National Textile University

Department of Computer Science

Subject:

Operating system

Submitted To:

Sir Nasir

Submitted By:

Noor ul Ain

Registration No:

23-NTU-CS-1221

Assignment No:

1

Semester:

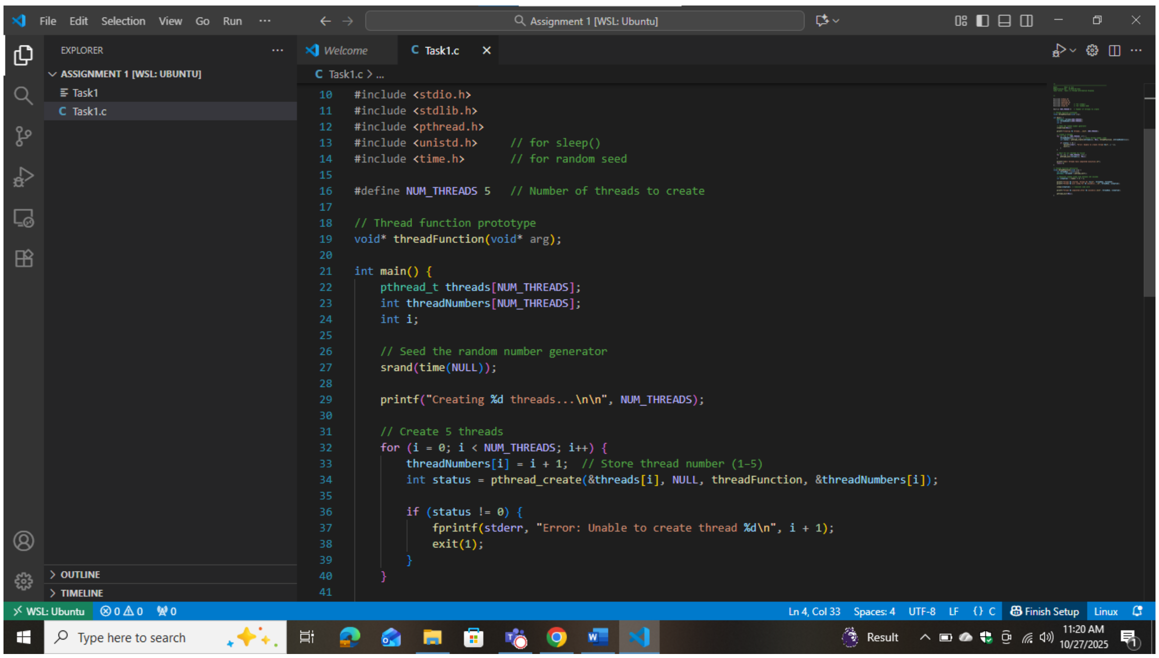
5

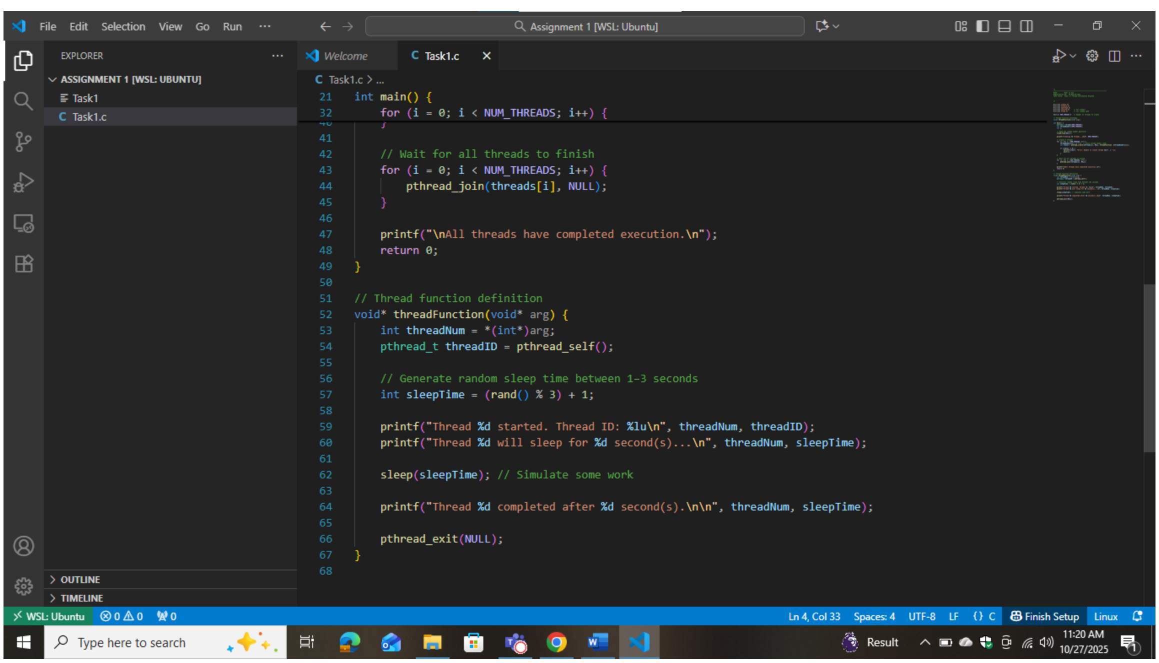
Task 1:

Write a program that creates 5 threads. Each thread should:

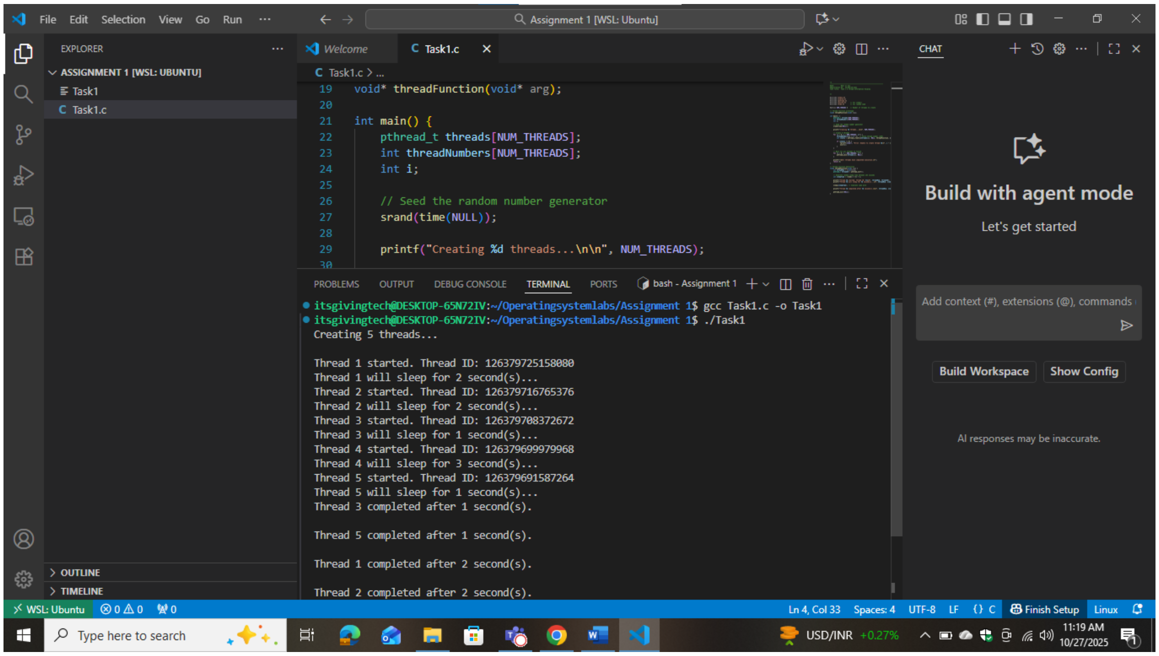
- Print its thread ID using `pthread_self()`.
- Display its thread number (1st, 2nd, etc.).
- Sleep for a random time between 1–3 seconds.
- Print a completion message before exiting.

Code:





Output:

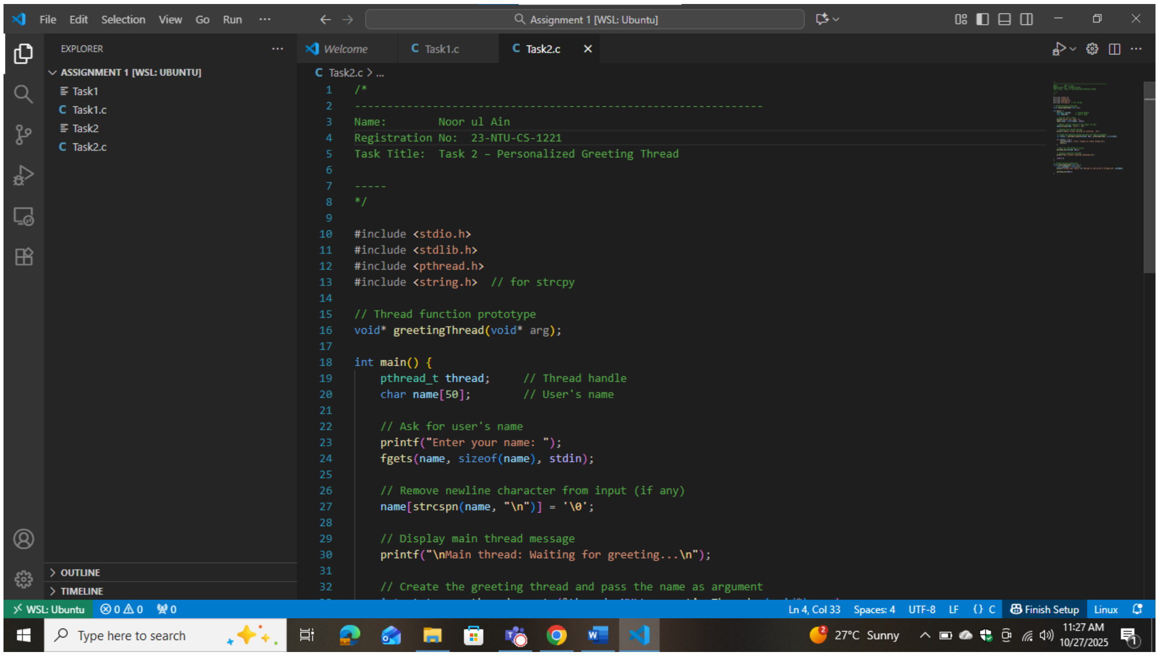


Task 2:

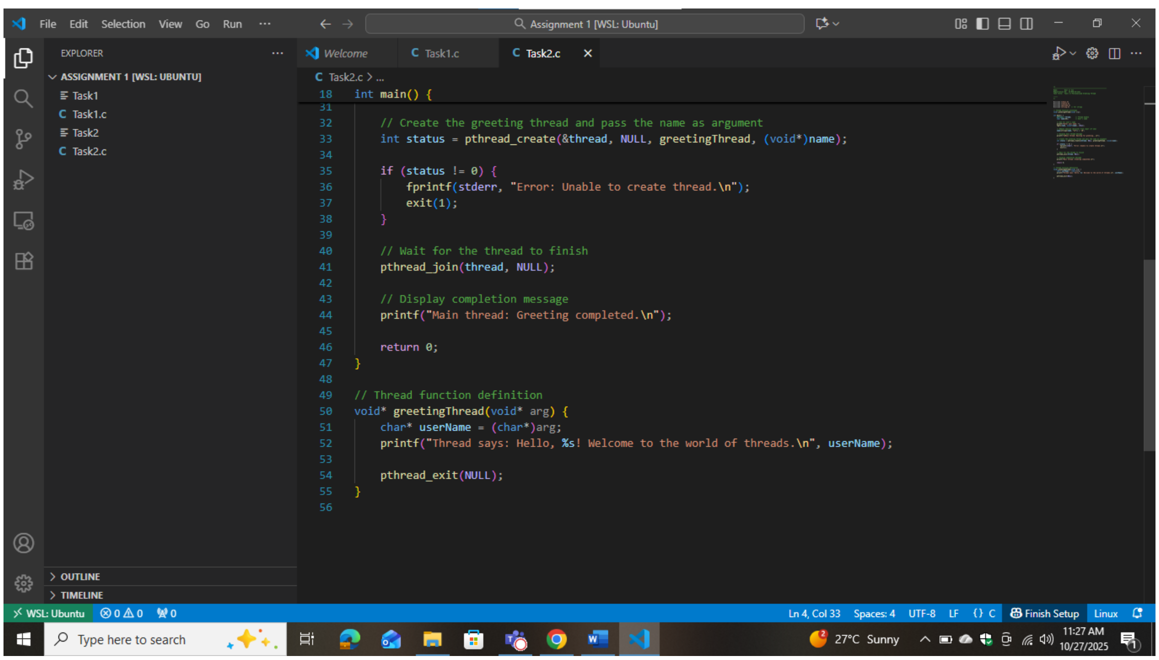
Write a C program that:

- Creates a thread that prints a personalized greeting message.
- The message includes the user's name passed as an argument to the thread.
- The main thread prints "Main thread: Waiting for greeting…" before joining the created thread.

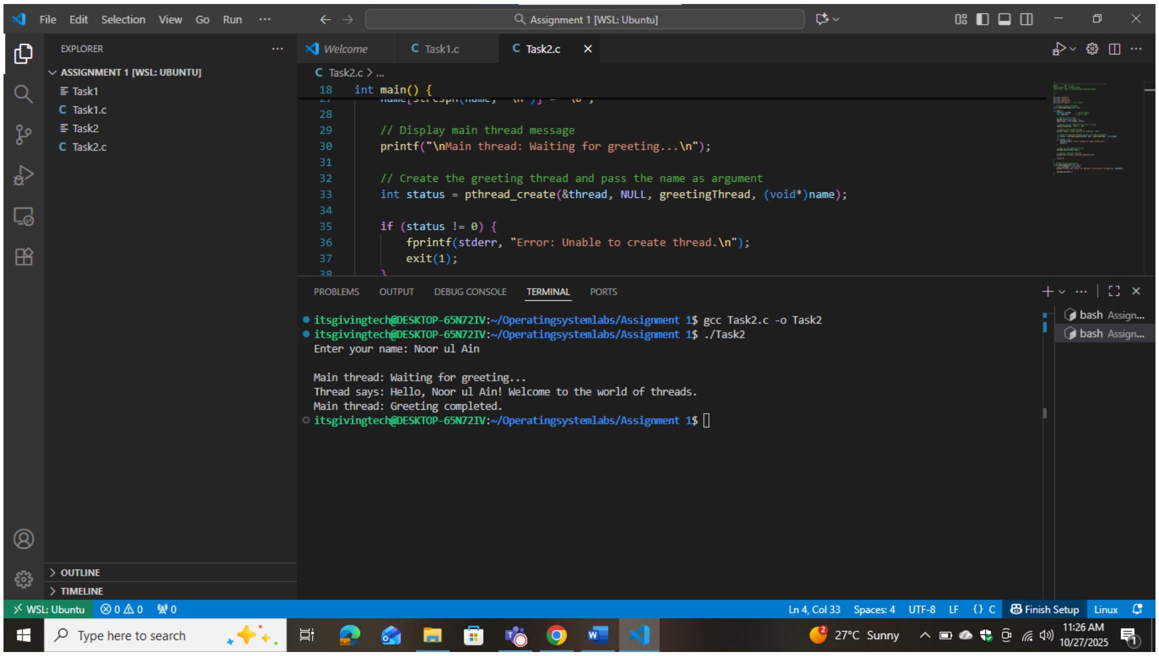
Code:


/*

Name: Noor ul Ain
Registration No: 23-NTU-CS-1221
Task Title: Task 2 – Personalized Greeting Thread
-----
*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h> // for strcpy
// Thread function prototype
void* greetingThread(void* arg);
int main() {
pthread_t thread;   // Thread handle
char name[50];   // User's name
// Ask for user's name
printf("Enter your name: ");
fgets(name, sizeof(name), stdin);
// Remove newline character from input (if any)
name[strcspn(name, "\n")] = '\0';
// Display main thread message
printf("\nMain thread: Waiting for greeting...\n");
// Create the greeting thread and pass the name as argument


int main() {
// Create the greeting thread and pass the name as argument
int status = pthread_create(&thread, NULL, greetingThread, (void*)name);
if (status != 0) {
fprintf(stderr, "Error: Unable to create thread.\n");
exit(1);
}
// Wait for the thread to finish
pthread_join(thread, NULL);
// Display completion message
printf("Main thread: Greeting completed.\n");
return 0;
}
// Thread function definition
void* greetingThread(void* arg) {
char* userName = (char*)arg;
printf("Thread says: Hello, %s! Welcome to the world of threads.\n", userName);
pthread_exit(NULL);
}

output:

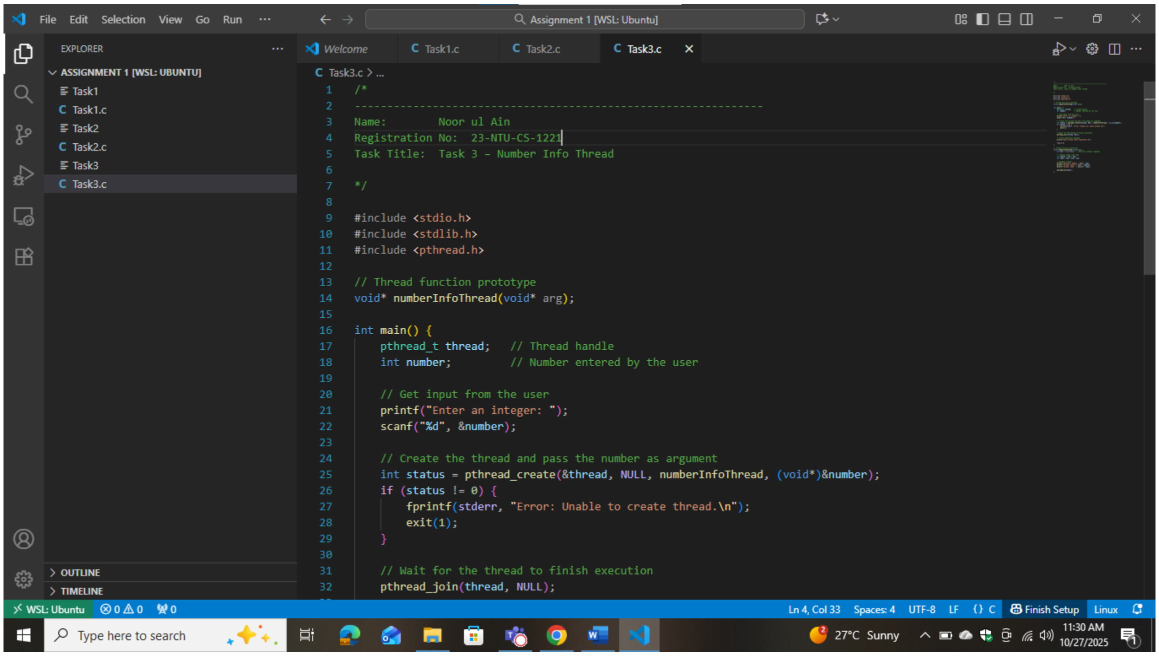


Task 3:

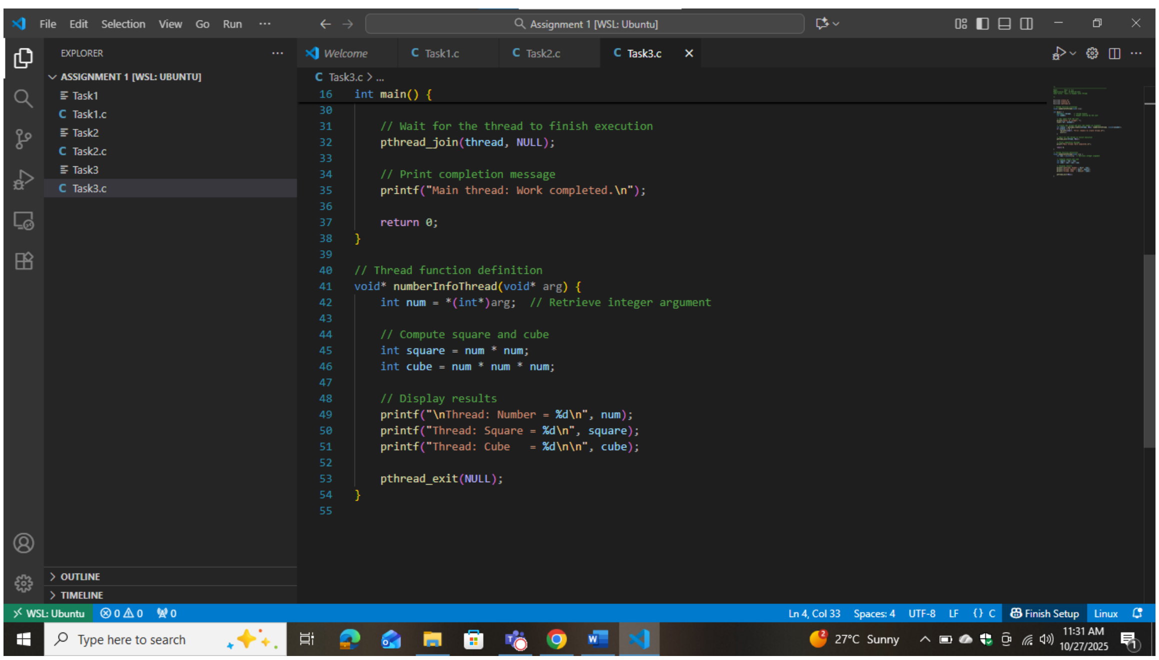
Write a program that:

- Takes an integer input from the user.
- Creates a thread and passes this integer to it.
- The thread prints the number, its square, and cube.
- The main thread waits until completion and prints “Main thread: Work completed.”

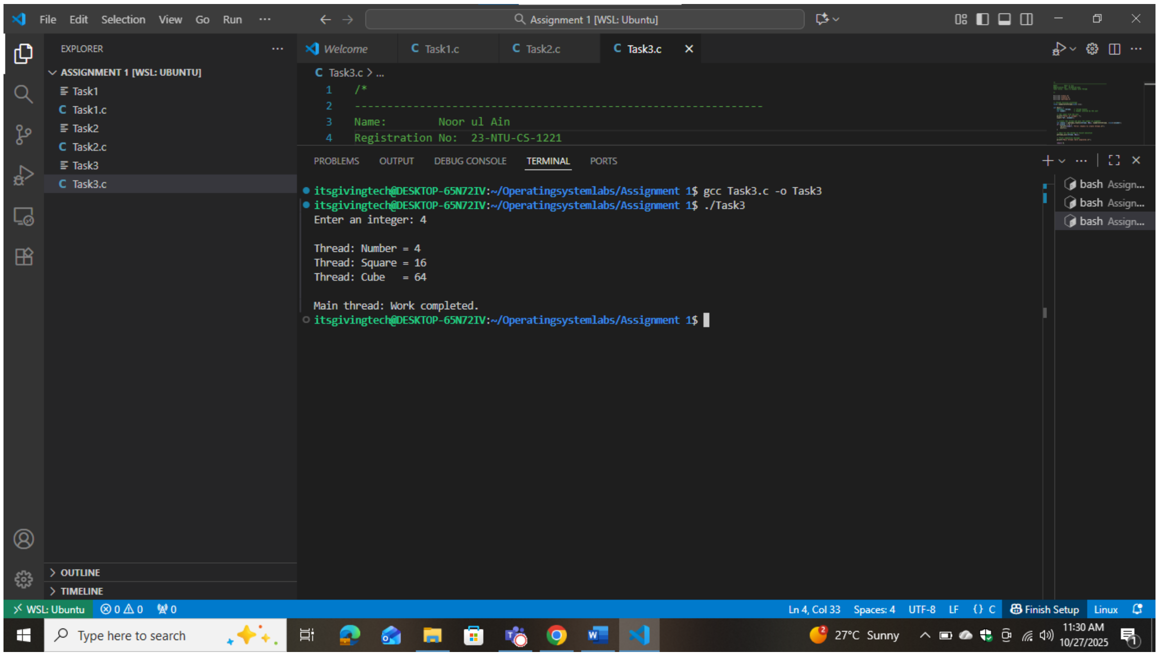
Code:


Assignment 1 [WSL: Ubuntu]
EXPLORER
ASSIGNMENT 1 [WSL: UBUNTU]
Task1
Task1.c
Task2
Task2.c
Task3
Task3.c
Welcome
Task3.c
/*

Name: Noor ul Ain
Registration No: 23-NTU-CS-1221
Task Title: Task 3 - Number Info Thread
*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
// Thread function prototype
void* numberInfoThread(void* arg);
int main() {
pthread_t thread;   // Thread handle
int number;   // Number entered by the user
// Get input from the user
printf("Enter an integer: ");
scanf("%d", &number);
// Create the thread and pass the number as argument
int status = pthread_create(&thread, NULL, numberInfoThread, (void*)&number);
if (status != 0) {
fprintf(stderr, "Error: Unable to create thread.\n");
exit(1);
}
// Wait for the thread to finish execution
pthread_join(thread, NULL);
OUTLINE
TIMELINE
WSL: Ubuntu
Ln 4, Col 33 Spaces: 4 UTF-8 LF C Finish Setup Linux
Type here to search
27°C Sunny
11:30 AM
10/27/2025


Assignment 1 [WSL: Ubuntu]
EXPLORER
ASSIGNMENT 1 [WSL: UBUNTU]
Task1
Task1.c
Task2
Task2.c
Task3
Task3.c
Welcome
Task3.c
int main() {
// Wait for the thread to finish execution
pthread_join(thread, NULL);
// Print completion message
printf("Main thread: Work completed.\n");
return 0;
}
// Thread function definition
void* numberInfoThread(void* arg) {
int num = *(int*)arg; // Retrieve integer argument
// Compute square and cube
int square = num * num;
int cube = num * num * num;
// Display results
printf("\nThread: Number = %d\n", num);
printf("Thread: Square = %d\n", square);
printf("Thread: Cube   = %d\n\n", cube);
pthread_exit(NULL);
}
OUTLINE
TIMELINE
WSL: Ubuntu
Ln 4, Col 33 Spaces: 4 UTF-8 LF C Finish Setup Linux
Type here to search
27°C Sunny
11:31 AM
10/27/2025

Output:

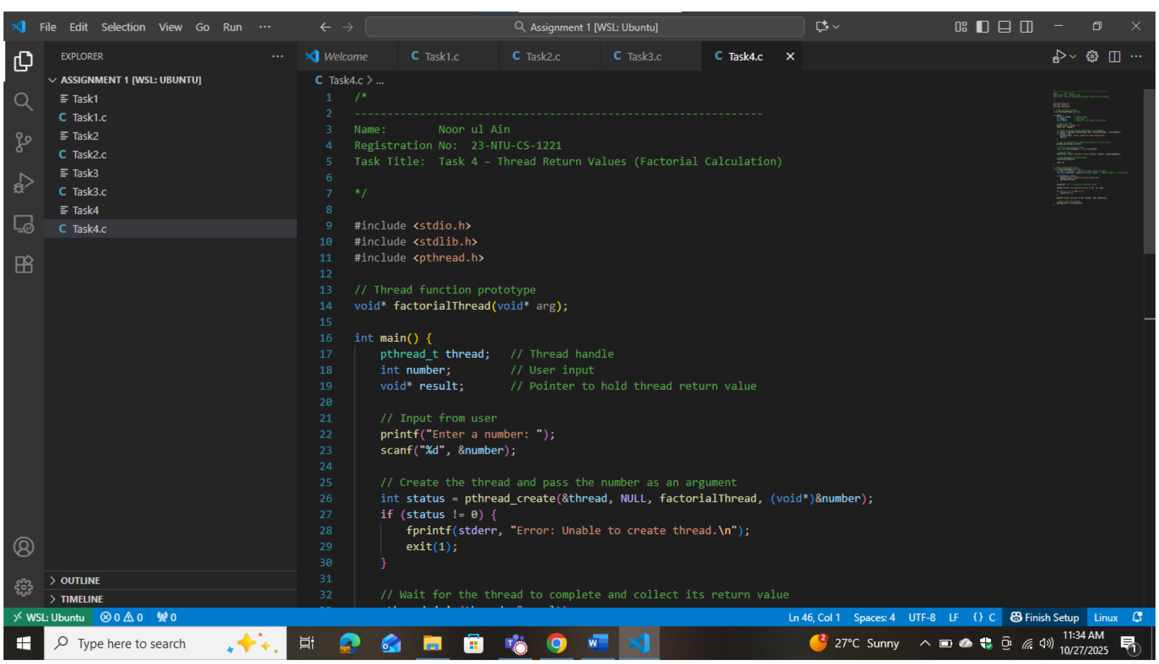


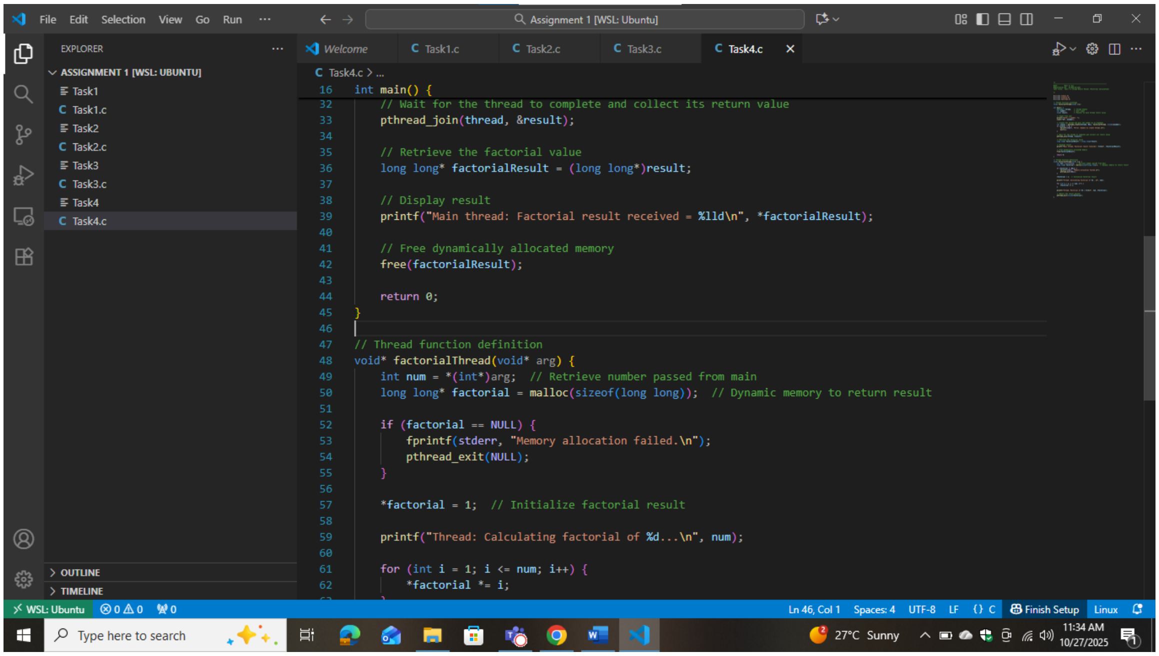
Task4:

Write a program that creates a thread to compute the factorial of a number entered by the user.

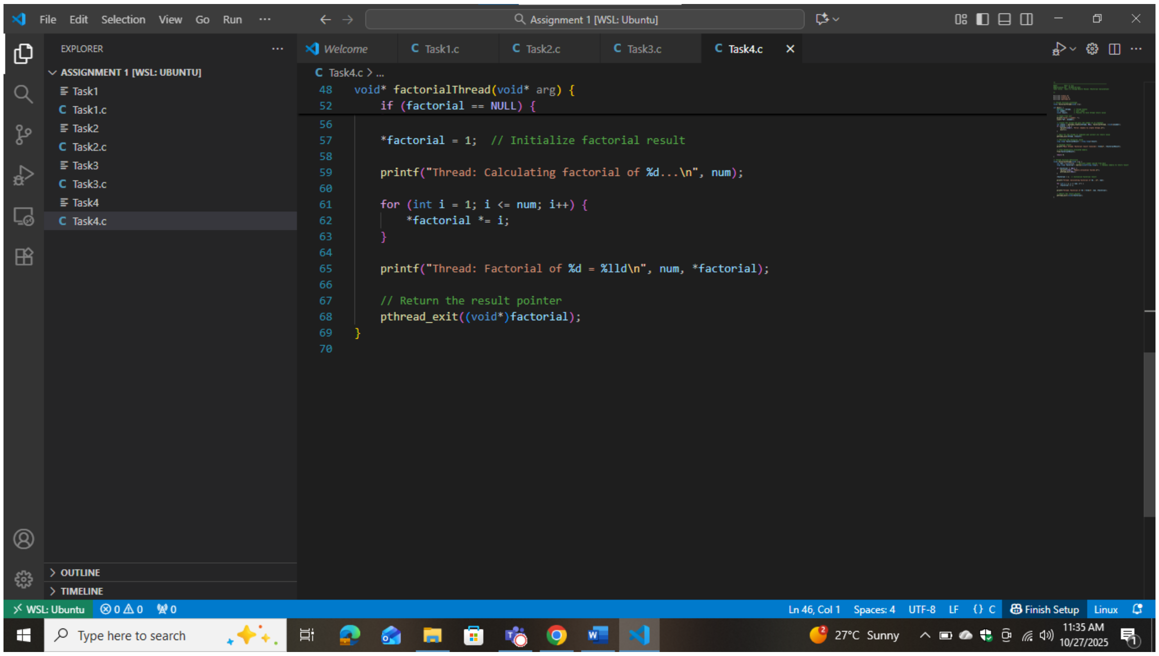
- The thread should return the result using a pointer.
- The main thread prints the result after joining.

Code:

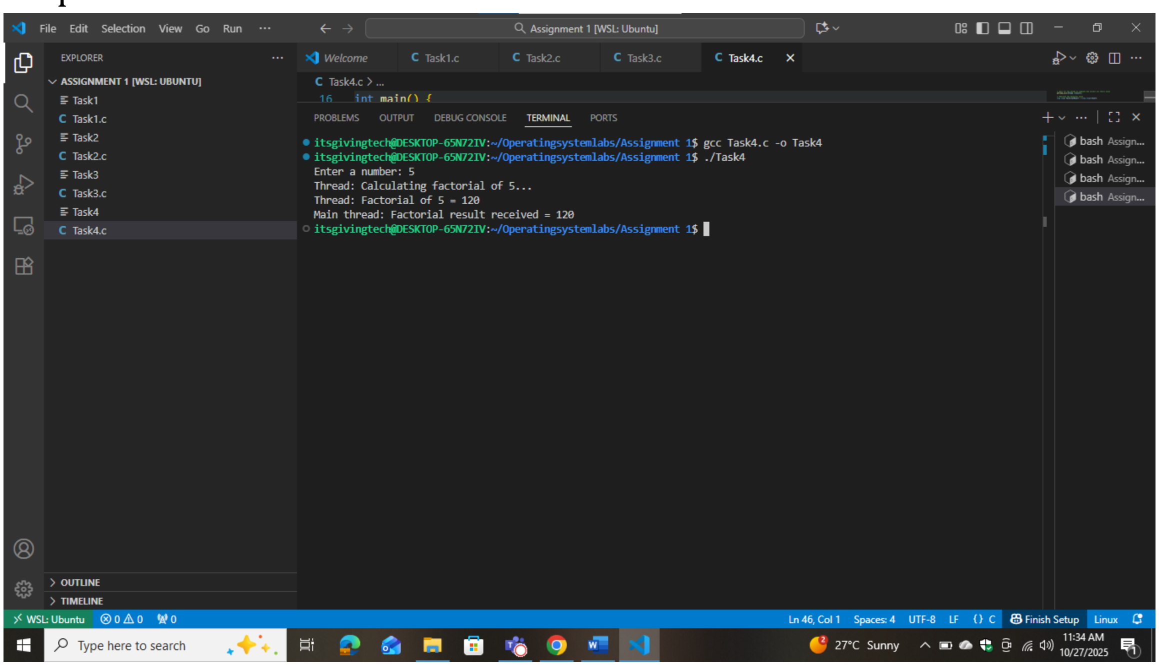




File Edit Selection View Go Run
Assignment 1 [WSL: Ubuntu]
EXPLORER
ASSIGNMENT 1 [WSL: UBUNTU]
Task1
Task1.c
Task2
Task2.c
Task3
Task3.c
Task4
Task4.c
Welcome
Task1.c
Task2.c
Task3.c
Task4.c
int main() {
// Wait for the thread to complete and collect its return value
pthread_join(thread, &result);
// Retrieve the factorial value
long long* factorialResult = (long long*)result;
// Display result
printf("Main thread: Factorial result received = %lld\n", *factorialResult);
// Free dynamically allocated memory
free(factorialResult);
return 0;
}
// Thread function definition
void* factorialThread(void* arg) {
int num = *(int*)arg; // Retrieve number passed from main
long long* factorial = malloc(sizeof(long long)); // Dynamic memory to return result
if (factorial == NULL) {
fprintf(stderr, "Memory allocation failed.\n");
pthread_exit(NULL);
}
*factorial = 1; // Initialize factorial result
printf("Thread: Calculating factorial of %d...\n", num);
for (int i = 1; i <= num; i++) {
*factorial *= i;
OUTLINE
TIMELINE
WSL: Ubuntu
Ln 46, Col 1
Spaces: 4
UTF-8
LF
C
Finish Setup
Linux
Type here to search
27°C Sunny
11:34 AM
10/27/2025

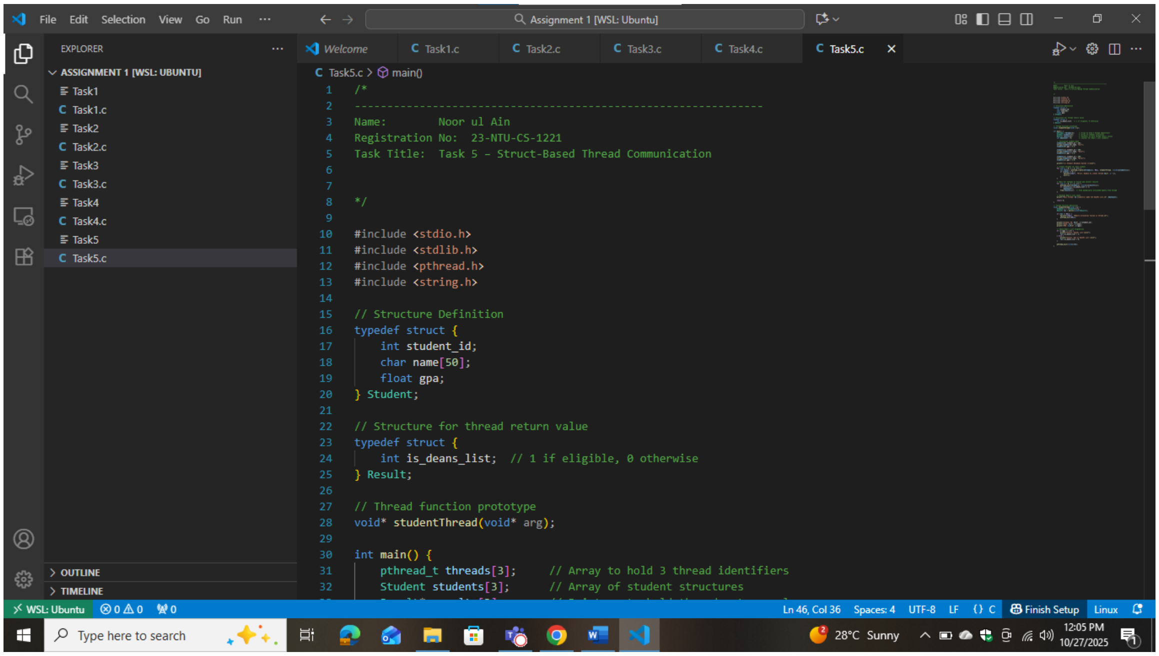


output:

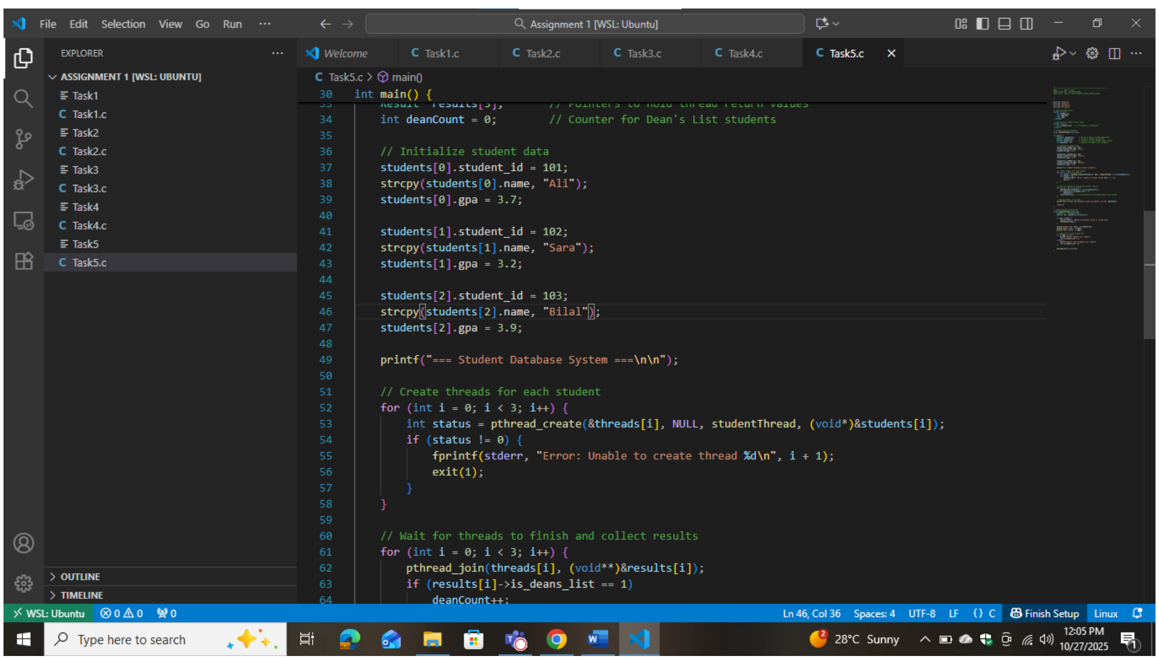


Task 5:

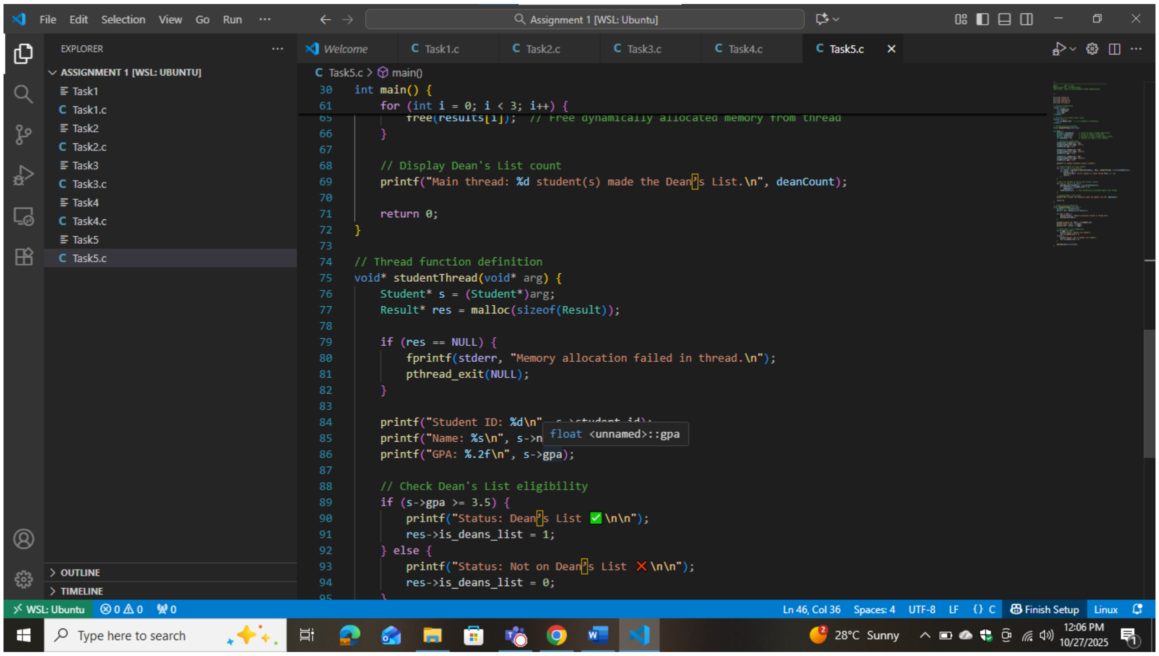
Code:



```c
/*
----------------------------------------------------------------
Name:        Noor ul Ain
Registration No:  23-NTU-CS-1221
Task Title:  Task 5 - Struct-Based Thread Communication


*/

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

// Structure Definition
typedef struct {
    int student_id;
    char name[50];
    float gpa;
} Student;

// Structure for thread return value
typedef struct {
    int is_deans_list;  // 1 if eligible, 0 otherwise
} Result;

// Thread function prototype
void* studentThread(void* arg);

int main() {
    pthread_t threads[3];     // Array to hold 3 thread identifiers
    Student students[3];      // Array of student structures
```

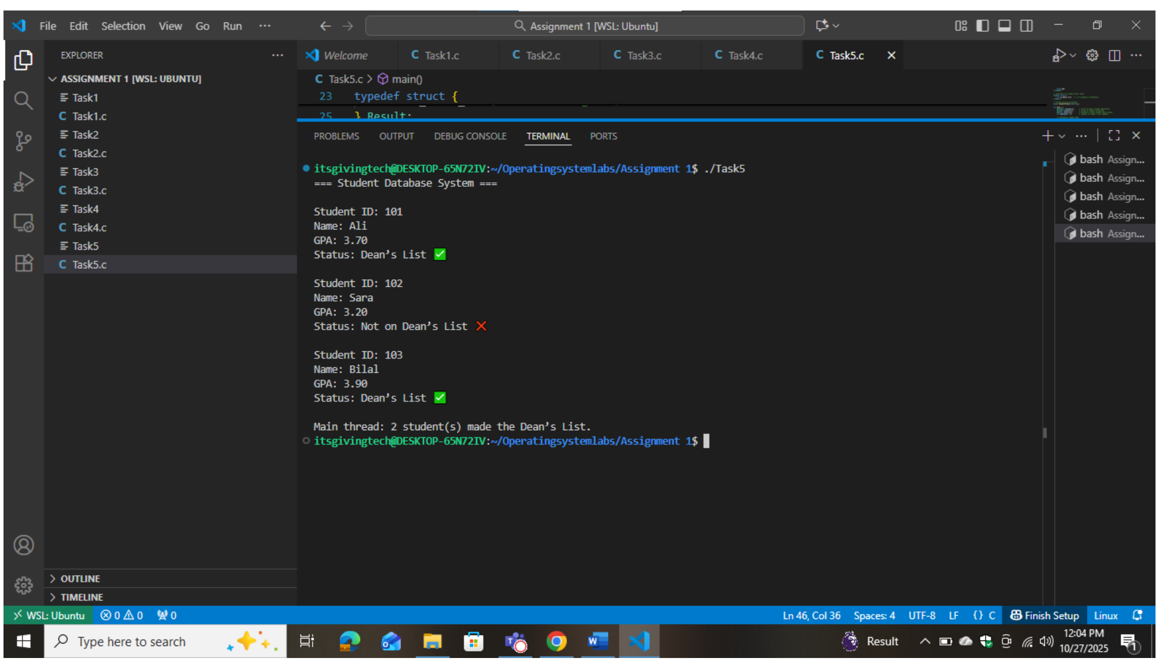


```c
int main() {
    int deanCount = 0;        // Counter for Dean's List students

    // Initialize student data
    students[0].student_id = 101;
    strcpy(students[0].name, "Ali");
    students[0].gpa = 3.7;

    students[1].student_id = 102;
    strcpy(students[1].name, "Sara");
    students[1].gpa = 3.2;

    students[2].student_id = 103;
    strcpy(students[2].name, "Bilal");
    students[2].gpa = 3.9;

    printf("=== Student Database System ===\n\n");

    // Create threads for each student
    for (int i = 0; i < 3; i++) {
        int status = pthread_create(&threads[i], NULL, studentThread, (void*)&students[i]);
        if (status != 0) {
            fprintf(stderr, "Error: Unable to create thread %d\n", i + 1);
            exit(1);
        }
    }

    // Wait for threads to finish and collect results
    for (int i = 0; i < 3; i++) {
        pthread_join(threads[i], (void**)&results[i]);
        if (results[i]->is_deans_list == 1)
            deanCount++;
```



Assignment 1 [WSL: Ubuntu]
EXPLORER
ASSIGNMENT 1 [WSL: UBUNTU]
Task1
Task1.c
Task2
Task2.c
Task3
Task3.c
Task4
Task4.c
Task5
Task5.c
Welcome
Task5.c
int main() {
for (int i = 0; i < 3; i++) {
}
// Display Dean's List count
printf("Main thread: %d student(s) made the Dean's List.\n", deanCount);
return 0;
}
// Thread function definition
void* studentThread(void* arg) {
Student* s = (Student*)arg;
Result* res = malloc(sizeof(Result));
if (res == NULL) {
fprintf(stderr, "Memory allocation failed in thread.\n");
pthread_exit(NULL);
}
printf("Student ID: %d\n"
printf("Name: %s\n", s->n
printf("GPA: %.2f\n", s->gpa);
float <unnamed>::gpa
// Check Dean's List eligibility
if (s->gpa >= 3.5) {
printf("Status: Dean's List ✅\n\n");
res->is_deans_list = 1;
} else {
printf("Status: Not on Dean's List ❌\n\n");
res->is_deans_list = 0;
OUTLINE
TIMELINE
WSL: Ubuntu
Ln 46, Col 36
Spaces: 4
UTF-8
LF
C
Finish Setup
Linux
Type here to search
28°C Sunny
12:06 PM
10/27/2025

Output:



Assignment 1 [WSL: Ubuntu]
typedef struct {
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
itsgivingtech@DESKTOP-65N72IV:~/Operatingsystemlabs/Assignment 1$ ./Task5
=== Student Database System ===
Student ID: 101
Name: Ali
GPA: 3.70
Status: Dean's List ✅
Student ID: 102
Name: Sara
GPA: 3.20
Status: Not on Dean's List ❌
Student ID: 103
Name: Bilal
GPA: 3.90
Status: Dean's List ✅
Main thread: 2 student(s) made the Dean's List.
itsgivingtech@DESKTOP-65N72IV:~/Operatingsystemlabs/Assignment 1$
bash Assign...
Type here to search
Result
12:04 PM
10/27/2025

Section-B:

Short Questions

1. **Define an Operating System in a single line.**
   Answer: An Operating System (OS) is system software that manages computer hardware and software resources and provides services to users and programs. It acts as an intermediary layer between system's hardware and user.
2. **What is the primary function of the CPU scheduler?**
   Answer:
   It decides which process, present in the ready queue, should run next by the CPU.
3. **List any three states of a process.**
   Answer:
   - Ready
   - Running
   - Waiting(blocked)
4. **What is meant by a Process Control Block (PCB)?**
   Answer:
   A PCB is a data structure that stores necessary information about the process such as its process ID, process state, it' s registers, memory details and accounting data.

5. **Differentiate between a process and a program.**
   Answer:

Program	Process
Collection of instructions stored on disk.	Program which is in active execution in memory.
It does not consume any memory	It consumes system resources like memory and space
Example: Notepad.exe file	Notepad running on screen

6. **What do you understand by context switching?**

Answer:

It is the process in which CPU saves the state of one process (registers, program counters etc.) and loads the state of another to switch the CPU between them. This allows multiple processes to share the CPU effectively.

7. **Define CPU utilization and throughput.**

Answer:

• **CPU Utilization:** It measures how much of the CPU's time is actively used to execute the process or simply

Percentage of time the CPU is busy.

• **Throughput:** It is the number of processes completed per unit time.

It indicates how productive the system is.

8. **What is the turnaround time of a process?**

Answer:

The amount of time taken from the process submission to the process completion.
Turnaround Time = Completion Time − Arrival Time.

9. **How is waiting time calculated in process scheduling?**

Answer:

Waiting time is the total time a process spends waiting in the ready queue.
Waiting Time = Turnaround Time − Burst Time

10. **Define response time in CPU scheduling.**

Answer:

It is the time from when a request is submitted until the first response is produced.
It measures how quickly reacts to user's input.

11. **What is preemptive scheduling?**

Answer:

A scheduling method where the CPU can be taken away from a process before it finishes if a high priority of time critical process arrives.

Example:

Example: **Round Robin** or **Priority Scheduling.**

12. **What is non-preemptive scheduling?**

Answer:

It is the type of scheduling in which once a process starts execution, it cannot be stopped until it finishes or voluntarily releases the CPU.
Example: **FCFS** or **SJF (non-preemptive)**.

13. **State any two advantages of the Round Robin scheduling algorithm.**

Answer:

• In this scheduling every process gets equal CPU time (fairness).

- It provides good response time in interactive systems.

14. Mention one major drawback of the Shortest Job First (SJF) algorithm.

Answer:

SJF may cause **starvation** because long processes can be delayed indefinitely if short jobs keep arriving.

15. Define CPU idle time.

Answer:

CPU idle time is the duration when the CPU is not executing any process due to an empty ready queue or waiting for I/O operations.

16. State two common goals of CPU scheduling algorithms.

Answer:

1. **Maximize CPU efficiency** It minimizes the idle time.
2. **Reduce average waiting and turnaround time** to improve system performance.

17. List two possible reasons for process termination.

Answer:

1. The process finishes its execution normally (calls `exit()`).
2. It is terminated by the OS or another process due to an error or violation (e.g., memory overflow).

18. Explain the purpose of the wait () and exit () system calls.

Answer:

- •**wait ():** Makes a parent process wait until its child process finishes.
- •**exit ():** Terminates a process after completion.

19. Differentiate between shared memory and message-passing models of inter-process communication.

Answer:

Shared Memory	Message Passing
Processes share a common memory space.	Processes communicate by sending messages.
It is faster but requires synchronization.	It is slower but safer (no shared data conflicts).
Processes share a common memory space to exchange data directly.	Processes communicate by sending and receiving messages through the OS

20. Differentiate between a thread and a process.

Answer:

Thread	Process
A lightweight unit within a process.	An independent program in execution.
Threads can share data easily.	Processes are isolated from one another.
Faster to create and switch.	Slower due to separate memory.

21. Define multithreading.

Answer:

I t is the process of running multiple threads of the same process to run concurrently, improving performance and resource utilization or simply can be defined as handling requests of multiple users simultaneously.

22. Explain the difference between a CPU-bound process and an I/O-bound process.

Answer:

CPU-bound	I/O-bound
Spends most of its time performing computations on CPU.	Spends most of its time waiting for I/O operations
Example: Mathematical calculations.	Example: File reading or printing.

23. What are the main responsibilities of the dispatcher?

Answer: The dispatcher is a small program that performs the actual process switching. It:

1. Saves and loads process states.
2. Updates CPU registers and memory mapping.
3. Transfers control to the next process selected by the scheduler.

Define starvation and aging in process scheduling.

Answer:

- **Starvation:** When a process waits too long for CPU time.
- **Aging:** Gradually increasing a waiting process's priority to prevent starvation.

24. What is a time quantum (or time slice)?

Answer:

A fixed small time given assigned to each process in Round Robin scheduling.

25. What happens when the time quantum is too large or too small?

Answer:

- **Too large:** Acts like FCFS (bad response).
- **Too small:** Too many context switches (overhead).

26. Define the turnaround ratio (TR/TS).

Answer:

Turnaround Ratio = Turnaround Time / Service Time

27. What is the purpose of a ready queue?

Answer:

The purpose of ready queue is to hold all the processes that are ready and waiting for the CPU time.

28. Differentiate between a CPU burst and an I/O burst.

Answer:

CPU Burst	I/O Burst
The process executes instructions on CPU.	The process performs input/output operations.
It uses CPU resources.	It uses I/O devices (e.g., disk, printer).

29. Which scheduling algorithm is starvation-free, and why?

Answer:

The scheduling algorithm which is starvation-free is Round-Robin, because every process gets a fair share of CPU time.

30. Outline the main steps involved in process creation in UNIX.

- **Answer:**
  • Parent calls `fork ()` to create a child.
- Child gets a copy of parent's memory.
- `Exec ()` loads new program.
- Parent uses `wait ()` for child to finish.

31. Define zombie and orphan processes.
Answer:

Type	Definition
Zombie	Process finished but still has entry in process table.
Orphan	Process whose parent has finished before it.

32. Differentiate between Priority Scheduling and Shortest Job First (SJF).
Answer:

Priority Scheduling	SJF Scheduling
It is based on priority value assigned to each process.	It is based on the shortest CPU burst time.
High-priority process runs first.	Process with least execution time runs first.

33. Define context switch time and explain why it is considered overhead.
Answer:

- **Context switch time** is the time taken to save and load process states during switching.
  It is **overhead** because no useful work is done during it.

34. List and briefly describe the three levels of schedulers in an Operating System.
Answer:

Scheduler	Function
Long-term	Selects processes to load into memory.
Short-term	Selects process to run on CPU.
Medium-term	Suspends or resumes processes.

35. Differentiate between User Mode and Kernel Mode in an Operating System.
Answer:

User Mode	Kernel Mode
Used for running user applications.	Used for executing OS kernel code.
Limited access to hardware	Full access to system resources.

User Mode	Kernel Mode
Errors affect only the process.	Errors may crash the entire system.

Section-C:

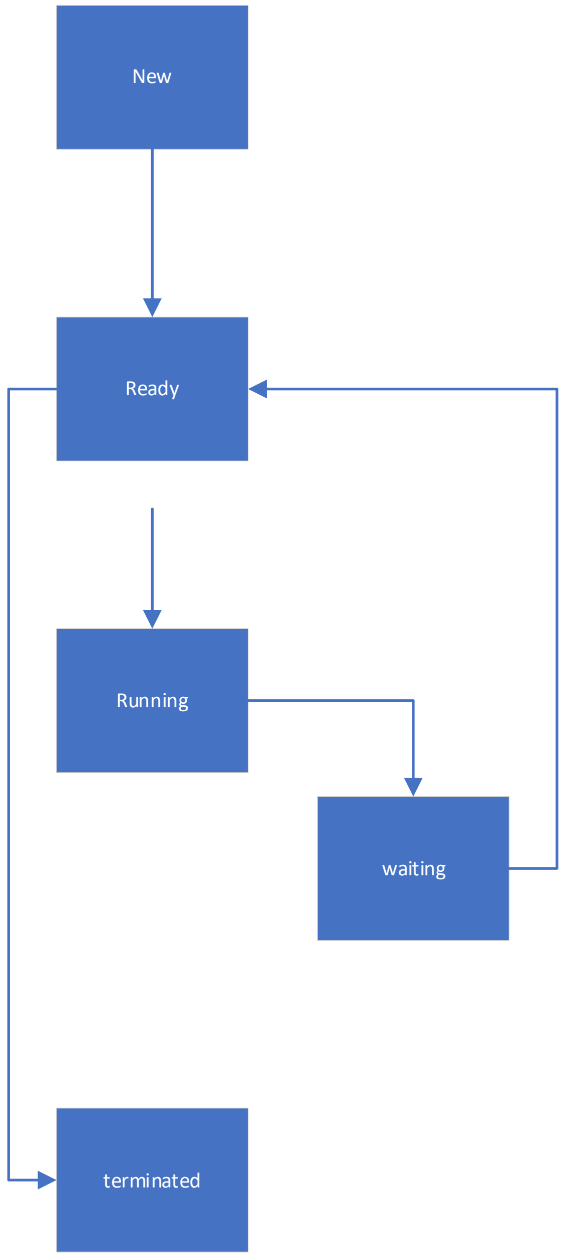
Technical / Analytical Questions (4 marks each)

1. Describe the complete life cycle of a process with a neat diagram showing transitions between New, Ready, Running, Waiting, and Terminated states.

Answer:
A process goes through several states during its execution. The OS manages transitions between these states using scheduling and I/O handling.

States and Transitions:

1. **New:** Process is being created.
2. **Ready:** Process is loaded into main memory in the queue and waiting for the CPU allocation.
3. **Running:** The process actively executes instructions here.
4. **Waiting (Blocked):** Process is waiting for an I/O event or signal to complete or the other event to occur.
5. **Terminated:** Process has completed its execution and is removed from the system.



2. Write a short note on context switch overhead and describe what information must be saved and restored.
   Answer:

Context switching is the process of saving the current state of a running process(like it's id, registers etc) and loading the state of another process to resume its execution.

Context Switch Overhead:

- The time the CPU spends switching between processes is called **context switch overhead**.
- During this time, **no useful computation** is done — hence, it reduces CPU efficiency.
- Frequent context switches (especially in preemptive systems) increase overhead.

Information Saved and Restored:

When switching, the OS must **save** the following for the outgoing process and **restore** it for the incoming process:

1. **CPU registers** (general-purpose and special registers).
2. **Program Counter (PC)** – address of next instruction.
3. **Stack Pointer** – location of current stack.
4. **Memory management info** – page tables, base/limit registers.
5. **Process State** – ready, running, or waiting.
6. **Accounting information** – CPU time used, priority, etc.

3. List and explain the components of a Process Control Block (PCB).
   Answer:
   The **Process Control Block (PCB)** is a data structure maintained by the Operating System to store **all information about a process**.
   It acts as the **"identity card"** of a process.

Component	Description
Process ID (PID)	It is a unique identifier for the process.
Process State	Current status (New, Ready, Running, Waiting, Terminated) of the process.
Program Counter (PC)	Address of the next instruction to be executed.
CPU Registers	Contents of all CPU registers when the process is paused.
Memory Management Info	Base and limit registers, page/segment tables.
CPU Scheduling Info	Priority, scheduling queue pointers, etc.
Accounting Info	CPU usage, process execution time, and limits.
I/O Status Info	List of I/O devices allocated or files opened by the process.

Purpose:

- The OS uses the PCB to **save the process state during a context switch** and **restore it** when the process is scheduled again.

4. Differentiate between Long-Term, Medium-Term, and Short-Term Schedulers with examples.
   Answer:

Scheduler Type	Function	Frequency of Execution	Example
Long-Term Scheduler (Job Scheduler)	Decides which new processes should be admitted to the ready queue (controls degree of multiprogramming).	Runs less frequently.	Loads jobs from disk into memory.
Medium-Term Scheduler	Temporarily removes (suspends) and later resumes processes to control memory use (swapping).	Runs occasionally.	Suspends background processes when RAM is low.
Short-Term Scheduler (CPU Scheduler)	Selects which ready process should run next on the CPU.	Runs very frequently (milliseconds).	Decides next process in Round Robin scheduling.

Summary:

- Long-term controls **which processes enter the system**.
- Medium-term controls **which processes stay in memory**.
- Short-term controls **which process runs next**.

5. Explain CPU Scheduling Criteria (Utilization, Throughput, Turnaround, Waiting, and Response) and their optimization goals
   Answer:

Criterion	Definition	Optimization Goal
CPU Utilization	Percentage of time the CPU is actively executing processes.	Maximize (keep the CPU busy as much as possible.
Throughput	Number of processes completed per unit time.	Maximize — complete more jobs in less time.
Turnaround Time	The total amount of time taken from submission to completion of a process.	Minimize (reduce the overall process completion time.)
Waiting Time	Total time a process spends waiting in the ready queue.	Minimize (improve overall efficiency.)
Response Time	Time from submission until the first response is produced (important for interactive systems).	Minimize (improve system responsiveness.)

Example:
An ideal scheduler keeps **CPU utilization high**, **throughput large**, and **waiting/turnaround/response times low**.

Section-D:

CPU Scheduling Calculations

• Perform the following calculations for each part (A–C).

- a) Draw Gantt charts for FCFS, RR (Q=4), SJF, and SRTF.
- b) Compute Waiting Time, Turnaround Time, TR/TS ratio, and CPU Idle Time.
- c) Compare average values and identify which algorithm performs best.

Part A:

Part-A

Process	Arrival Time	Service Time
P1	0	4
P2	2	5
P3	4	2
P4	6	3
P5	9	4

FCFS:

Gantt chart:

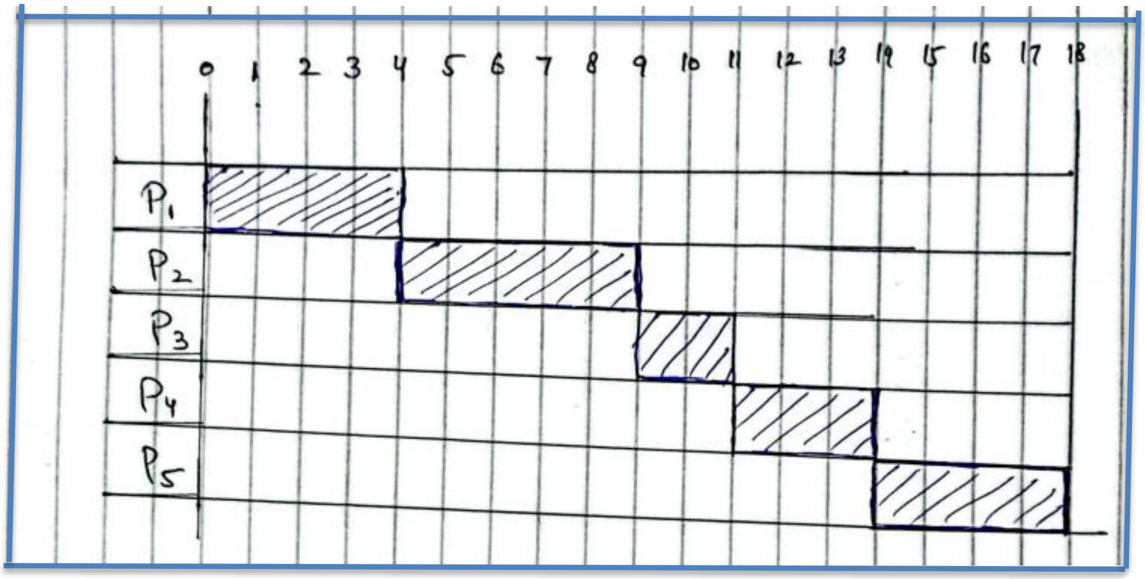


Table:

	P1	P2	P3	P4	P5
Turnaround time	4	7	7	8	9
Waiting time	0	2	5	5	5
TR/TS	1	1.4	3.5	2.66	2.25

Round robin:

Gantt chart:

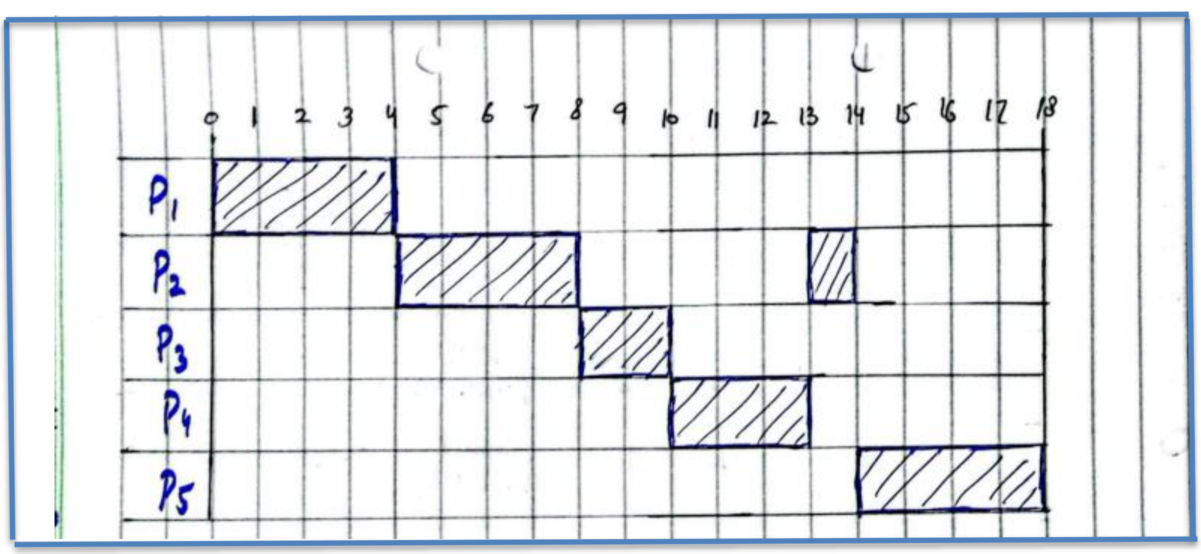


Table:

	P1	P1	P3	P4	P5
Turnaround time	**4**	**12**	**6**	**7**	**9**
Waiting time	**0**	**7**	**4**	**4**	**5**
TR/TS	**1**	**2.4**	**3**	**2.33**	**2.25**

CPU idle time is zero

SJF:

Gantt chart:

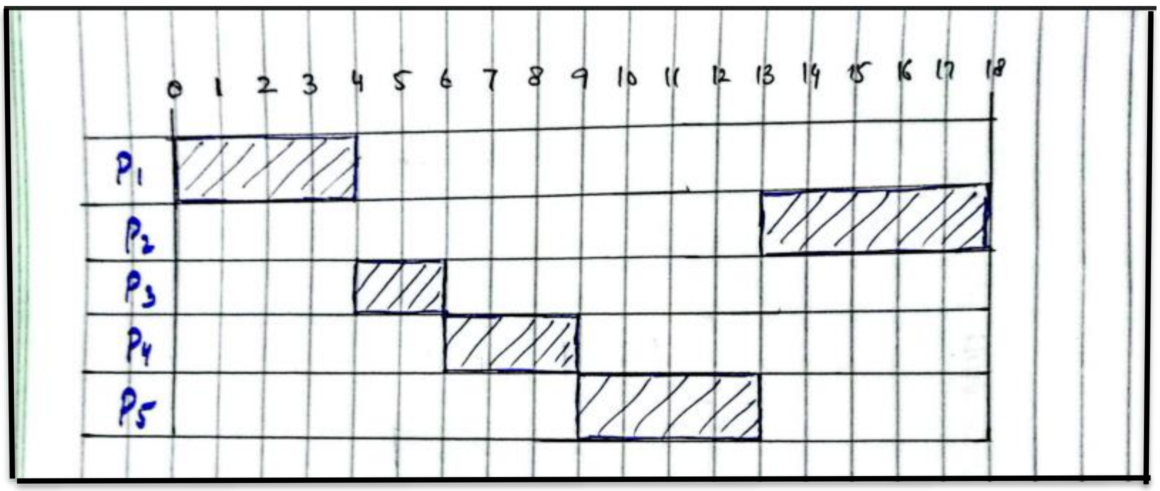


Table:

	P1	P2	P3	P4	P5
Turnaround time	**4**	**16**	**2**	**3**	**4**
Waiting time	**0**	**11**	**0**	**0**	**0**
TR/TS	**1**	**3.2**	**1**	**1**	**1**

CPU idle time=0

SRTF:

Gantt chart:

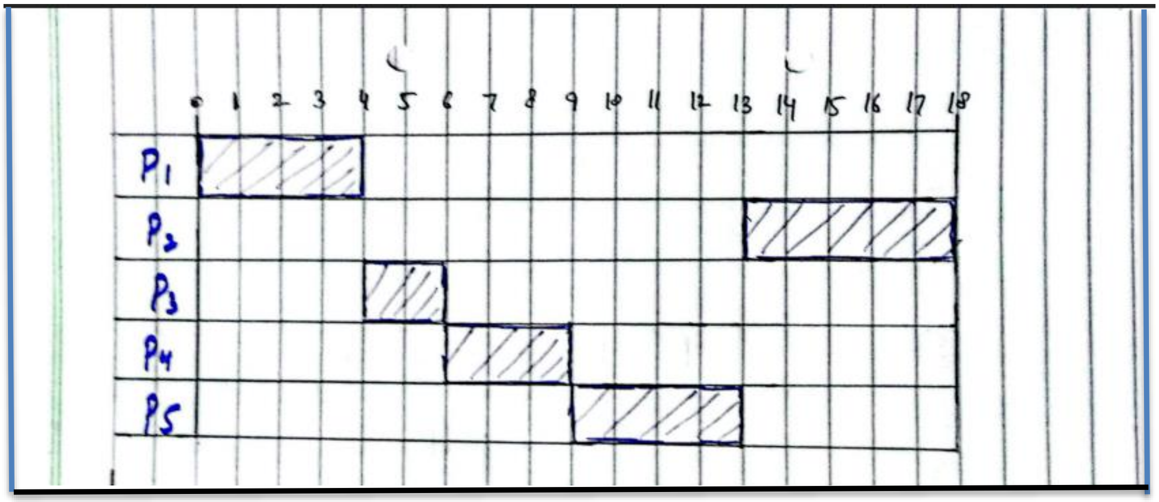


Table:

	P1	P2	P3	P4	P5
Turnaround time	**4**	**16**	**2**	**3**	**4**
Waiting time	**0**	**11**	**0**	**0**	**0**
TR/TS	**1**	**3.2**	**1**	**1**	**1**

CPU idle time =0

PART B:

Part-B

Process	Arrival Time	Service Time
P1	0	3
P2	1	5
P3	3	2
P4	9	6
P5	10	4

FCFS:

Gantt chart:

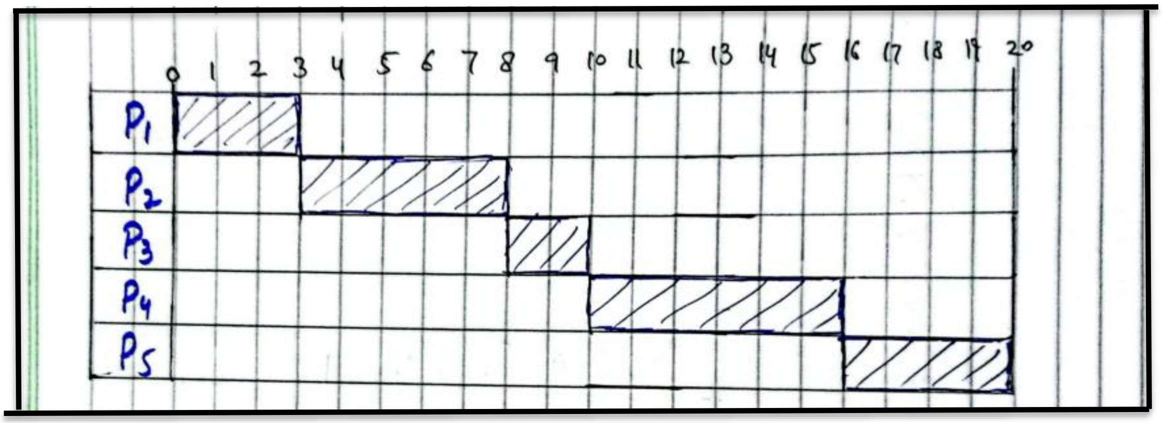


Table:

	P1	**P2**	**P3**	**P4**	**P5**
Turnaround time	**3**	**7**	**7**	**7**	**10**
Waiting time	**0**	**2**	**5**	**1**	**6**
TR/TS	**1**	**1.4**	**3.5**	**1.166**	**2.5**

Round robin:

Gantt chart:

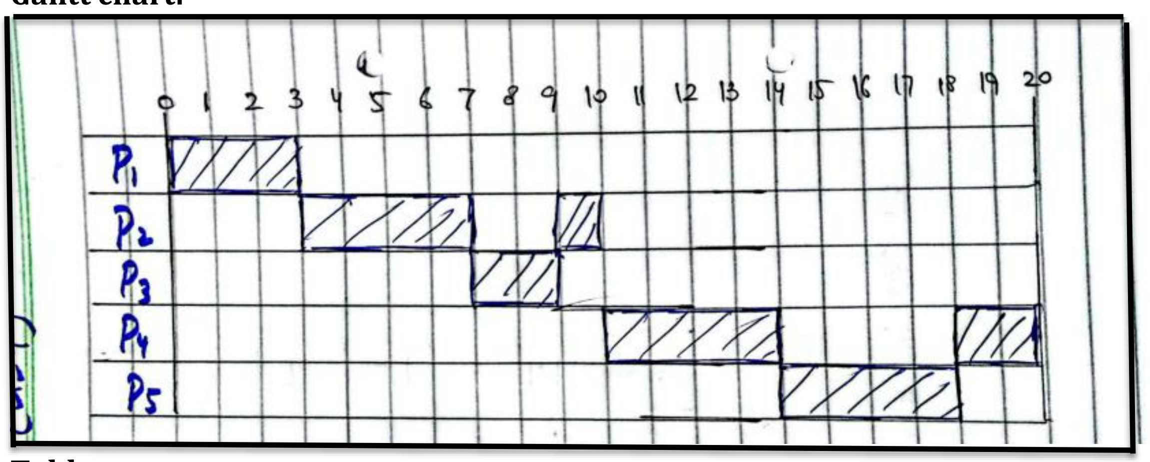


Table:

	P1	**P2**	**P3**	**P4**	**P5**
Turnaround time	**3**	**9**	**6**	**11**	**8**
Waiting time	**0**	**4**	**4**	**5**	**4**
TR/TS	**1**	**1.8**	**3**	**1.833**	**2**

CPU idle time=0

SJF:

Gantt chart:

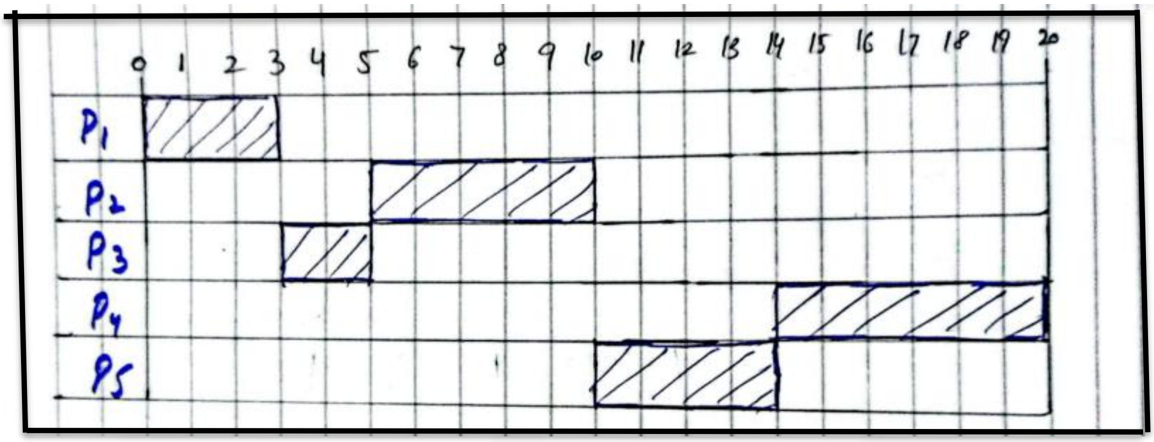


Table:

	P1	P2	P3	P4	P5
Turnaround time	**3**	**9**	**2**	**11**	**4**
Waiting time	**0**	**4**	**0**	**5**	**0**
TR/TS	**1**	**1.8**	**1**	**1.833**	**1**

CPU idle time=0

SRTF:

Gantt chart:

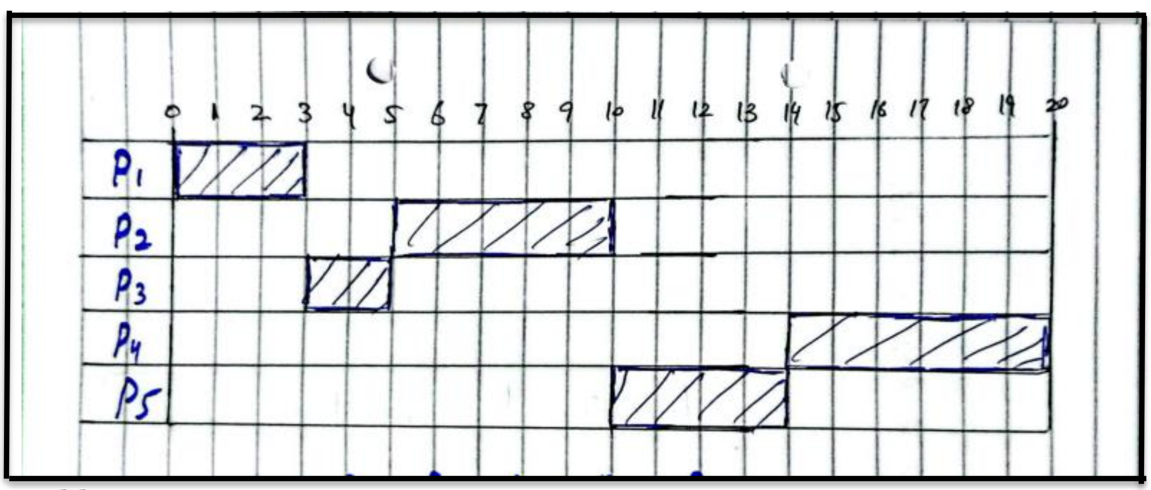


Table:

	P1	P2	P3	P4	P5
Turnaround time	**3**	**9**	**2**	**11**	**4**
Waiting time	**0**	**4**	**0**	**5**	**0**
TR/TS	**1**	**1.8**	**1**	**1.833**	**1**

CPU idle time=0

Part C:

Process	Arrival time	Service time
P1	0	3
P2	2	5
P3	5	2
P4	7	6
P5	11	1

FCFS:

Gantt chart:

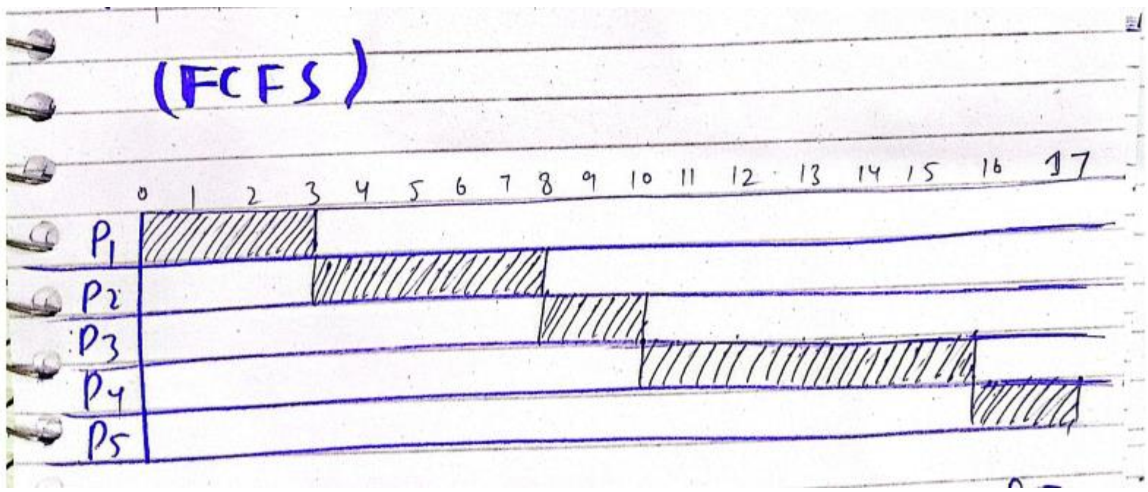


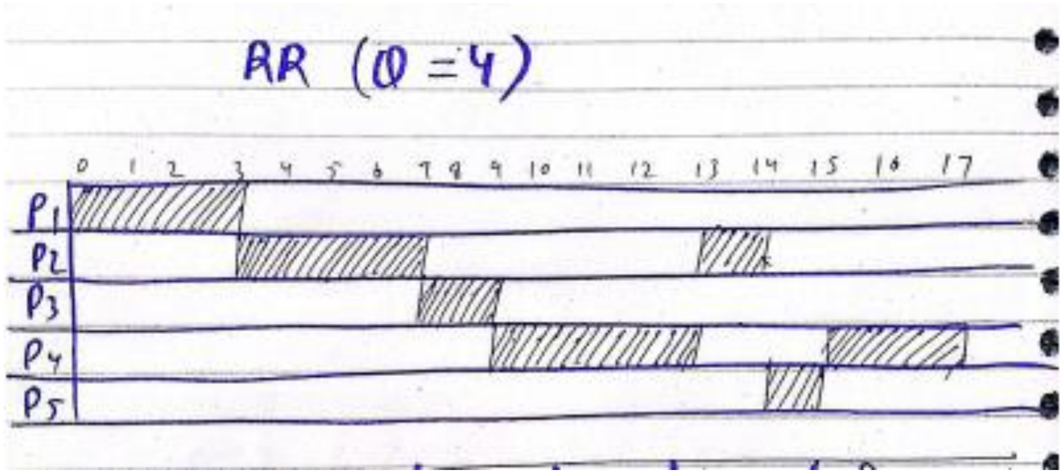
Table:

	P1	P2	P3	P4	P5
Turnaround time	3	6	5	9	6
Waiting time	0	1	3	3	5
TR/TS	1	1.2	2.5	1.5	6

CPU idle time=0

Round robin:

Gantt chart:

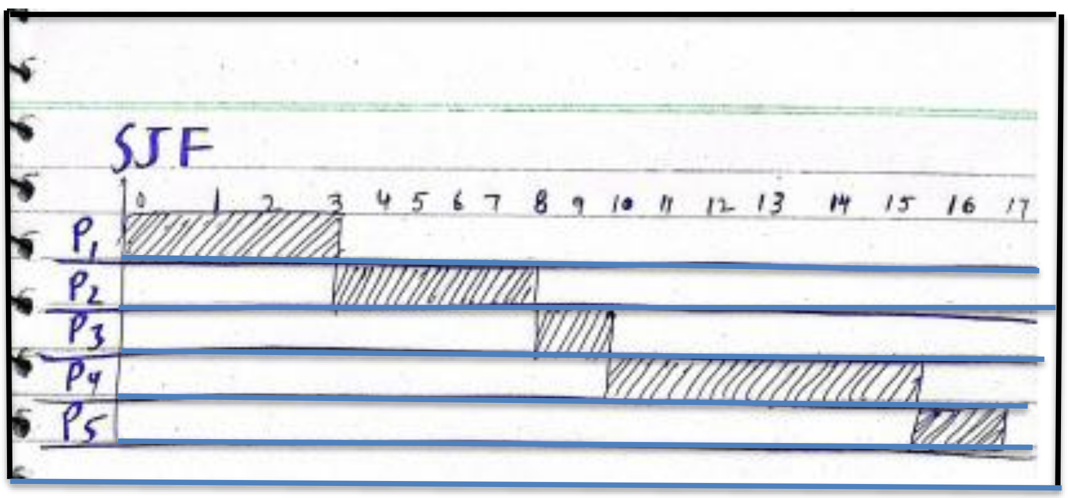


	P1	P2	P3	P4	P5
Turnaround time	**3**	**12**	**4**	**10**	**4**
Waiting time	**0**	**8**	**2**	**4**	**3**
TR/TS	**1**	**2.4**	**2**	**1.66**	**4**

CPU idle time=0

SJF:

Gantt chart:

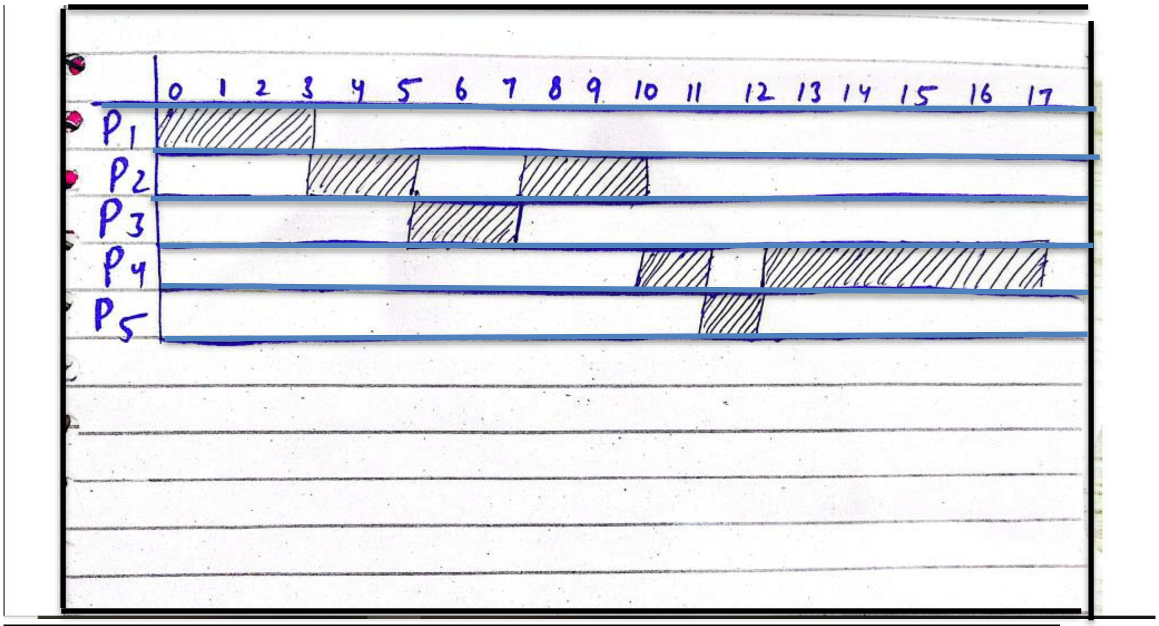


	P1	P2	P3	P4	P5
Turnaround time	**3**	**6**	**5**	**9**	**6**

Waiting time	**0**	**1**	**3**	**3**	**5**
TR/TS	**1**	**1.2**	**2.5**	**1.5**	**6**

SRTF:

Gantt chart:



	P1	**P2**	**P3**	**P4**	**P5**
Turnaround time	**3**	**6**	**5**	**9**	**6**
Waiting time	**0**	**1**	**3**	**3**	**5**
TR/TS	**1**	**1.2**	**2.5**	**1.5**	**6**

Competing the average values:

PART A:

ALGORITHM	**Average TR**	**Average waiting**	**Average TR/TS**	**Average idle time**
FCFS	**7**	**3.4**	**2.162**	**0**
RR	**7.6**	**4**	**2.196**	**0**
SJF	**5.8**	**2.2**	**1.44**	**0**
SRTF	**5.8**	**2.2**	**1.44**	**0**

Both SJF and STRF are best having minimum the TR, waiting time and TR/TS

PART B:

ALGORITHM	Average TR	Average waiting	Average TR/TS	Average idle time
FCFS	7.8	2.8	1.9132	0
RR	7.4	3.4	1.9266	0
SJF	5.8	1.8	1.326	0
SRTF	5.8	1.8	1.326	0

Both SJF and STRF are best having the minimum TR, waiting time and TR/TS

PART C:

ALGORITHM	Average TR	Average waiting	Average TR/TS	Average idle time
FCFS	5.8	2.4	2.41667	0
RR	6.6	3.4	1.9411	0
SJF	5.8	2.4	2.41667	0
SRTF	5.8	2.4	2.44	0