National Textile University

Department of Computer Science

Subject: Operating System

Submitted to: Nasir Mahmood

Submitted by: Eemaan Fatima

Reg number:23-NTU-CS-1146

Assignment No.1

Semester:5th

Section A:

Task 1:

Thread Information Display

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

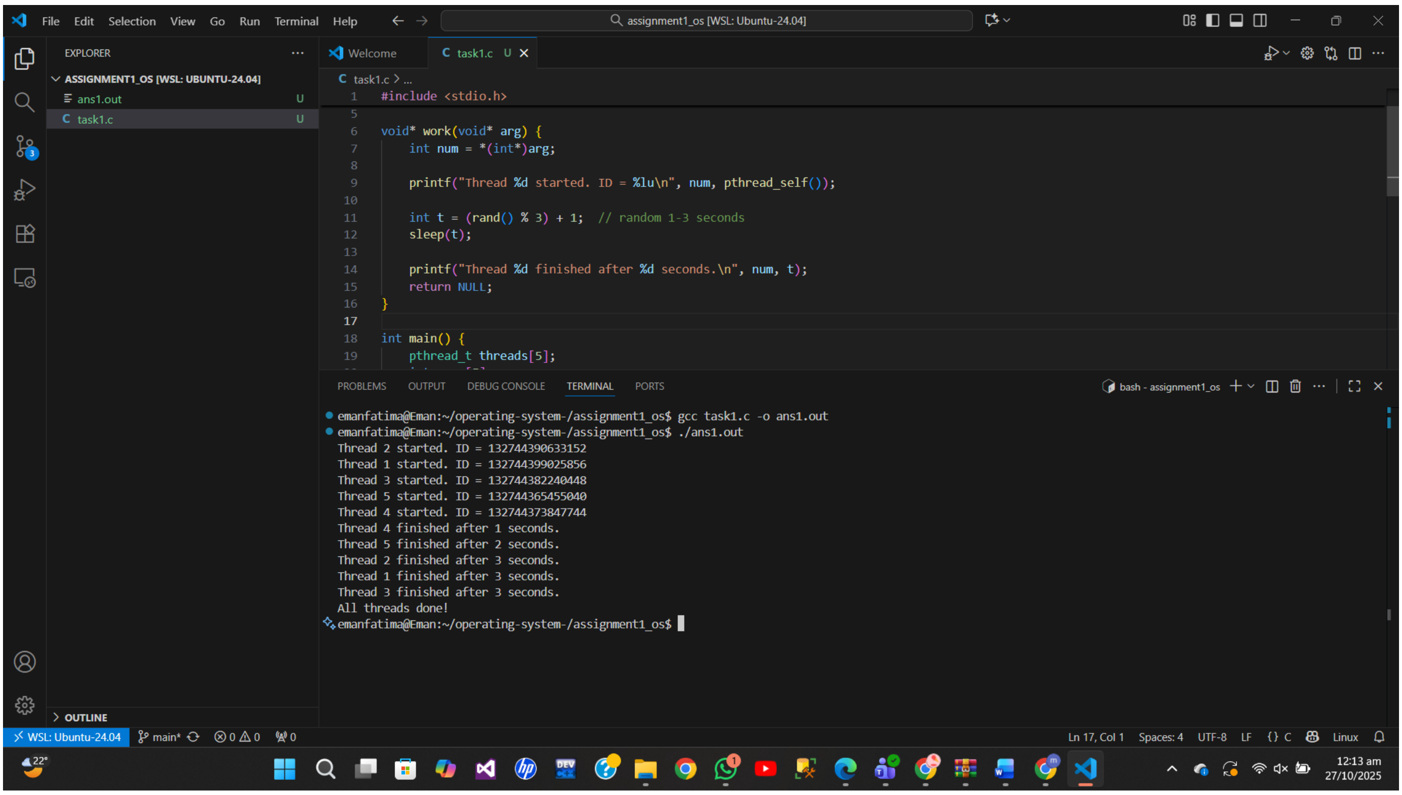
- **Print its thread ID using `pthread_self()`.**
- **Display its thread number (1st, 2nd, etc.).**
- **Sleep for a random time between 1–3 seconds.**
- **Print a completion message before exiting**.

Code :

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>


void *work(void *arg)

{

    int num = *(int *)arg; // int num = *(int*)arg; : argument ko integer mein convert karta hai.

    // arg thread ko diya gya argument


    printf("Thread %d started. ID = %lu\n", num, pthread_self()); // pthread_self() : current thread ka unique
ID print karta hai.


    int t = (rand() % 3) + 1; // random 1-3 seconds

    sleep(t);


    printf("Thread %d finished after %d seconds.\n", num, t);

    return NULL;

}


int main()

{
```

```
    pthread_t threads[5];
    int nums[5];
    srand(time(NULL)); // srand(time(NULL)) : random numbers ko alag alag banane ke liye.

    for (int i = 0; i < 5; i++)
    {
        nums[i] = i + 1;
        pthread_create(&threads[i], NULL, work, &nums[i]); // loop chly gi har thread ko ik unique no mily ga
    }

    for (int i = 0; i < 5; i++)
    {
        pthread_join(threads[i], NULL);
    }
    // pthread_join() : main thread wait karega jab tak har thread khatam nahi hota.
    printf("All threads done!\n");
    return 0;
}
```

o



Task 2:

Personalized Greeting Thread5

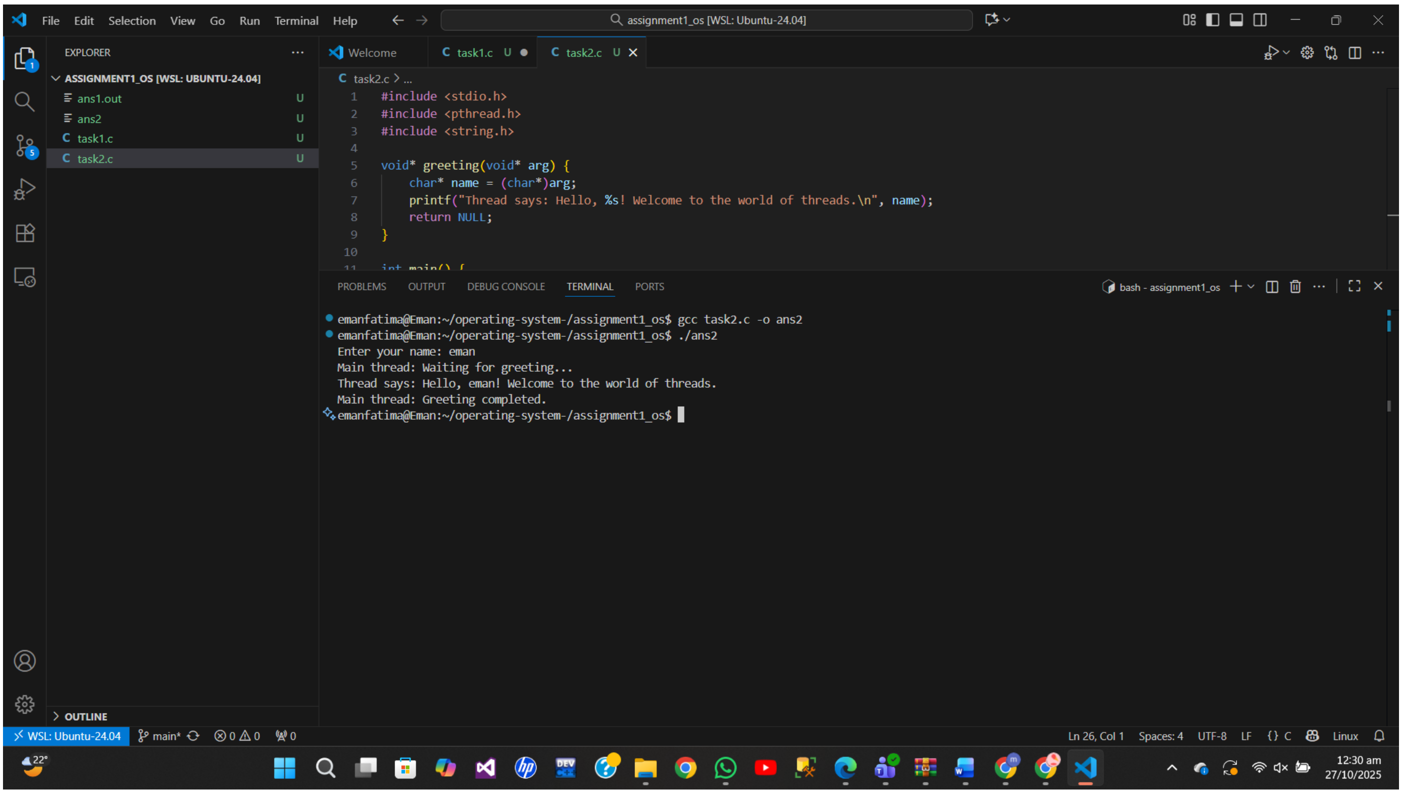
Write a C program that:

- **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 :

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

void *greeting(void *arg)
{
    char *name = (char *)arg; // argument ko string change krna
    printf("Thread says: Hello, %s! Welcome to the world of threads.\n", name);
    return NULL;
}

int main()
{
    pthread_t thread; // pthread_t thread; : ek thread variable declare karta hai jisme thread id store hogi
    char name[50];    // user ka naam store karne ke liye

    printf("Enter your name: ");
    scanf("%s", name); // user ka naam read karta hai aur name variable mein store karta hai

    pthread_create(&thread, NULL, greeting, (void *)name);
    printf("Main thread: Waiting for greeting...\n"); // thread create karta hai jo greeting function ko call
karega

    pthread_join(thread, NULL);
    printf("Main thread: Greeting completed.\n"); // wait karta hai jab tak greeting thread complete nahi hota
```

```
    return 0;
}
```



Task 3:

Number Info Thread

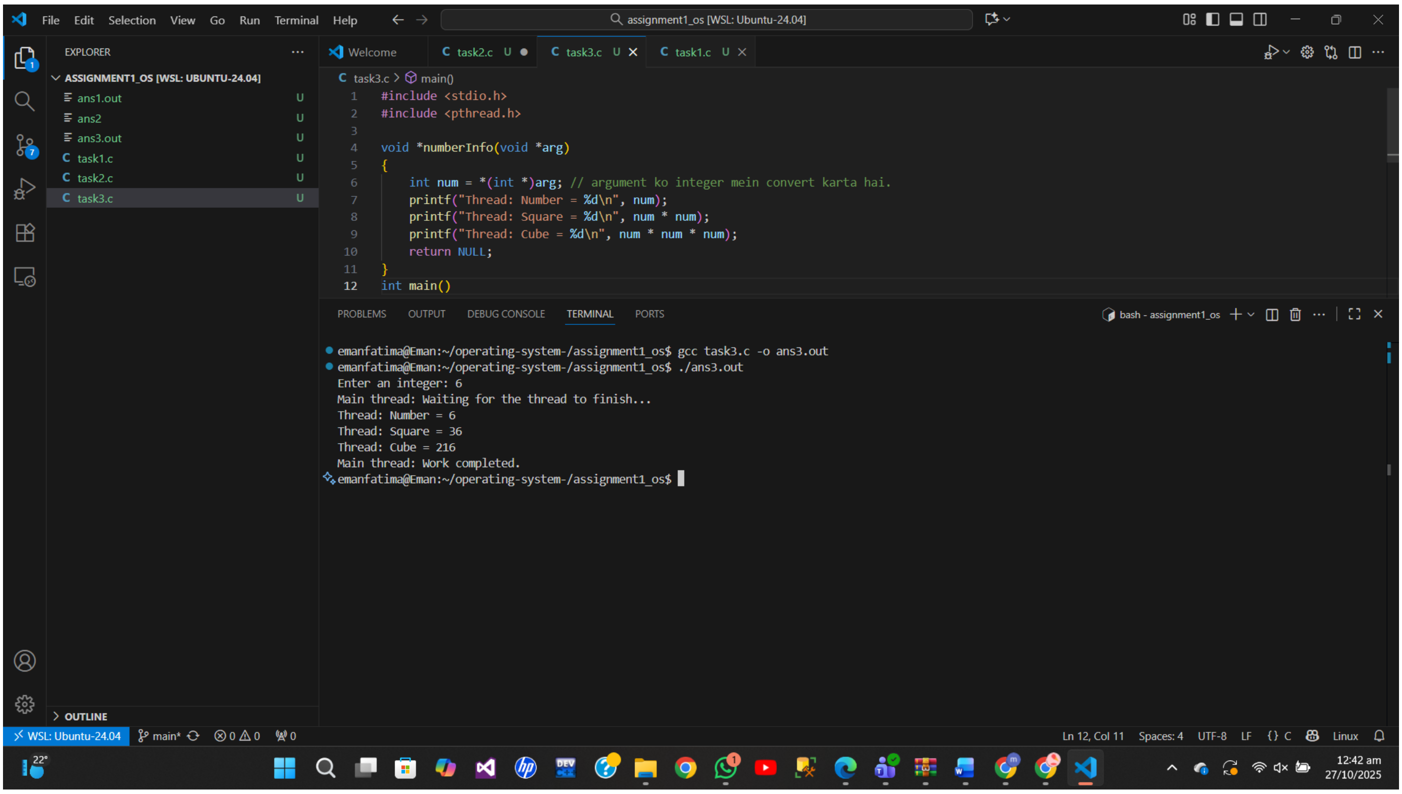
Write a program that:

- **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 :

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

void *numberInfo(void *arg)
{
```

```
    int num = *(int *)arg; // argument ko integer mein convert karta hai.
    printf("Thread: Number = %d\n", num);
    printf("Thread: Square = %d\n", num * num);
    printf("Thread: Cube = %d\n", num * num * num);
    return NULL;
}
int main()
{
    pthread_t thread;
    int number;
    printf("Enter an integer: ");
    scanf("%d", &number); // user se integer input leta ha
    pthread_create(&thread, NULL, numberInfo, (void *)&number);
    printf("Main thread: Waiting for the thread to finish...\n");
    pthread_join(thread, NULL);
    printf("Main thread: Work completed.\n"); // wait karta hai jab tak numberInfo thread complete nahi hota
    return 0;
}
```



Task 4:

Thread Return Values

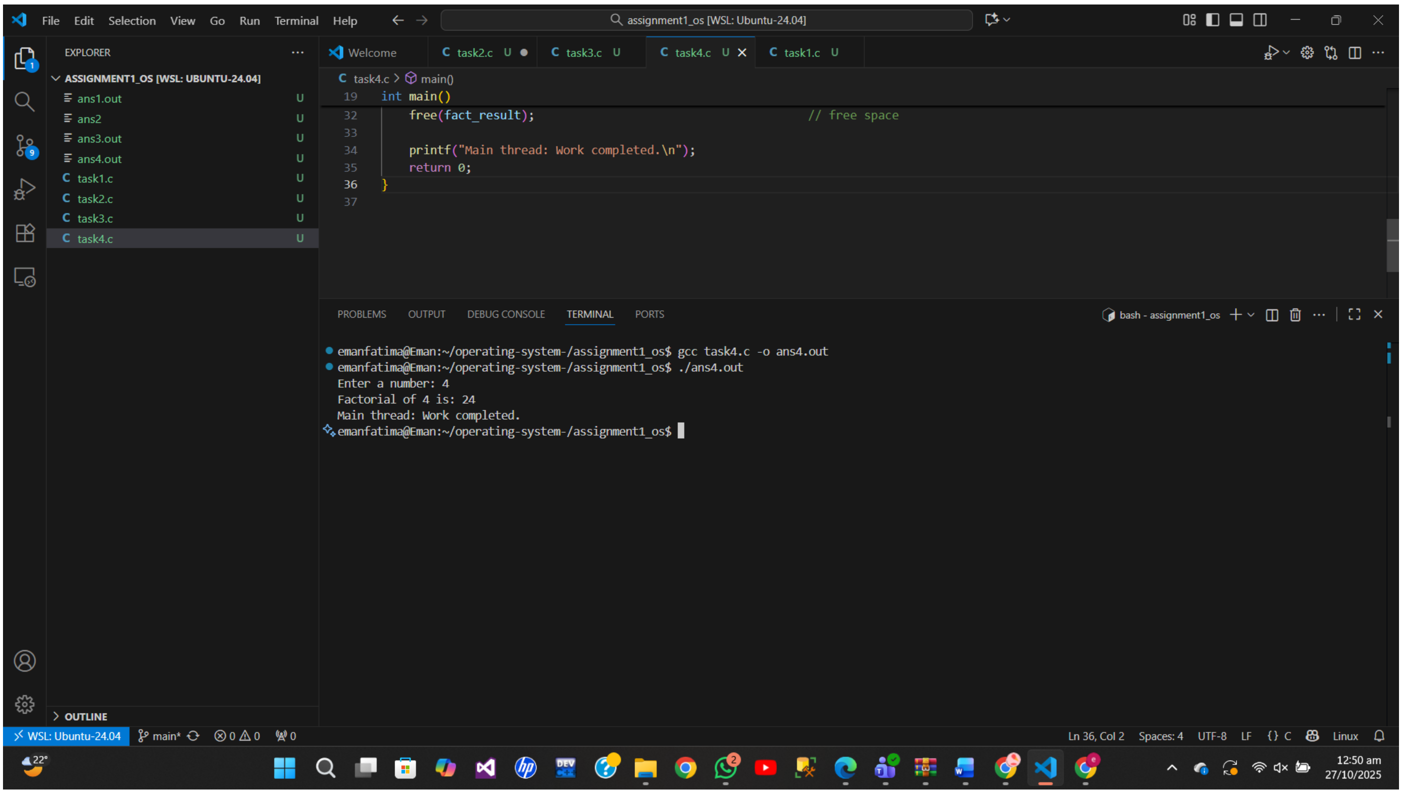
Write a program that creates a thread to compute the factorial of a number entered by the user.

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

Code :

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>


void *factorial(void *arg)

{

    int n = *((int *)arg);

    long long *result = malloc(sizeof(long long)); // result store karne ke liye dynamic memory allocate karta
hai.

    *result = 1;                                // factorial vlaue 1 start karta hai


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

    {

        *result *= i;

    } // loop ke through factorial calculate karta hai


    pthread_exit((void *)result);

}


int main()

{

    pthread_t thread;

    int num;

    long long *fact_result; // for result


    printf("Enter a number: ");

    scanf("%d", &num);
```

```
    pthread_create(&thread, NULL, factorial, &num);
    pthread_join(thread, (void **)&fact_result);

    printf("Factorial of %d is: %lld\n", num, *fact_result); //%lld: long long integer ke liye format specifier.
    free(fact_result);                          // free space

    printf("Main thread: Work completed.\n");
    return 0;
}
```



Task 5:

Struct-Based Thread Communication

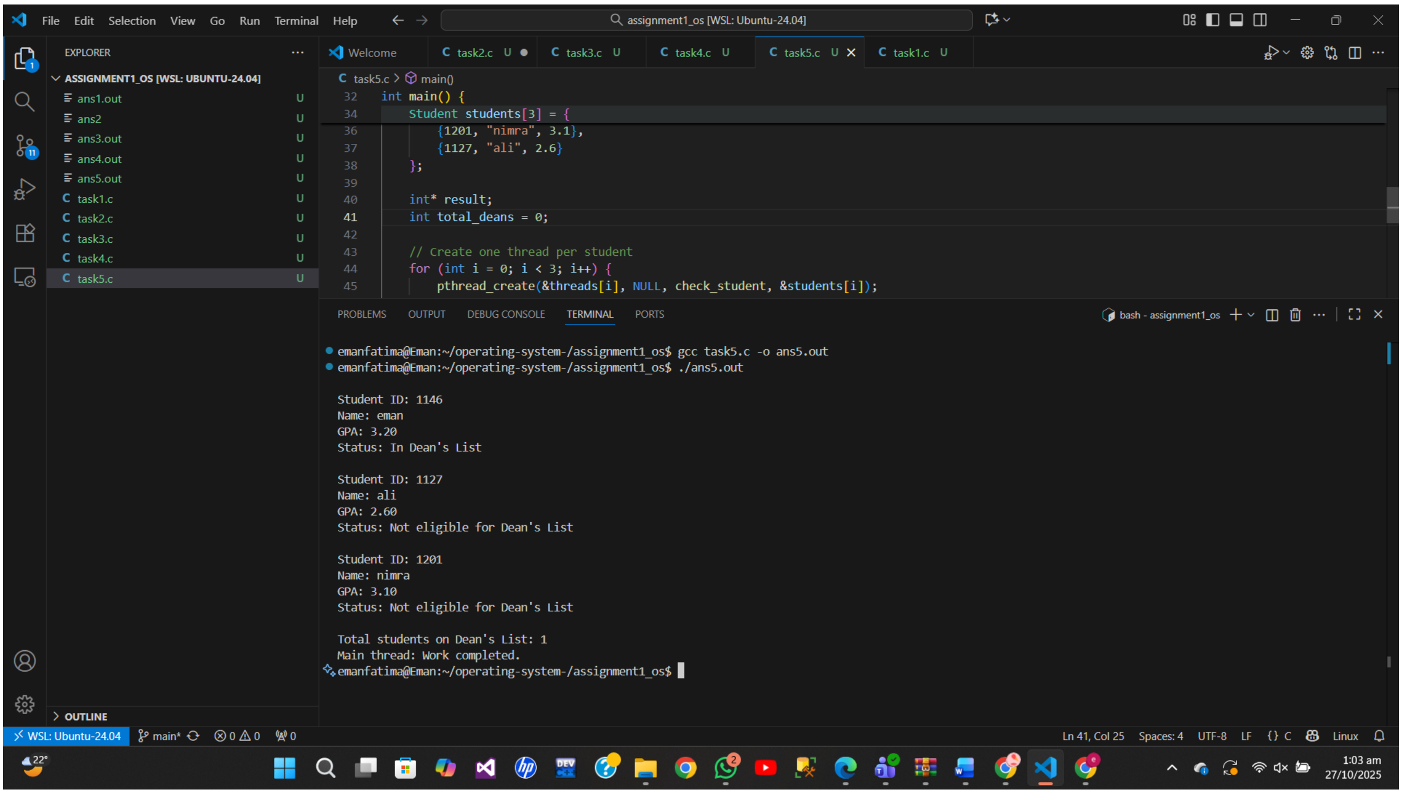
Create a program that simulates a simple student database system.

- **Define a struct: `typedef struct { int student_id; char name[50]; float gpa; } Student;`**
- **Create 3 threads, each receiving a different Student struct.**
- **Each thread prints student info and checks Dean's List eligibility (GPA ≥ 3.5).**
- **The main thread counts how many students made the Dean's List.**

Code :

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

// Structure to store student information
typedef struct
{
    int student_id;
    char name[50];
    float gpa;
} Student;

// Thread function to check Dean's list eligibility
void *check_student(void *arg)
{
    Student *s = (Student *)arg; // Convert void* to Student*
    printf("\nStudent ID: %d\n", s->student_id);
    printf("Name: %s\n", s->name);
    printf("GPA: %.2f\n", s->gpa);

    int *is_deans_list = malloc(sizeof(int)); // Allocate memory for result
    if (s->gpa >= 3.2)
    {
        printf("Status: In Dean's List \n");
        *is_deans_list = 1; // eligible
    }
    else
    {
        printf("Status: Not eligible for Dean's List \n");
        *is_deans_list = 0; // not eligible
    }
```

```c
    pthread_exit(is_deans_list); // Return result
}

int main()
{
    pthread_t threads[3]; // Thread array
    Student students[3] = {
        {1146, "eman", 3.2},
        {1201, "nimra", 3.1},
        {1127, "ali", 2.6}};

    int *result;
    int total_deans = 0;

    // Create one thread per student
    for (int i = 0; i < 3; i++)
    {
        pthread_create(&threads[i], NULL, check_student, &students[i]);
    }

    // Join all threads and collect results
    for (int i = 0; i < 3; i++)
    {
        pthread_join(threads[i], (void **)&result);
        total_deans += *result;
        free(result); // Free memory allocated in thread
    }

    printf("\nTotal students on Dean's List: %d\n", total_deans);
    printf("Main thread: Work completed.\n");
    return 0;
}
```



Section B:

Short Questions:

Q1 :Define an Operating System in a single line.

An operating system is the controller of a computer a piece of software that manages all the hardware and software resources, and provides a simple, consistent way for humans and applications to interact with the machine without needing to know its complex inner workings.

Q2 :What is the primary function of the CPU scheduler?

Primary Function: The primary function of the CPU scheduler is to decide which process from the ready queue should be assigned to the CPU next, to maximize CPU utilization, ensure fairness, and improve overall system performance.

Q3 : List any three states of a process.

1. New: The process is being created. (Like a customer just walking into the restaurant.)
2. Ready: The process is loaded into memory and is waiting for its turn to use the CPU. (Like your order is in the queue, waiting for the chef to start cooking it.)
3. Running: The process's instructions are being executed by the CPU. (The chef is actively cooking your order right now.)
4. Waiting: The process cannot proceed until some external event occurs, like waiting for user input or data from a disk. (The chef has sent a helper to get more tomatoes from the storeroom. Your order is paused until the helper returns.)
5. Terminated: The process has finished execution. (Your meal is served, and the order is complete.)

Q4 : What is meant by a Process Control Block (PCB)?

A Process Control Block is like a process's ID card or passport. It's a data structure where the operating system keeps all the crucial information it needs to manage a specific process, such as its process ID, current state, and where it is in memory.

Q5:Differentiate between a process and a program.

Program	Process
A program is just a set of instructions saved on your disk.	A process is that program when it starts running in your computer's memory.
Its inactive (passive) .it just sits in storage (like .exe, .py, or .c file).	Its active (running) .it's using CPU, memory, etc.
Stored on hard drive.	Exists in RAM (main memory).
Example: notepad.exe file in your C: drive.	Example: When you double-click Notepad, Windows loads that file into memory and starts a process (Notepad window you see).
One program can start many processes.	Each process runs its own copy of the program.

Q6 :What do you understand by context switching?
Context switching is the process in which the operating system temporarily stops one process and starts another by saving the current process's state (like CPU registers, program counter, and memory information) and loading the saved state of the next process. This allows multiple programs to share a single CPU efficiently. For example, when you are listening to music on Spotify and suddenly a WhatsApp notification appears, the OS saves Spotify's state, switches to WhatsApp to display the message, and then later restores Spotify's state to continue playing the song. This switching makes multitasking possible but also adds a little overhead because saving and loading states take some time.

Q7: Define CPU utilization and throughput.

- CPU Utilization: This measures how much of the CPU's time is spent doing useful work (not sitting idle). It's like tracking the percentage of a workers 8 hour shift that they are actually working.

Example: If the CPU was busy for 9 out of 10 seconds, the utilization is 90%. We want this to be as high as possible.

- Throughput: This is the number of processes completed per unit of time. It's a measure of the total work done.

Example: If a scheduling algorithm can complete 10 processes in one second, its throughput is 10 processes per second. A higher throughput is better.

Q 8: What is the turnaround time of a process?
Turnaround time is the total time taken from the moment a process is submitted to the system until the moment it completes. It's the process's total life span in the system.

Example: A process arrives at time 0. It finishes its execution at time 10. Its turnaround time is 10 - 0 = 10 units.

Q 9: How is waiting time calculated in process scheduling?
Waiting time is the total time a process spends waiting in the ready queue. It does not include the time it is actually running or doing I/O operations.

Example:

- Arrives at time 0.
- Waits in ready queue from time 0 to 4. (4 sec wait)
- Runs on CPU from time 4 to 7.
- Waits in ready queue again from time 7 to 9. (2 sec wait)
- Runs from time 9 to 10 and finishes.
- Total Waiting Time = 4 + 2 = 6.

Q10:Define response time in CPU scheduling.
Response time is the duration from when a request is submitted until the first response is produced. It's crucial for interactive systems where users expect quick feedback.

Example: You click a button in a program. The time between your click and the moment the program shows a loading indicator or the first piece of data is the response time. You don't have to wait for the entire operation to finish to get this initial feedback.

Q11 :What is preemptive scheduling?
Preemptive scheduling allows the operating system to interrupt or stop a currently running process so that a higher-priority or shorter process can use the CPU. This helps improve responsiveness and ensures that important processes are not delayed. Example: Round Robin, Preemptive Priority, and SRTF are preemptive algorithms.

Q12 :What is non-preemptive scheduling?
In non-preemptive scheduling, once a process starts execution, it cannot be stopped until it finishes or voluntarily gives up the CPU. The CPU remains with the same process until it is complete. Example: FCFS (First Come First Serve) and Non-preemptive SJF.

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

- Starvation Free: Every process, regardless of its size or priority, gets a fair share of the CPU time. A long process won't block shorter ones indefinitely because it will be interrupted after its time quantum.
- Good for Time Sharing Systems: It provides excellent response time for interactive users. Since each user gets frequent, small slices of CPU time, the system feels responsive to everyone simultaneously.

Q 14 :Mention one major drawback of the Shortest Job First (SJF) algorithm.
The major drawback of the Shortest Job First (SJF) algorithm is that it is difficult to use in real systems because the exact time a process will take to finish (its CPU burst time) is not known in advance. In other words, the operating system cannot accurately guess how long each process will need the CPU before it actually runs. It’s like trying to arrange people in a line according to how quickly they will finish their tasks, without knowing their task duration beforehand. Therefore, SJF is mostly used as a theoretical concept rather than in real practical scheduling..

Q 15 : Define CPU idle time.
CPU Idle Time means the time period when the CPU is not doing any work because there are no processes ready to run. This often happens when all processes are waiting for something else like input and output operations to finish.

Example:
Imagine your computer is copying files from a USB drive. While it’s waiting for the USB to send data, the CPU has nothing to process at that moment so it stays idle

In short, CPU idle time = CPU waiting time with no active process to execute.
High idle time means the CPU is underutilized, while low idle time means the CPU is busy and efficient.

Q 16 : State two common goals of CPU scheduling algorithms.

- Maximize CPU Efficiency: The primary goal is to keep the CPU as busy as possible. An idle CPU is a wasted resource. This is measured by CPU Utilization.
- Ensure Fairness: Every process should get a reasonable chance to run on the CPU. No single process should be stuck waiting forevera problem known as starvation. Algorithms like Round Robin are designed specifically for this.

Q17. Explain the purpose of the wait() and exit() system calls.

- wait() system call allows a parent process to pause its execution until one of its child processes finishes. It helps the parent to collect the child's exit status and ensure proper synchronization.
  Example: A parent process waits for a child process (like a compiler) to finish before starting another task.

- exit() system call is used by a process to end its execution and return a status code to the operating system or parent process.
  Example: When a program finishes execution successfully, it calls exit(0).

Q18. Difference between Shared Memory and Message Passing (Inter-Process Communication):

Shared Memory	Message Passing
Processes share a common memory area.	Processes communicate by sending and receiving messages.
Faster because data is directly accessed.	Slower due to message copying between processes.
Needs synchronization tools like semaphores.	No manual synchronization required.
Suitable for large data transfers.	Suitable for small data exchanges.
Example: Producer-Consumer model.	Example: Client-Server communication.

Q19 :Difference between a Thread and a Process:

Thread	Process
A lightweight part of a process.	An independent program in execution.
Shares memory and resources with other threads.	Has its own memory and resources.
Faster to create and switch.	Slower to create and switch.
Failure of one thread can affect others.	Failure of one process doesn't affect others.
Example: Multiple tabs in a browser.	Example: Running Chrome and Word simultaneously.

Q20 : Define Multithreading.

Multithreading is the ability of a CPU or program to run multiple threads at the same time within a single process. It helps in improving performance, responsiveness, and efficient use of CPU resources.
Example: A web browser downloading files and rendering a webpage simultaneously.

Q21:. Difference between CPU-Bound and I/O-Bound Processes:

CPU-Bound Process	I/O-Bound Process
Spends most time using the CPU.	Spends most time waiting for I/O operations.
Performs heavy computations.	Performs frequent input/output tasks.
Example: Video rendering, data encryption.	Example: Reading a file, printing a document.
Needs a fast processor.	Needs fast I/O devices.

Q22: Main Responsibilities of the Dispatcher:

- Transfers control of the CPU to the process selected by the scheduler.
- Performs context switching between processes.
- Switches the CPU to user mode for execution.
- Jumps to the appropriate program location to continue the process.
  Example: The dispatcher switches from one running process to another in a multitasking OS like Linux.

Q23: Define Starvation and Aging in Process Scheduling.

- Starvation: A process waits indefinitely because other processes keep getting CPU time.
  Example: In Priority Scheduling, a low-priority process may never execute.
- Aging: A technique to prevent starvation by gradually increasing a waiting process's priority.
  Example: A process waiting for a long time in the queue gets higher priority over time.

Q 24. Define Time Quantum (Time Slice).

A time quantum is a fixed period for which a process can use the CPU before being interrupted or preempted.
Example: In Round Robin scheduling, each process gets 100 milliseconds to execute.

Q 25 :What happens when the Time Quantum is too large or too small?

- Too Large: The system behaves like First Come First Serve (FCFS), leading to poor response time.
- Too Small: Too many context switches occur, increasing system overhead.
  Example: A very small time slice causes frequent switching between tasks, reducing efficiency.

Q26. Define Turnaround Ratio (TR/TS).

Turnaround Ratio is the ratio of Turnaround Time (TR) to Service Time (TS). It shows how efficiently a process is completed compared to its actual service time.
Formula: TR/TS = Turnaround Time / Service Time
Example: If a process takes 10 seconds to complete but needs only 5 seconds of CPU time, TR/TS = 2.

Q27 : Purpose of the Ready Queue.

The Ready Queue holds all processes that are ready to run and waiting for CPU time. The scheduler selects processes from this queue to execute next.
Example: In Windows OS, multiple running applications are placed in the ready queue awaiting CPU allocation.

Q 28: Difference between a CPU Burst and an I/O Burst:

CPU Burst	I/O Burst
Process executes instructions on the CPU.	Process waits for input/output operations to complete.
Involves computations and logic.	Involves waiting for devices like disk, printer, or keyboard.
Example: Performing arithmetic operations.	Example: Reading data from a hard drive.

Q29 : Which scheduling algorithm is starvation-free and why?

Round Robin (RR) scheduling is starvation-free because every process gets CPU time in a fixed circular order. This ensures that no process waits indefinitely.
Example: In a time-shared system, each user program gets a fair CPU share.

Q 30 : Main Steps in Process Creation in UNIX:

1. The parent process creates a child using the fork() system call.
2. The OS allocates memory and resources to the child.
3. The child process loads a new program using exec().
4. The new process starts execution.
5. The parent process uses wait() to wait for the child's completion.

Example: When you run a command in Linux terminal, a new child process is created using fork().

Q 31 : Define Zombie and Orphan Processes.

- Zombie Process: Occurs when a process finishes execution, but its parent hasn't collected its exit status using wait(). The process remains in the process table as a "zombie."
  Example: A finished child process still listed in ps output.
- Orphan Process: Occurs when a parent process ends before its child. The orphan process is then adopted by the init process.
  Example: A background process continues running after the parent terminal is closed.

Q 32: Difference between Priority Scheduling and Shortest Job First (SJF):

Priority Scheduling	Shortest Job First (SJF)
Executes processes based on priority.	Executes the process with the shortest CPU burst first.
May cause starvation of low-priority processes.	May cause starvation of long processes.

Priority Scheduling	Shortest Job First (SJF)
Priorities can be static or dynamic.	Based on estimated CPU burst time.
Example: Real-time systems where some tasks are urgent.	Example: Batch systems where short jobs are preferred.

Q 33. Define Context Switch Time and Explain Why It's Overhead.

Context Switch Time is the time taken by the CPU to save the current process's state and load the next process's state. It is considered overhead because it uses CPU time but doesn't perform any actual computation.
Example: Switching between user programs in Windows reduces efficiency slightly due to context saving.

Q34 :Three Levels of Schedulers in an Operating System:

1. Long-Term Scheduler: Decides which new processes should enter the system.
   Example: Adding jobs to the ready queue.
2. Medium-Term Scheduler: Temporarily removes processes from memory (suspension) and later brings them back.
   Example: Swapping processes between RAM and disk.
3. Short-Term Scheduler: Selects which ready process gets the CPU next.
   Example: Deciding the next process to execute in multitasking.

Q35 :Difference between User Mode and Kernel Mode:

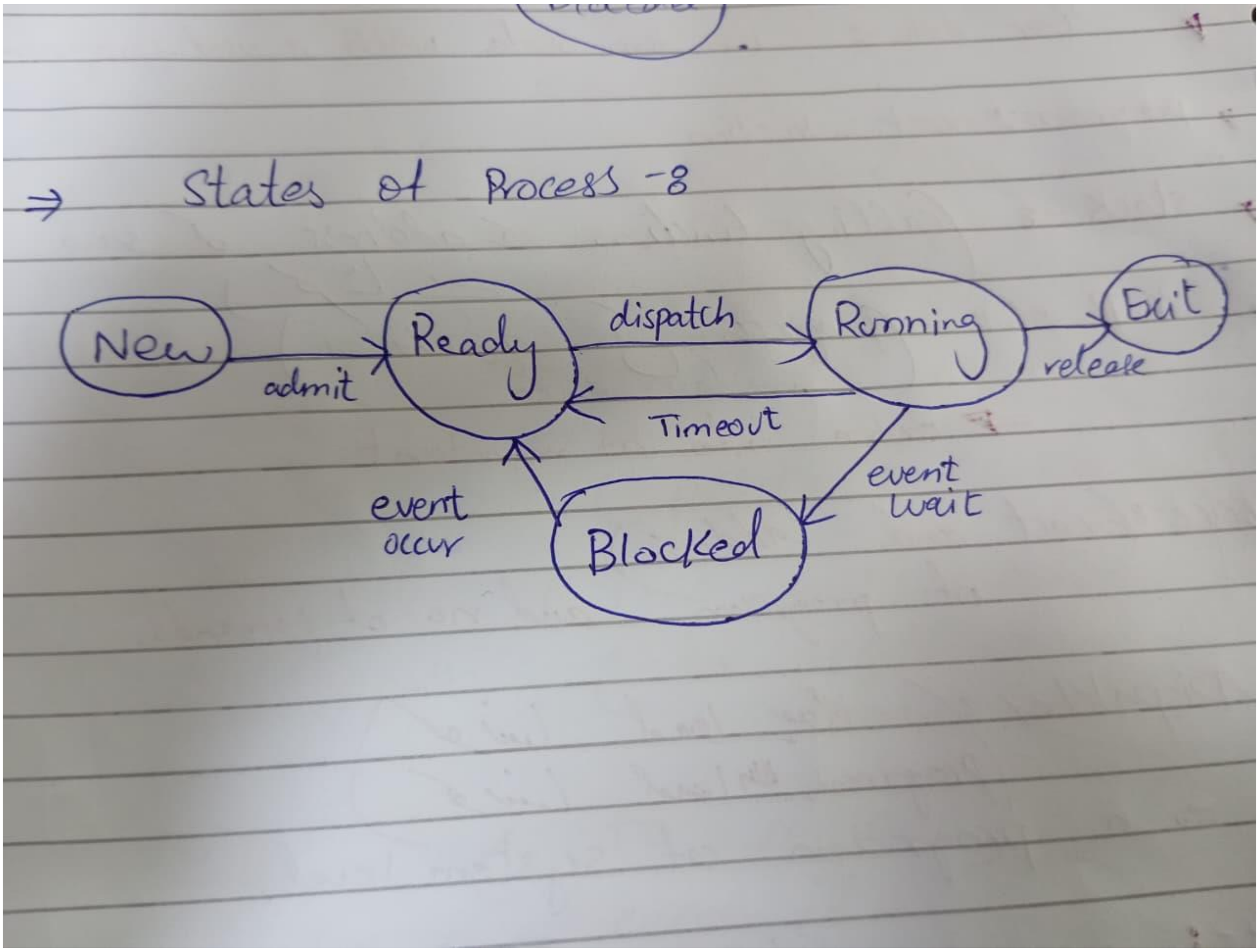
User Mode	Kernel Mode
Executes user programs with limited privileges.	Executes operating system code with full privileges.
Cannot directly access hardware or system memory.	Can access all hardware and memory directly.
Errors affect only the current program.	Errors can crash the whole system.
Example: Running a web browser or text editor.	Example: Running device drivers or handling system calls.

Section C :

Technical / Analytical Questions :

Q 1:

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



New State
The process is newly created and has not yet been moved to the main memory. It is waiting for the operating system to admit it for execution.

Ready State
The process is loaded into memory and is ready to execute, but it is waiting for the CPU to become available.

Running State
The process is currently being executed by the CPU. It is actively performing its instructions and using system resources.

Blocked State
The process cannot continue execution because it is waiting for some event to occur, such as an input/output operation to complete.

Exit State
The process has finished its execution or has been terminated by the operating system, and all the resources used by it are released.

Q 2 :

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

A context switch happens when the CPU stops one process and starts another.
The time used by the operating system to save the current process details and load the next process details is called context switch overhead.
It is called overhead because it takes CPU time but does not do any real user work.

Information that must be saved and restored:

1. Program Counter: The address of the next instruction to be executed.
2. CPU Registers: Data and temporary values used by the process.
3. Process State: Shows if the process is running, waiting, or ready.
4. Memory Information: Includes base and limit register values.
5. I/O and Accounting Information: Open files, CPU usage time, and process priority.

Example:
When you switch from a web browser to a text editor, the OS saves the browser's data and loads the text editor's data so both can continue from where they stopped.

Q3:

List and explain the components of a Process Control Block (PCB)

A Process Control Block (PCB) is a structure in the operating system that keeps all important information about a process.
It helps the system manage and control multiple processes at the same time.

Main components of PCB:

1. Process State: Shows the current condition of the process such as running, ready, or waiting.
2. Program Counter: Tells the address of the next instruction the process will run.
3. CPU Registers: Stores temporary data, addresses, and results for the process.
4. CPU Scheduling Information: Includes priority and scheduling details for process selection.
5. Memory Management Information: Tells how much memory the process is using and its location in memory.
6. Accounting Information: Records CPU time used, process ID, and total execution time.
7. I/O Status Information: Contains details about open files and input/output devices used by the process.

Example:
When a process is paused and later continued, the PCB helps the OS remember where it left off and resume correctly.

Q4:

Difference between Long-Term, Medium-Term, and Short-Term Schedulers

Feature	Long-Term Scheduler (Job Scheduler)	Medium-Term Scheduler (Swapper)	Short-Term Scheduler (CPU Scheduler)
Main Task	Decides which new processes will enter memory for execution.	Temporarily removes or resumes processes from main memory.	Selects which process will run next on the CPU.
Speed	Works slowly.	Works at a medium speed.	Works very fast.
When It Works	When a new job or program arrives.	When the system memory is full or overloaded.	After each CPU burst or I/O completion.
Purpose	Controls how many programs are loaded in memory.	Balances memory use and system load.	Keeps the CPU busy and ensures fairness among processes.
Example	Loads 3 jobs out of 10 waiting into memory.	Swaps a process to disk to free up space.	Chooses which ready process runs next using Round Robin.

In short:

- Long-term scheduler controls which jobs enter memory.
- Medium-term scheduler manages which jobs stay or leave memory.
- Short-term scheduler decides which job runs on CPU next.

Q5:

Explain CPU Scheduling Criteria and Their Optimization Goals

Criteria	Meaning	Goal (Optimization)	Example
CPU Utilization	Shows how much the CPU is kept busy.	Maximize CPU use and reduce idle time.	CPU should always be working on some process.
Throughput	Number of processes completed in a fixed time.	Increase the number of completed processes.	More tasks finished per minute means better throughput.
Turnaround Time	Time taken from the start to the completion of a process.	Reduce total completion time.	A program finishes sooner after starting.
Waiting Time	Total time a process spends waiting for CPU.	Decrease waiting time for better performance.	A document should not wait long before printing starts.
Response Time	Time between submitting a request and getting the first response.	Make response faster for better user experience.	When clicking an app, it should open immediately.

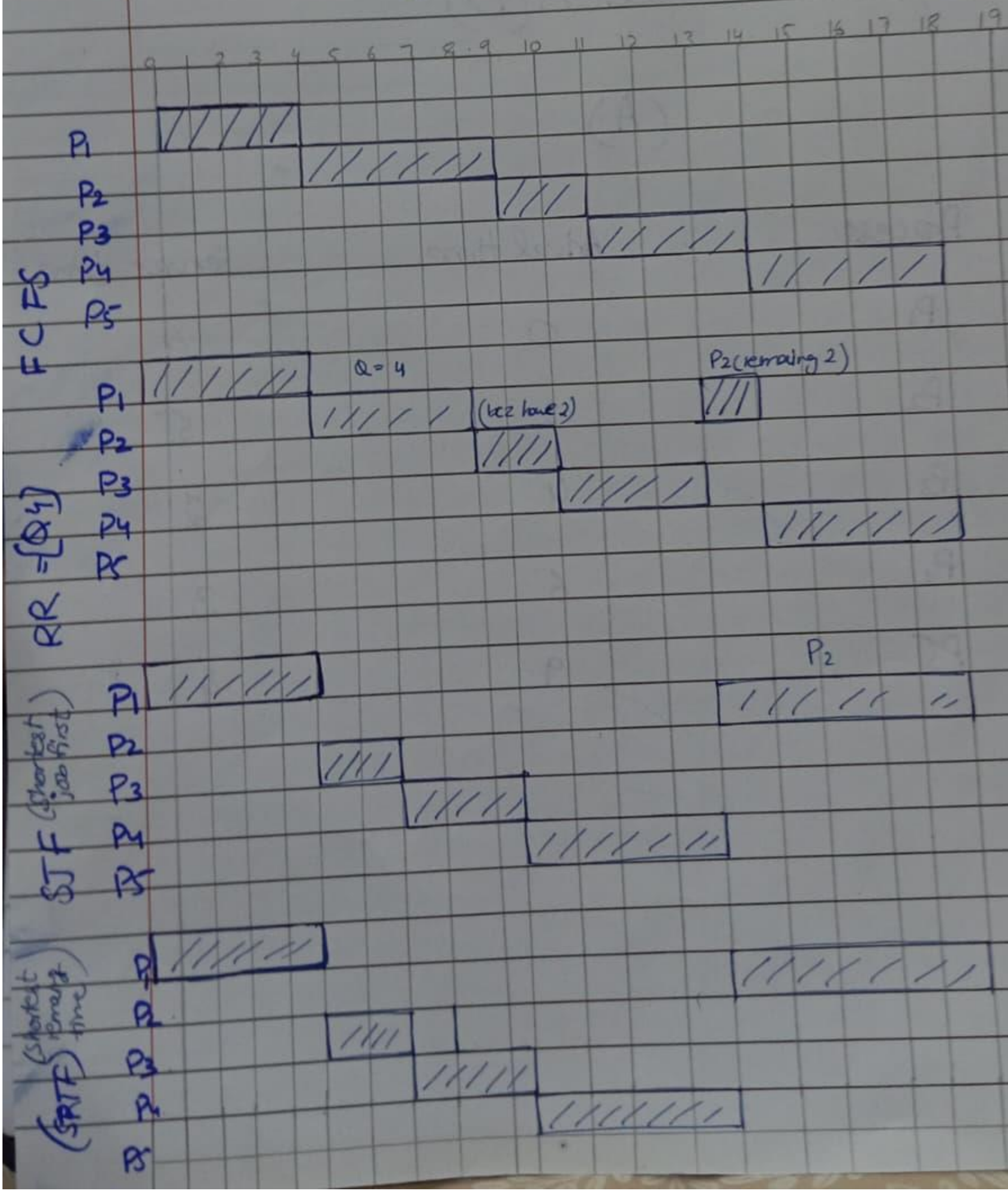
Section D

Section (D)

(A)

Process	Arival time	Service time
P_1	0	4
P_2	2	5
P_3	4	2
P_4	6	3
P5	9	4

(Ghant Chart)


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
FCFS
P1
P2
P3
P4
P5
RR =[Q4]
Q = 4
P2(remaining 2)
(bcz have 2)
SJF (Shortest job first)
SRTF (shortest remaining time)

FCFS

Process	Arrival	Service	Finish	Waiting time	Turnaround (Finish - Arrival)	Tr/Ts (Turnaround time / Service time)
P1	0	4	4	0	4	1.0
P2	2	5	9	2	7	1.4
P3	4	2	11	5	7	3.5
P4	6	3	14	5	8	2.67
P5	9	4	18	5	9	2.25
Average				3.40	7.0	2.16
CPU idle (time)	–	–	–	–	–	–

RR Q=4

Process	Arrival	Service	Finish	Waiting time	Turnaround	Tr/Ts
P1	0	4	4	0	4	1.0
P2	2	5	14	7	12	2.4
P3	4	2	10	4	6	3.0
P4	6	3	13	4	7	2.3
P5	9	4	18	5	9	2.2
Average	–	–	–	4.0	7.6	2.19
idle CPU time	–	–	–	–	–	–

SJF

Process	Arrival	Service	Finish	Waiting time	Turnaround	Tr/Ts
P1	0	4	4	0	4	1.0
P2	2	5	18	11	16	3.2
P3	4	2	6	0	2	1.0
P4	6	3	9	0	3	1.0
P5	9	4	13	0	4	1.0
Average				2.20	5.8	1.44
CPU idle time	–	–	–	–	–	–

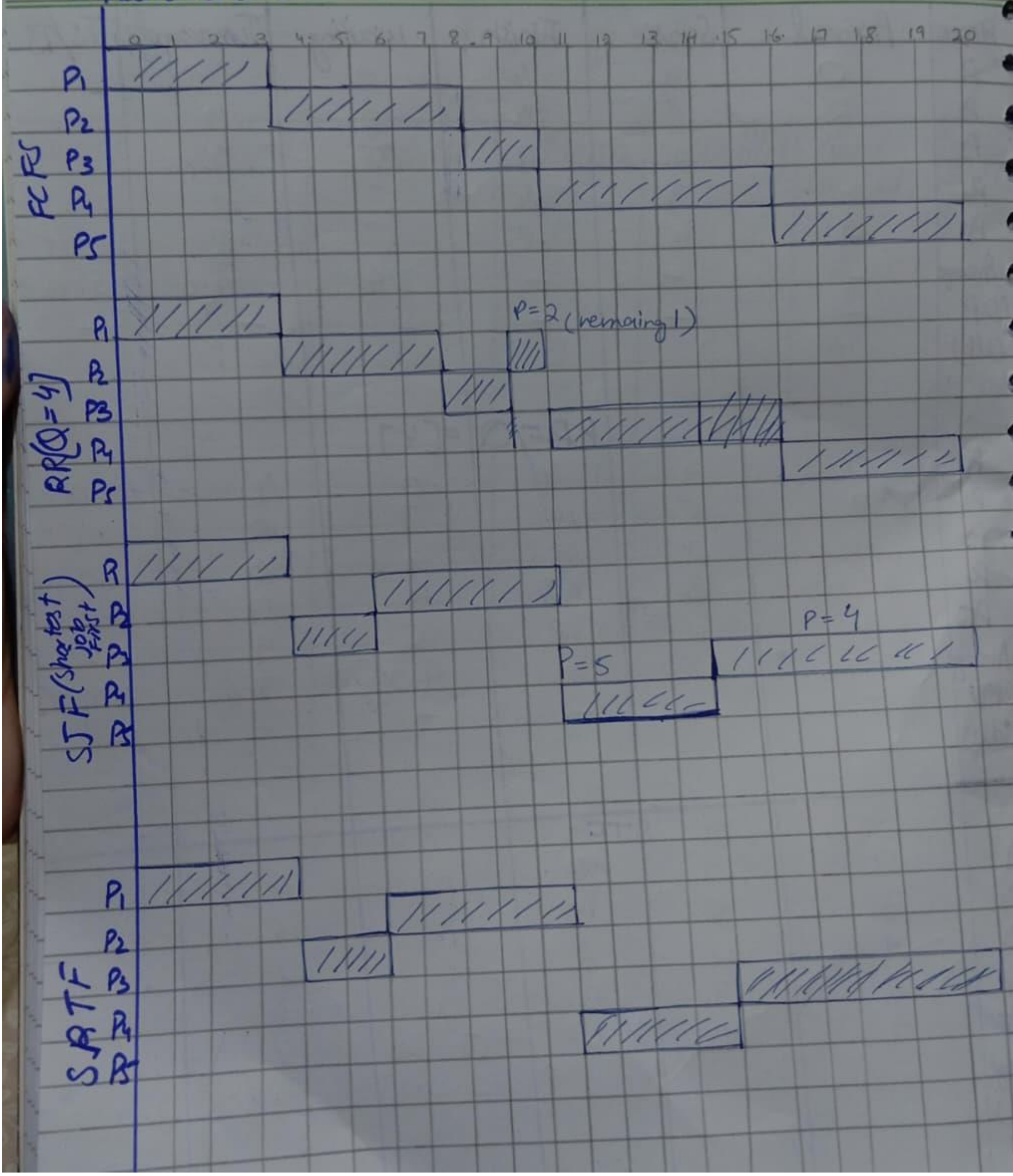
SRTF

Process	Arrival	Service	Finish	Waiting	(Turn Around)	Tr/Ts
P1	0	4	4	0	4	1.0
P2	2	5	18	11	16	3.2
P3	4	2	6	0	2	1.0
P4	6	3	9	0	3	1.0
P5	9	4	13	0	4	1.0
Average	—	—	—	2.20	5.80	1.40
CPU idle	—	—	—	—	—	—

Part B

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

Ghant Chart::


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
FCFS
P1
P2
P3
P4
P5
RR(Q=4)
P=2 (remaing 1)
SJF (shortest job first)
P=5
P=4
SRTF

FCFS

Process	Arrival	Service	Finish	Waiting	Turnaround	TS/TS
P1	0	4	4	0	4	1.6
P2	2	5	9	2	7	1.4
P3	4	2	11	5	7	3.5
P4	6	3	14	5	8	2.6
P5	9	4	18	5	9	2.25
Average	—	—	—	3.4	7.0	2.163
CUP (idle)	—	—	—	—	—	—

RR = Q = [4]

Process	Arrival	Service	Finish	Waiting	Turnaround	TS/TS
P1	0	4	4	0	4	1.0
P2	2	5	14	7	12	2.4
P3	4	2	10	4	6	3.0
P4	6	3	13	4	7	2.3
P5	9	4	18	5	9	2.25
Average	—	|	—	4.0	7.6	2.197
CPU (idle)	—	—	—	—	—	—

SJF

Process	Arrival	Service	Finish	Waiting	Turnaround	TS/TS
P1	0	4	4	0	4	1.0
P2	2	5	18	11	16	3.2
P3	4	2	6	0	2	1.0
P4	6	3	9	0	3	1.0
P5	9	4	13	2. [illegible]	4	1.0
Average				2.20	5.80	1.4
(idle) CPU						

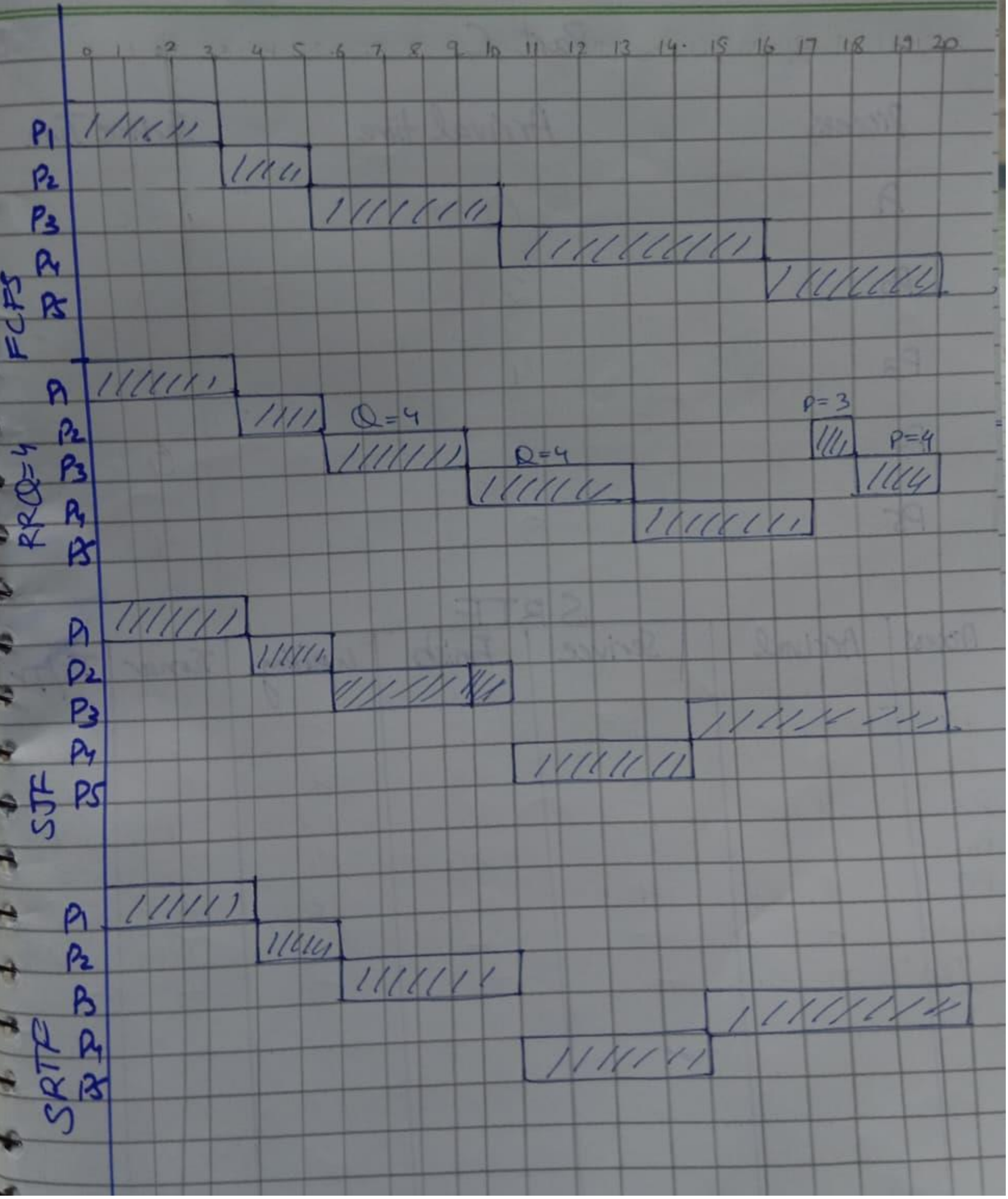
SRTF

Process	Arrival	Service	Finish	Waiting	(Turn 4around)	Tr/Ts
P_1	0	4	4	0	4	1.0
P_2	2	5	18	11	16	3.2
P3	4	2	6	0	2	1.0
P4	6	3	9	0	3	1.0
P5	9	4	13	0	4	1.0
Average	—	—	—	2.20	5.80	1.40
CPU(idle)	—	—	—	—	—	—

Part C

Process	Arrival time	Service Time
P1	0	3
P2	2	2
P3	4	5
P4	6	6
P5	8	4.

Ghant Chart :


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
FCFS
P1
P2
P3
P4
P5
RR Q=4
Q=4
Q=4
P=3
P=4
SJF
SRTF

FCFS

Process	Arrival	Service	Finish	Waiting	Turn around	Tr/Ts
P1	0	3	3	0	3	1.0
P2	2	2	5	1	3	1.5
P3	4	5	10	1	6	1.2
P4	6	6	16	4	10	1.67
P5	8	4	20	8	12	3.0
Average	—	—	—	2.80	6.8	1.67
CPU(idle)	—	—	—	—	—	—

SJF

P1	0	3	3	0	3	1.0
P2	2	2	5	1	3	1.5
P3	4	5	10	1	6	1.2
P5	6	6	14	2	6	1.5
P4	8	4	20	8	14	2.3
Average	—	—	—	2.40	6.40	1.507
CPU(idle)	—	—	—	—	—	—

Quantum = 4

P1	0	3	3	0	3	1.0
P2	2	2	5	1	3	1.5
P3	4	5	18	9	14	2.8
P4	6	6	20	8	14	2.3
P5	8	4	17	5	9	2.25
		—		4.6	8.6	1.97
Average	—	—	—	—	—	—
CPU(idle)	—					

SRTF

Process	Arrival	Serivce	Finish	waiting	Turnar	Ts/Tr
P1	0	3	5	5	2	1.67
P2	2	2	4	2	0	1.0
P3	4	5	16	12	7	2.4
P4	6	6	20	14	8	2.3
P5	8	4	18	10	6	2.5
Average	—	—	—	8.6	4.6	1.98
CPU(idle)	—	—	—	—	—	—