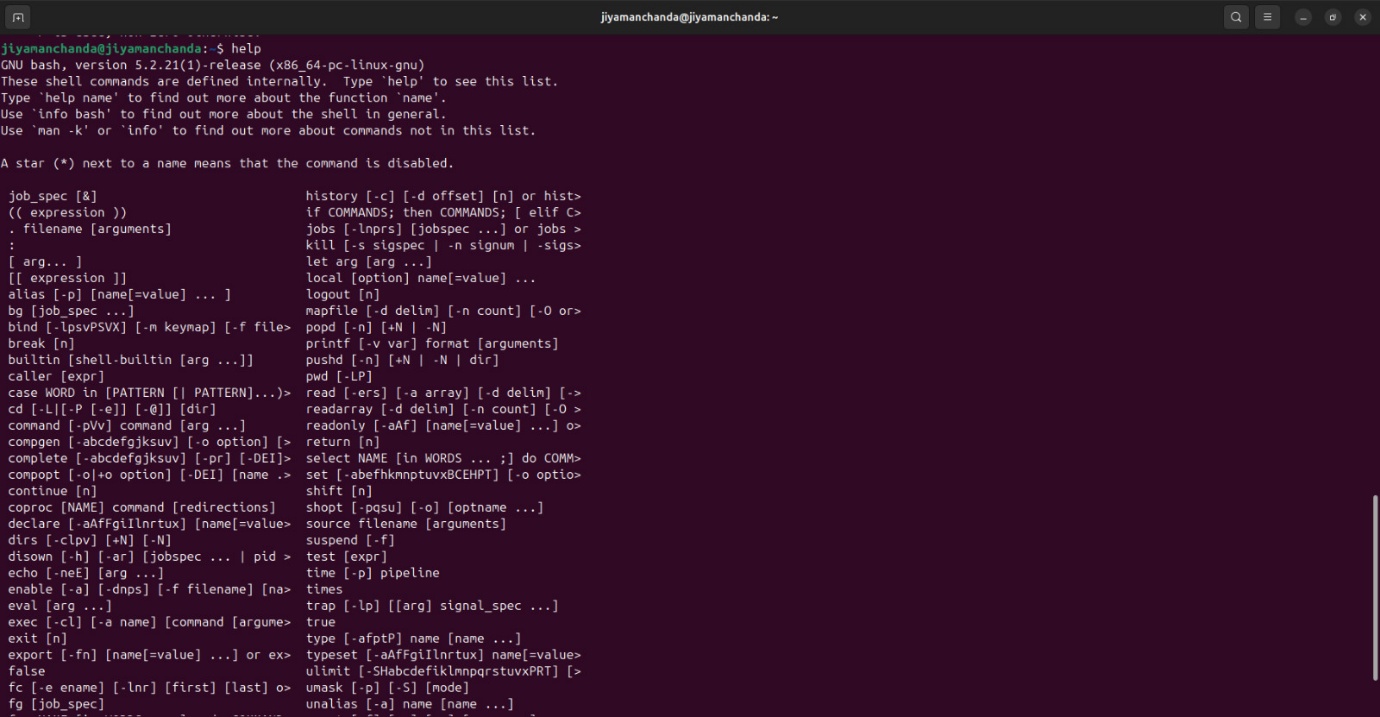
**Displays the manual pages of a command**  
✔ **Usage:** man [command]  
✔ **Example:** man ls

Use q to quit the manual page.



**4. help**

**Provides help for shell built-in commands**  
✔ **Usage:** help [command]  
✔ **Example:** help cd



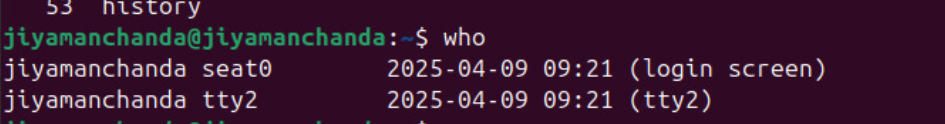
**5. history**

**Shows previously used commands**  
✔ **Usage:** history



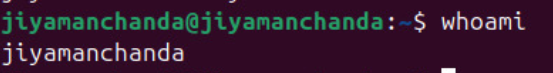
**6. who**

**Displays who is logged into the system**  
✔ **Usage:** who



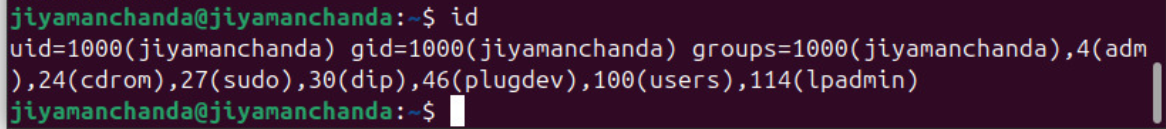
**7. whoami**

**Shows the current logged-in user**  
✔ **Usage:** whoami



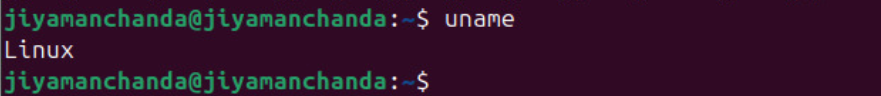
**8. id**

**Displays user ID and group ID**  
✔ **Usage:** id



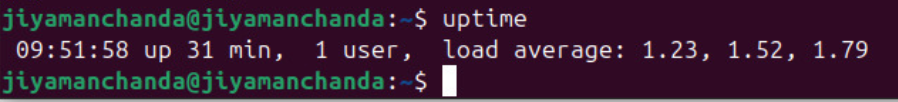
**9. uname**

**Displays system information**  
✔ **Usage:** uname -a  
✔ **Example:** Shows kernel, OS, etc.



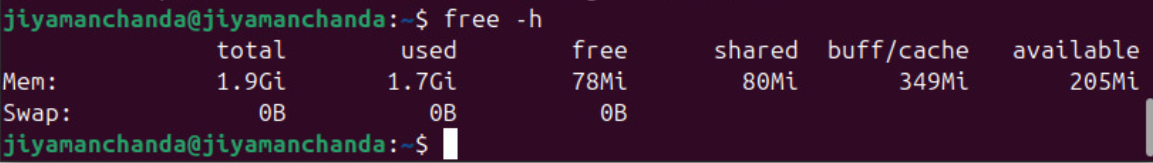
**10. uptime**

**Shows how long the system has been running**  
✔ **Usage:** uptime



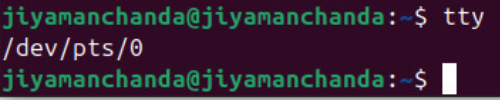
**11. free**

**Displays memory usage**  
✔ **Usage:** free -h



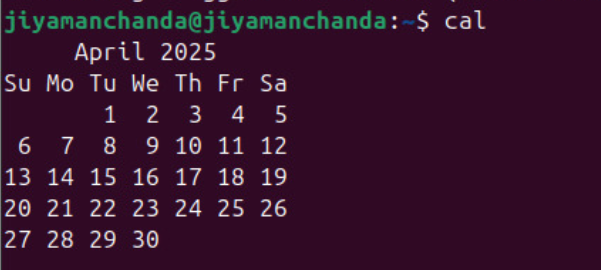
**12. tty**

**Prints the terminal type**  
✔ **Usage:** tty



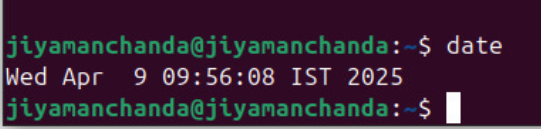
**13. cal**

**Displays the calendar**  
✔ **Usage:** cal



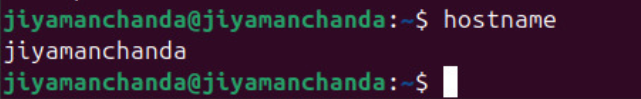
**14. date**

**Displays or sets the date and time**  
✔ **Usage:** date



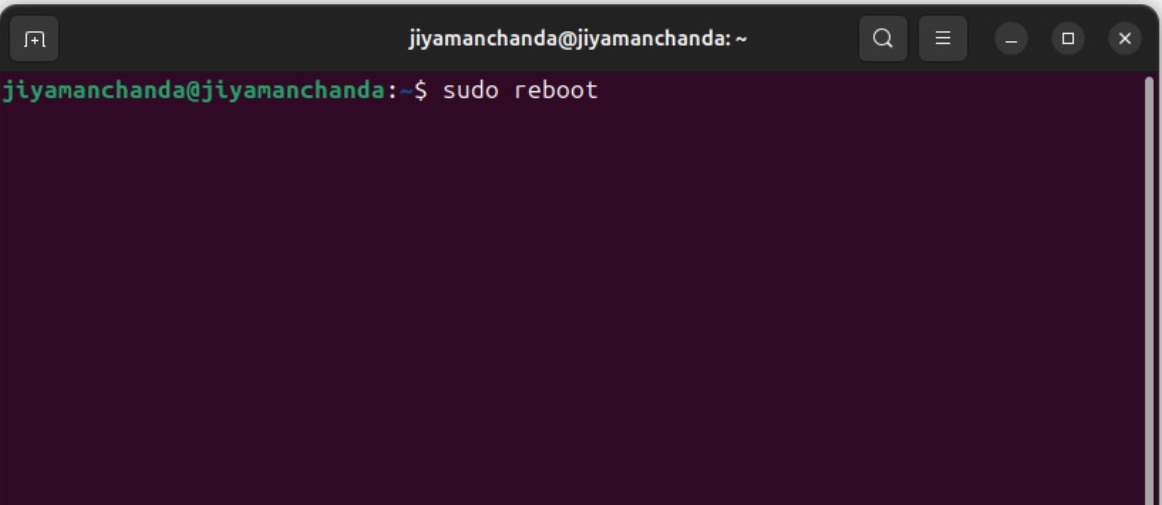
**15. hostname**

**Displays or sets the system’s hostname**  
✔ **Usage:** hostname



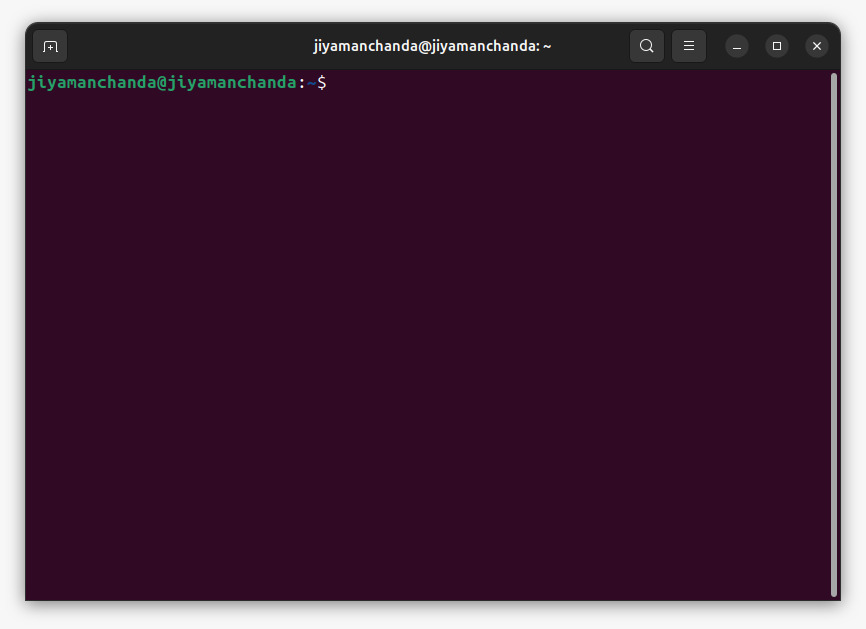
**16. reboot**

**Restarts the system**  
✔ **Usage:** sudo reboot



**17. clear**

**Clears the terminal screen**  
✔ **Usage:** clear



Top of Form

Bottom of Form

**Aim:** Write a program to implement process concepts using C language by printing process Id.

## Procedure :

To demonstrate process concepts using C language and print the process ID (PID), we can create a simple program that prints the PID of the current process.

CODE:

#include <stdio.h>

#include <unistd.h>

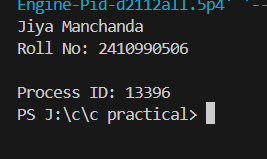
int main() {

printf("Process ID: %d\n", getpid());

return 0;

}

OUTPUT



# EXPERIMENT 8

**Aim:** Write a C program to implement FCFS (First Come First Serve) and SJF (Shortest Job First) scheduling algorithms.

## Theory:

### FCFS (First Come First Serve) Scheduling Algorithm:

FCFS is a non-preemptive scheduling algorithm where processes are executed in the order they arrive.

## Procedure :

To demonstrate the FCFS (First Come First Serve) scheduling algorithm in a C program, we'll simulate a simple job scheduling system where processes arrive and are scheduled based on these algorithms. We'll use basic data structures and functions to represent processes and perform scheduling.

CODE:

#include <stdio.h>

int main() {

int n, i;

int bt[20], wt[20], tat[20];

float avg\_wt = 0, avg\_tat = 0;

printf("Enter number of processes: ");

scanf("%d", &n);

printf("Enter Burst Time for each process:\n");

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

printf("P%d: ", i + 1);

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

}

wt[0] = 0;

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

wt[i] = wt[i - 1] + bt[i - 1];

}

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

tat[i] = bt[i] + wt[i];

avg\_wt += wt[i];

avg\_tat += tat[i];

}

printf("\nProcess\tBT\tWT\tTAT\n");

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

printf("P%d\t%d\t%d\t%d\n", i + 1, bt[i], wt[i], tat[i]);

}

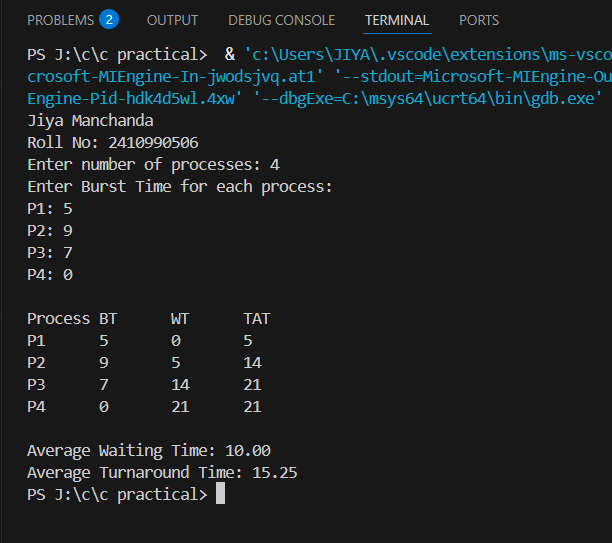
printf("\nAverage Waiting Time: %.2f", avg\_wt / n);

printf("\nAverage Turnaround Time: %.2f\n", avg\_tat / n);

return 0;

}

OUTPUT:



**Theory:**

### SJF (Shortest Job First) Scheduling Algorithm:

SJF is a scheduling algorithm that selects the shortest job (process) available in the ready queue for execution.

## Procedure :

To demonstrate the SJF (Shortest Job First) scheduling algorithms in a C program, we'll simulate a simple job scheduling system where processes arrive and are scheduled based on these algorithms. We'll use basic data structures and functions to represent processes and perform scheduling.

# Code:

# #include <stdio.h>

# int main() {

# int n, bt[20], p[20], wt[20], tat[20], i, j, temp;

# float avg\_wt = 0, avg\_tat = 0;

# printf("Enter number of processes: ");

# scanf("%d", &n);

# printf("Enter Burst Time for each process:\n");

# for(i = 0; i < n; i++) {

# printf("P%d: ", i + 1);

# scanf("%d", &bt[i]);

# p[i] = i + 1;

# }

# for(i = 0; i < n - 1; i++) {

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

# if(bt[i] > bt[j]) {

# temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;

# temp = p[i]; p[i] = p[j]; p[j] = temp;

# }

# }

# }

# wt[0] = 0;

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

# wt[i] = wt[i - 1] + bt[i - 1];

# }

# for(i = 0; i < n; i++) {

# tat[i] = bt[i] + wt[i];

# avg\_wt += wt[i];

# avg\_tat += tat[i];

# }

# printf("\nProcess\tBT\tWT\tTAT\n");

# for(i = 0; i < n; i++) {

# printf("P%d\t%d\t%d\t%d\n", p[i], bt[i], wt[i], tat[i]);

# }

# printf("\nAverage Waiting Time: %.2f", avg\_wt / n);

# printf("\nAverage Turnaround Time: %.2f\n", avg\_tat / n);

# return 0;

# }

# OUTPUT

# 

# EXPERIMENT 9

**Aim:** Write a C program to implement priority scheduling and RR(Round Robin) scheduling algorithms.

## Theory:

### Priority Scheduling Algorithm

Priority Scheduling is a non-preemptive scheduling algorithm where processes are scheduled based on priority. A process with a higher priority (lower numerical value) is executed first.

## Procedure:

To implement Priority Scheduling Scheduling algorithm in a C program, we'll create a simple simulation where processes are scheduled based on their priority or using a round-robin time-sharing mechanism. We'll define process structures, implement the scheduling algorithms, and simulate their execution.

**Code**

#include <stdio.h>

int main() {

int n, i, j, bt[20], p[20], pr[20], wt[20], tat[20], temp;

float avg\_wt = 0, avg\_tat = 0;

printf("Enter number of processes: ");

scanf("%d", &n);

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

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

scanf("%d %d", &bt[i], &pr[i]);

p[i] = i + 1;

}

for(i = 0; i < n - 1; i++) {

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

if(pr[i] > pr[j]) {

temp = pr[i]; pr[i] = pr[j]; pr[j] = temp;

temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;

temp = p[i]; p[i] = p[j]; p[j] = temp;

}

}

}

wt[0] = 0;

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

wt[i] = wt[i - 1] + bt[i - 1];

}

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

tat[i] = wt[i] + bt[i];

avg\_wt += wt[i];

avg\_tat += tat[i];

}

printf("\nProcess\tBT\tPriority\tWT\tTAT\n");

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

printf("P%d\t%d\t%d\t\t%d\t%d\n", p[i], bt[i], pr[i], wt[i], tat[i]);

}

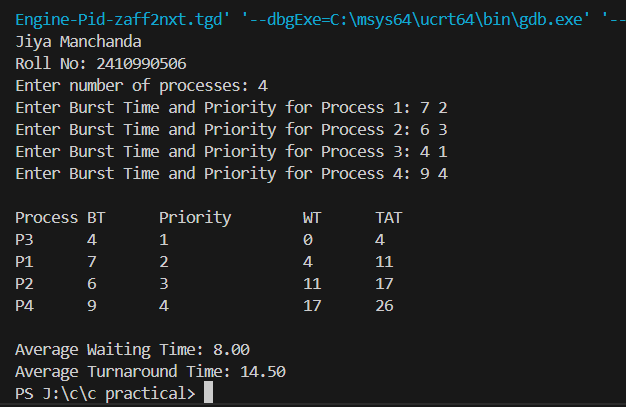
printf("\nAverage Waiting Time: %.2f", avg\_wt / n);

printf("\nAverage Turnaround Time: %.2f\n", avg\_tat / n);

return 0;

}

Output



## Theory:

### Round-Robin (RR) Scheduling Algorithm

Round-Robin Scheduling is a preemptive scheduling algorithm where each process is assigned a fixed time slice (time quantum). If a process does not finish within the time slice, it's placed at the end of the ready queue.

## Procedure:

To implement Round-Robin (RR) Scheduling algorithms in a C program, we'll create a simple simulation where processes are scheduled using a round-robin time-sharing mechanism. We'll define process structures, implement the scheduling algorithms, and simulate their execution.

Code

#include <stdio.h>

int main() {

int n, i, j, qt, time = 0, rem\_bt[20], bt[20], wt[20] = {0}, tat[20] = {0};

float avg\_wt = 0, avg\_tat = 0;

printf("Enter number of processes: ");

scanf("%d", &n);

printf("Enter Burst Time for each process:\n");

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

printf("P%d: ", i + 1);

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

rem\_bt[i] = bt[i];

}

printf("Enter Time Quantum: ");

scanf("%d", &qt);

while(1) {

int done = 1;

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

if(rem\_bt[i] > 0) {

done = 0;

if(rem\_bt[i] > qt) {

time += qt;

rem\_bt[i] -= qt;

} else {

time += rem\_bt[i];

wt[i] = time - bt[i];

rem\_bt[i] = 0;

}

}

}

if(done) break;

}

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

tat[i] = bt[i] + wt[i];

avg\_wt += wt[i];

avg\_tat += tat[i];

}

printf("\nProcess\tBT\tWT\tTAT\n");

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

printf("P%d\t%d\t%d\t%d\n", i + 1, bt[i], wt[i], tat[i]);

}

printf("\nAverage Waiting Time: %.2f", avg\_wt / n);

printf("\nAverage Turnaround Time: %.2f\n", avg\_tat / n);

return 0;

}

Output

