National Textile University

Department of Computer Science

Subject: Operating System

Submitted to: Nasir Mehmood

Submitted by: Nimra Tanveer

Reg number:23-NTU-CS-1201

Semester:5th

Assignment # 01

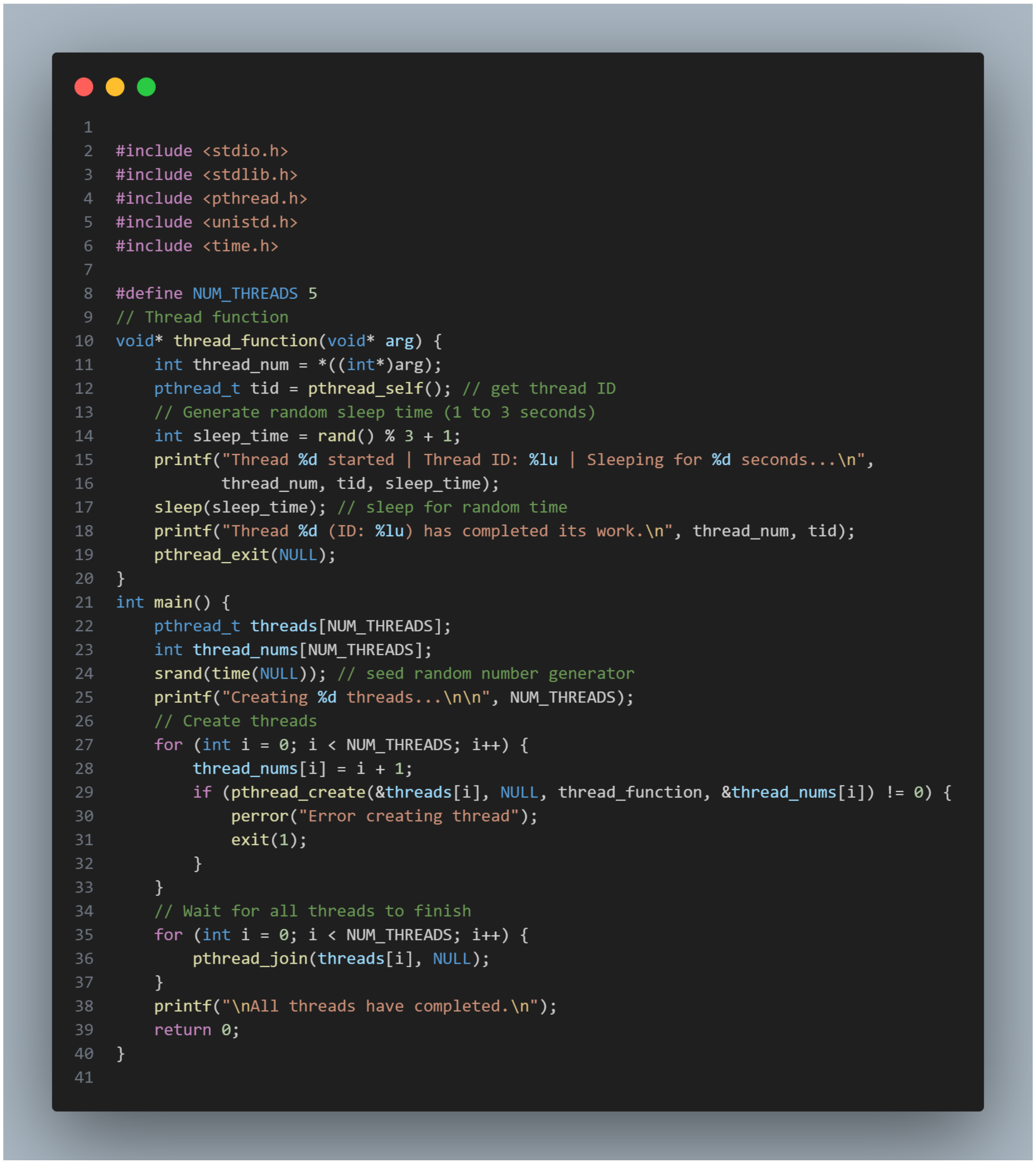
Section-A: Programming Tasks

Task 1 – Thread Information Display

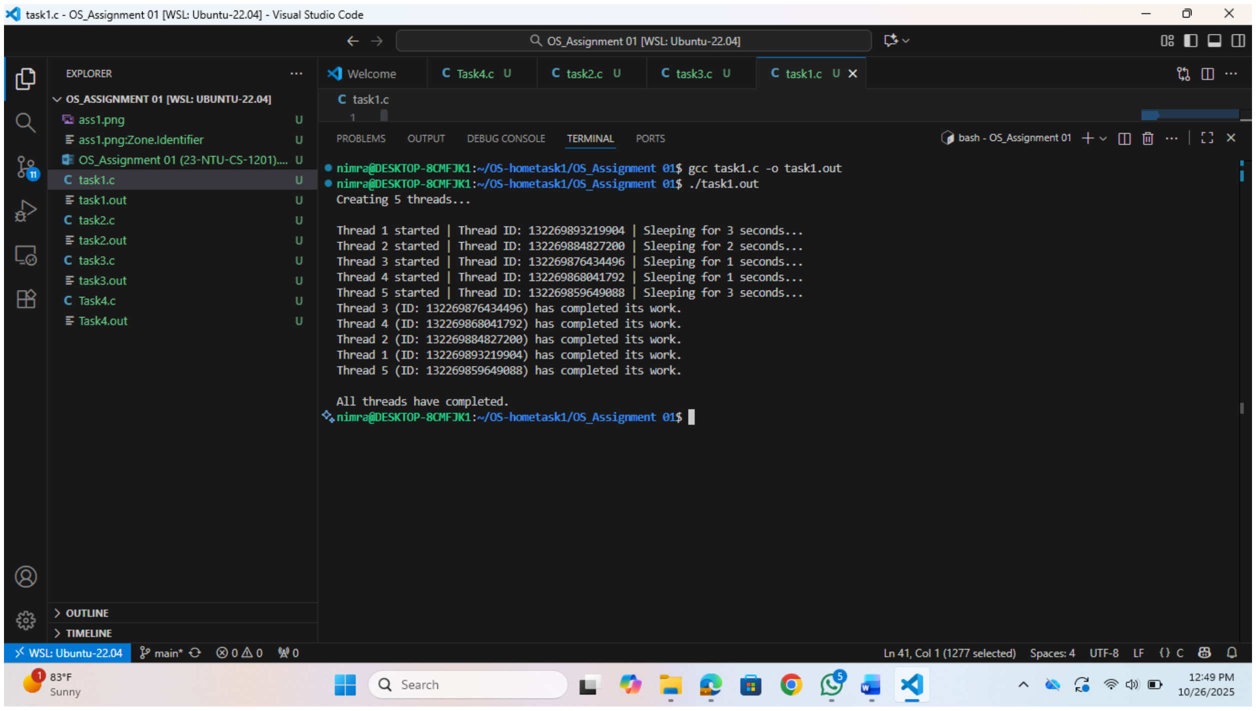
Write a program that creates 5 threads. Each thread should:

- 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.

Expected Output: Threads complete in different orders due to random sleep times.


```c

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <time.h>

#define NUM_THREADS 5
// Thread function
void* thread_function(void* arg) {
    int thread_num = *((int*)arg);
    pthread_t tid = pthread_self(); // get thread ID
    // Generate random sleep time (1 to 3 seconds)
    int sleep_time = rand() % 3 + 1;
    printf("Thread %d started | Thread ID: %lu | Sleeping for %d seconds...\n",
           thread_num, tid, sleep_time);
    sleep(sleep_time); // sleep for random time
    printf("Thread %d (ID: %lu) has completed its work.\n", thread_num, tid);
    pthread_exit(NULL);
}
int main() {
    pthread_t threads[NUM_THREADS];
    int thread_nums[NUM_THREADS];
    srand(time(NULL)); // seed random number generator
    printf("Creating %d threads...\n\n", NUM_THREADS);
    // Create threads
    for (int i = 0; i < NUM_THREADS; i++) {
        thread_nums[i] = i + 1;
        if (pthread_create(&threads[i], NULL, thread_function, &thread_nums[i]) != 0) {
            perror("Error creating thread");
            exit(1);
        }
    }
    // Wait for all threads to finish
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }
    printf("\nAll threads have completed.\n");
    return 0;
}

```

OUTPUT:

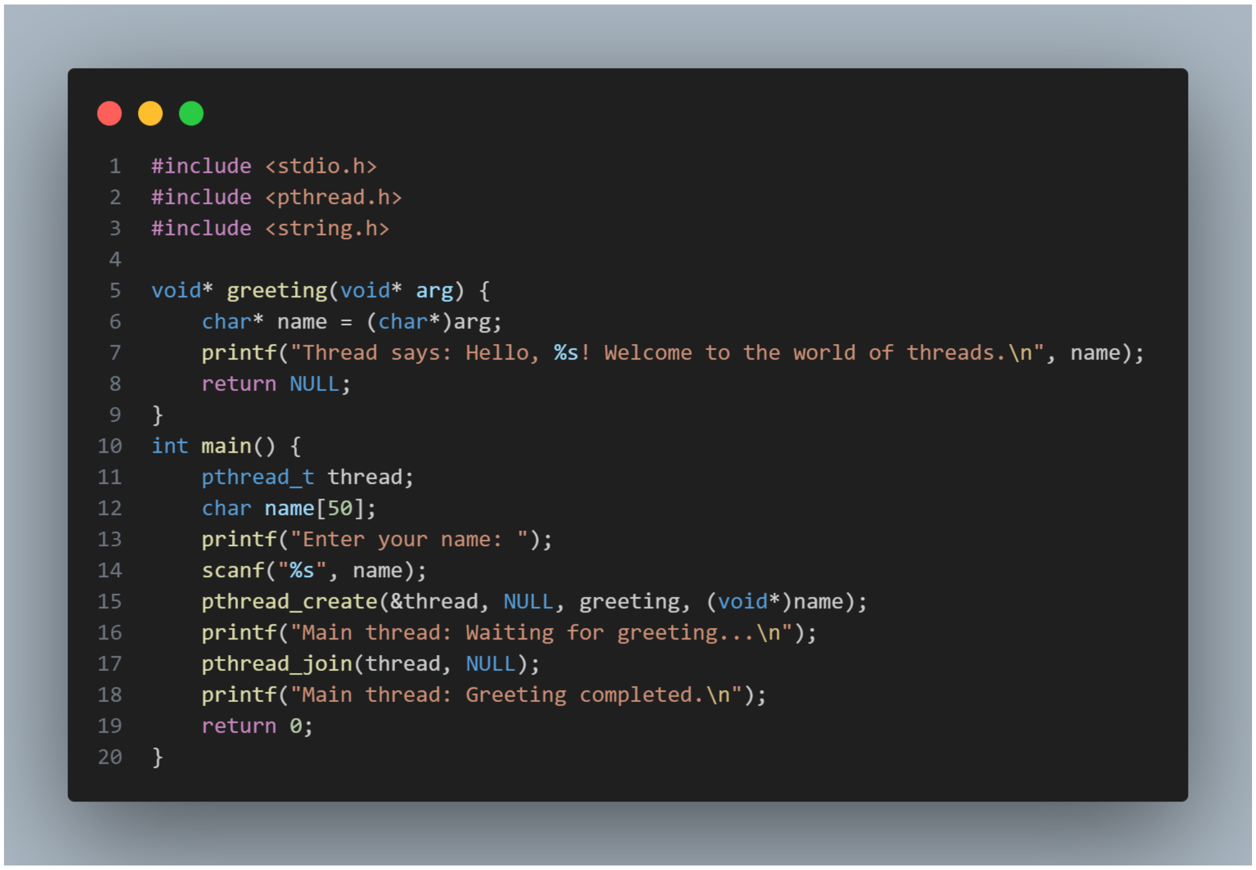


```
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/OS_Assignment 01$ gcc task1.c -o task1.out
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/OS_Assignment 01$ ./task1.out
Creating 5 threads...

Thread 1 started | Thread ID: 132269893219904 | Sleeping for 3 seconds...
Thread 2 started | Thread ID: 132269884827200 | Sleeping for 2 seconds...
Thread 3 started | Thread ID: 132269876434496 | Sleeping for 1 seconds...
Thread 4 started | Thread ID: 132269868041792 | Sleeping for 1 seconds...
Thread 5 started | Thread ID: 132269859649088 | Sleeping for 3 seconds...
Thread 3 (ID: 132269876434496) has completed its work.
Thread 4 (ID: 132269868041792) has completed its work.
Thread 2 (ID: 132269884827200) has completed its work.
Thread 1 (ID: 132269893219904) has completed its work.
Thread 5 (ID: 132269859649088) has completed its work.

All threads have completed.
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/OS_Assignment 01$
```

Task 2 – Personalized Greeting Thread

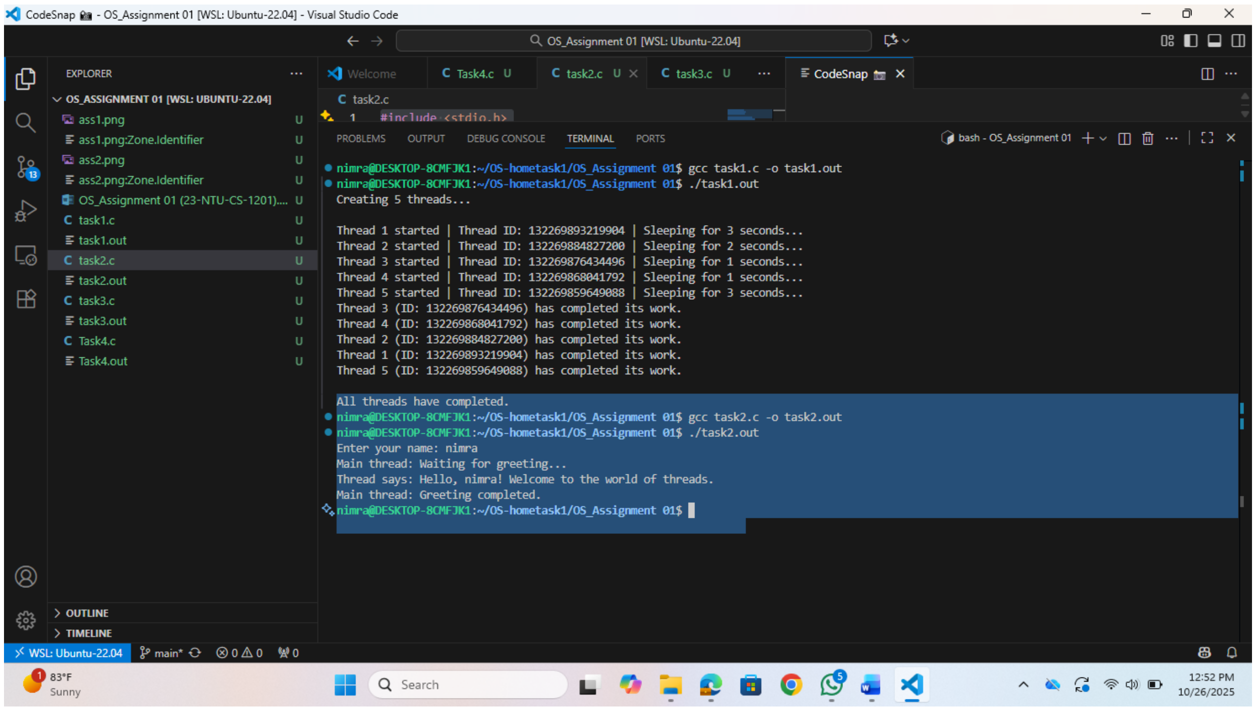
Write a C program that:
• Creates a thread that prints a personalized greeting message.
• The message includes the user's name passed as an argument to the thread.
• The main thread prints "Main thread: Waiting for greeting..." before joining the created thread.

Example Output:
Main thread: Waiting for greeting...
Thread says: Hello, Ali! Welcome to the world of threads.
Main thread: Greeting completed.



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

void* greeting(void* arg) {
    char* name = (char*)arg;
    printf("Thread says: Hello, %s! Welcome to the world of threads.\n", name);
    return NULL;
}
int main() {
    pthread_t thread;
    char name[50];
    printf("Enter your name: ");
    scanf("%s", name);
    pthread_create(&thread, NULL, greeting, (void*)name);
    printf("Main thread: Waiting for greeting...\n");
    pthread_join(thread, NULL);
    printf("Main thread: Greeting completed.\n");
    return 0;
}
```

OUTPUT:

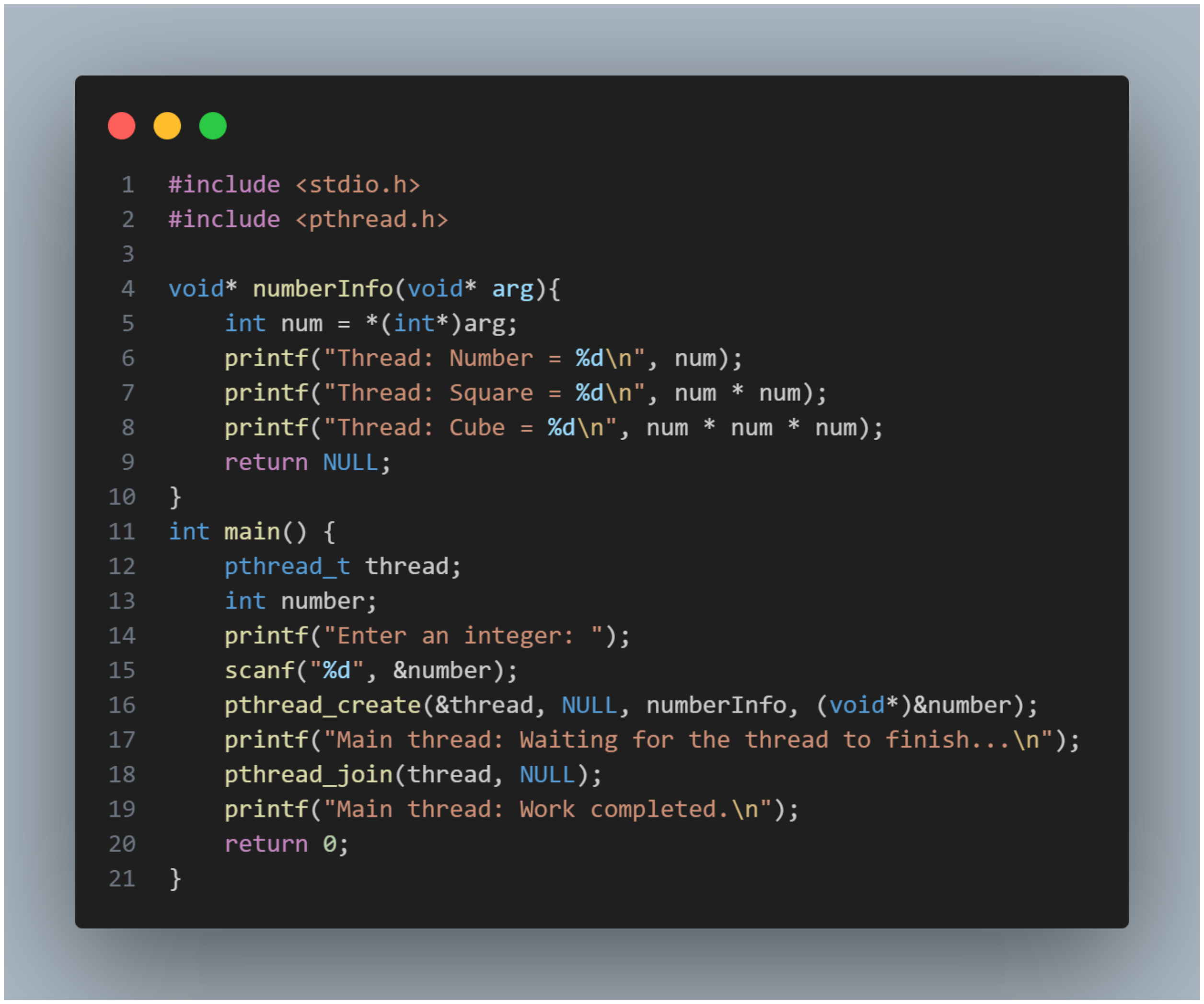


```
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/OS_Assignment 01$ gcc task1.c -o task1.out
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/OS_Assignment 01$ ./task1.out
Creating 5 threads...

Thread 1 started | Thread ID: 132269893219904 | Sleeping for 3 seconds...
Thread 2 started | Thread ID: 132269884827200 | Sleeping for 2 seconds...
Thread 3 started | Thread ID: 132269876434496 | Sleeping for 1 seconds...
Thread 4 started | Thread ID: 132269868041792 | Sleeping for 1 seconds...
Thread 5 started | Thread ID: 132269859649088 | Sleeping for 3 seconds...
Thread 3 (ID: 132269876434496) has completed its work.
Thread 4 (ID: 132269868041792) has completed its work.
Thread 2 (ID: 132269884827200) has completed its work.
Thread 1 (ID: 132269893219904) has completed its work.
Thread 5 (ID: 132269859649088) has completed its work.

All threads have completed.
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/OS_Assignment 01$ gcc task2.c -o task2.out
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/OS_Assignment 01$ ./task2.out
Enter your name: nimra
Main thread: Waiting for greeting...
Thread says: Hello, nimra! Welcome to the world of threads.
Main thread: Greeting completed.
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/OS_Assignment 01$
```

Task 3 – Number Info Thread

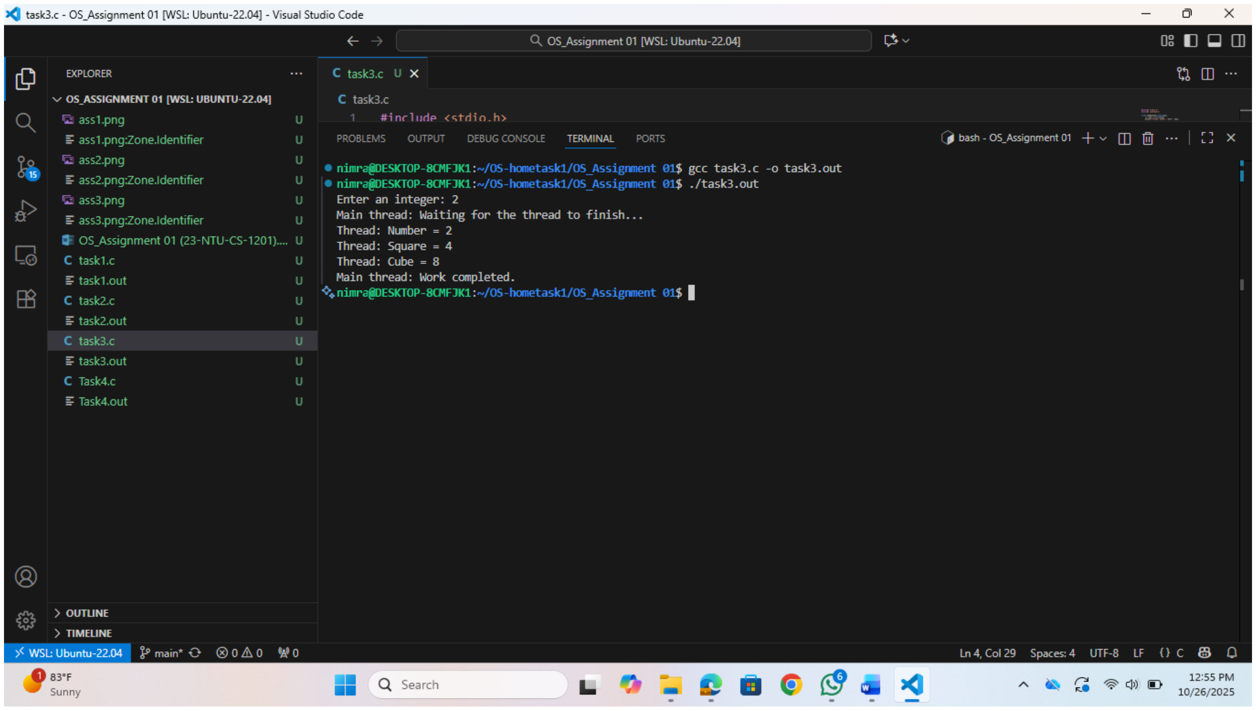
Write a program that:

- 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.”



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

void* numberInfo(void* arg){
    int num = *(int*)arg;
    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);
    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");
    return 0;
}
```

OUTPUT:

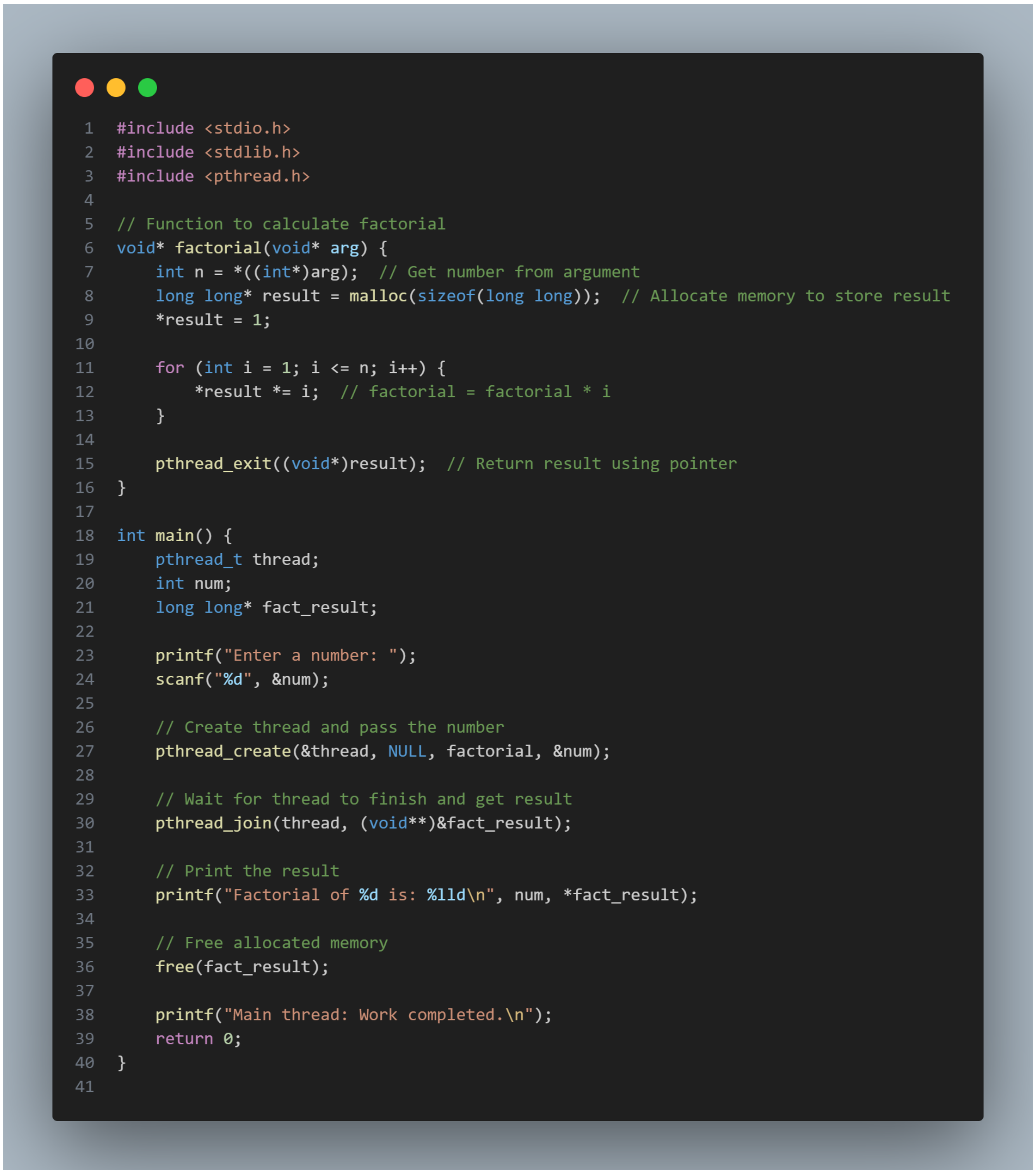


```
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/OS_Assignment 01$ gcc task3.c -o task3.out
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/OS_Assignment 01$ ./task3.out
Enter an integer: 2
Main thread: Waiting for the thread to finish...
Thread: Number = 2
Thread: Square = 4
Thread: Cube = 8
Main thread: Work completed.
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/OS_Assignment 01$
```

Task 4 – Thread Return Values

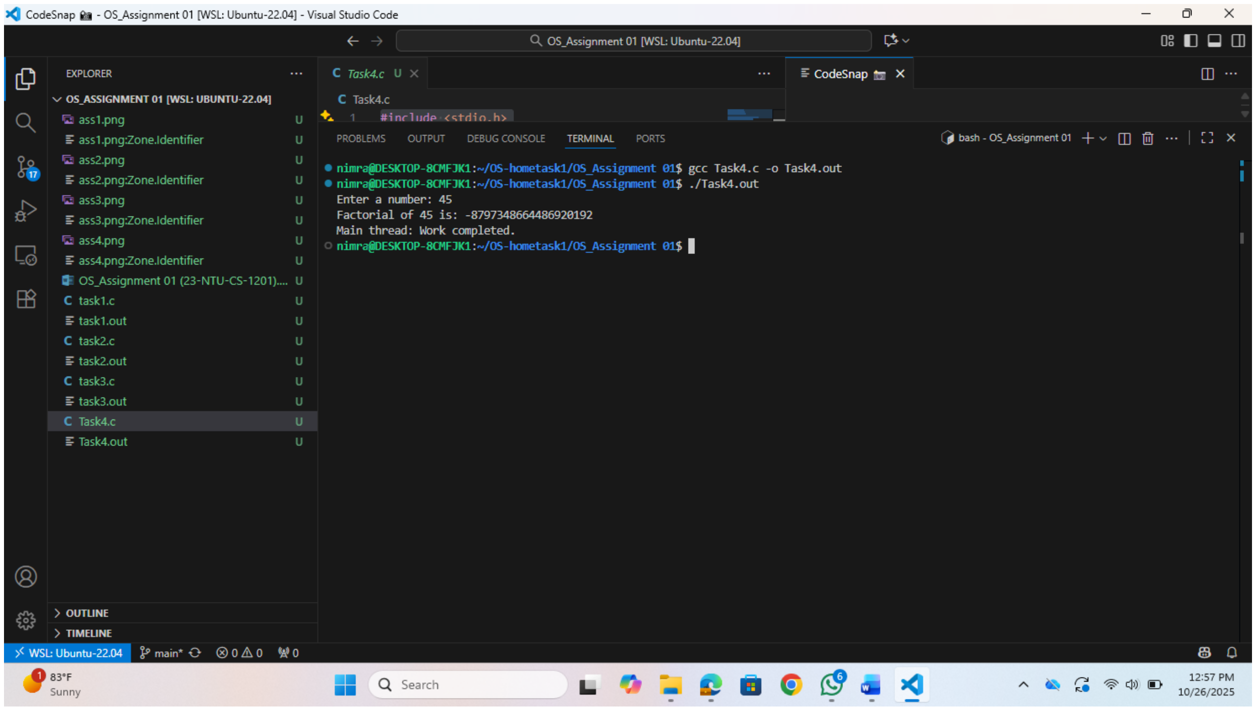
Write a program that creates a thread to compute the factorial of a number entered by the user.

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



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

// Function to calculate factorial
void* factorial(void* arg) {
    int n = *((int*)arg);  // Get number from argument
    long long* result = malloc(sizeof(long long));  // Allocate memory to store result
    *result = 1;

    for (int i = 1; i <= n; i++) {
        *result *= i;  // factorial = factorial * i
    }

    pthread_exit((void*)result);  // Return result using pointer
}

int main() {
    pthread_t thread;
    int num;
    long long* fact_result;

    printf("Enter a number: ");
    scanf("%d", &num);

    // Create thread and pass the number
    pthread_create(&thread, NULL, factorial, &num);

    // Wait for thread to finish and get result
    pthread_join(thread, (void**)&fact_result);

    // Print the result
    printf("Factorial of %d is: %lld\n", num, *fact_result);

    // Free allocated memory
    free(fact_result);

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

OUTPUT:

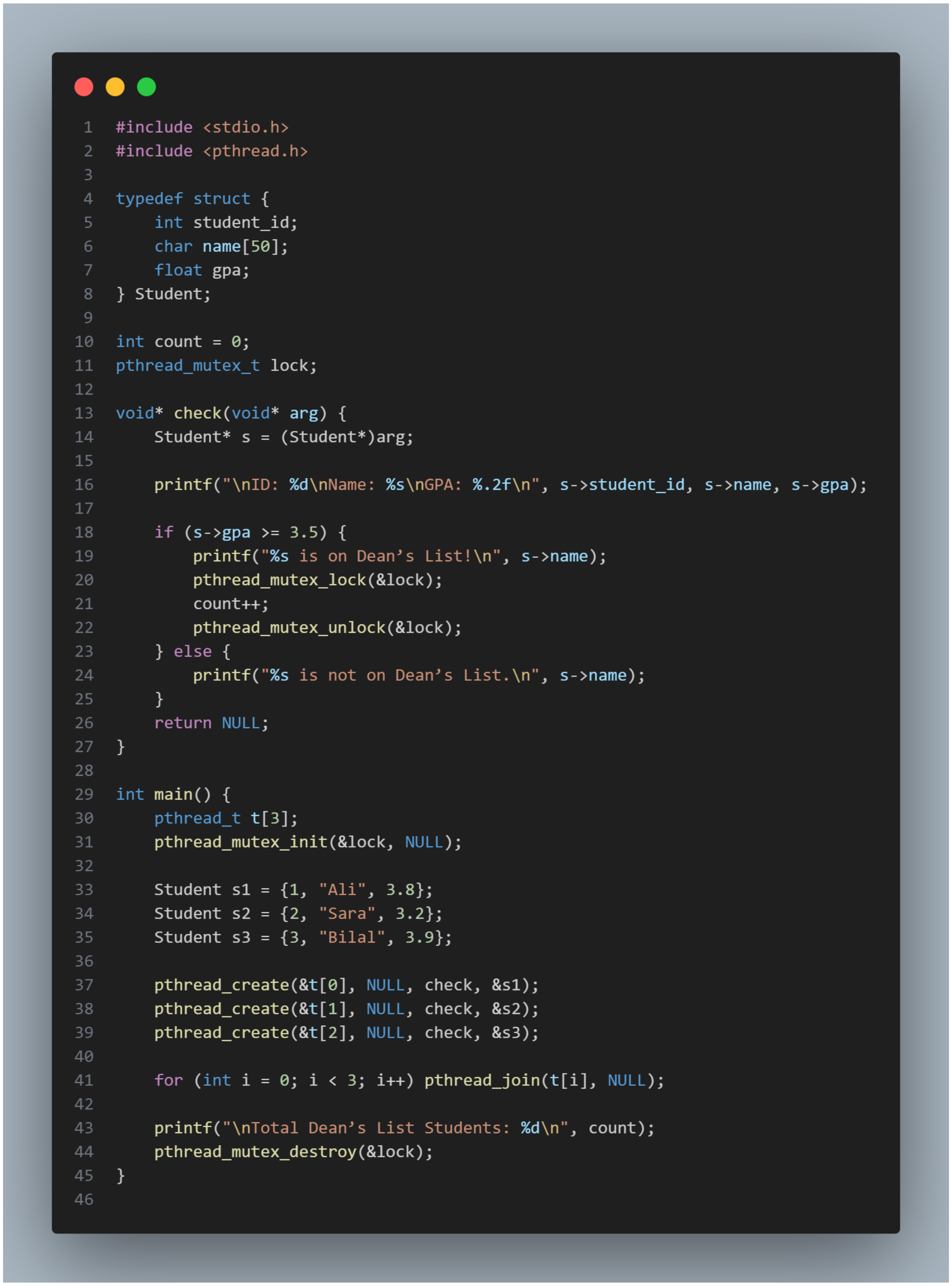


```
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/OS_Assignment 01$ gcc Task4.c -o Task4.out
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/OS_Assignment 01$ ./Task4.out
Enter a number: 45
Factorial of 45 is: -8797348664486920192
Main thread: Work completed.
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/OS_Assignment 01$
```

Task 5 – Struct-Based Thread Communication

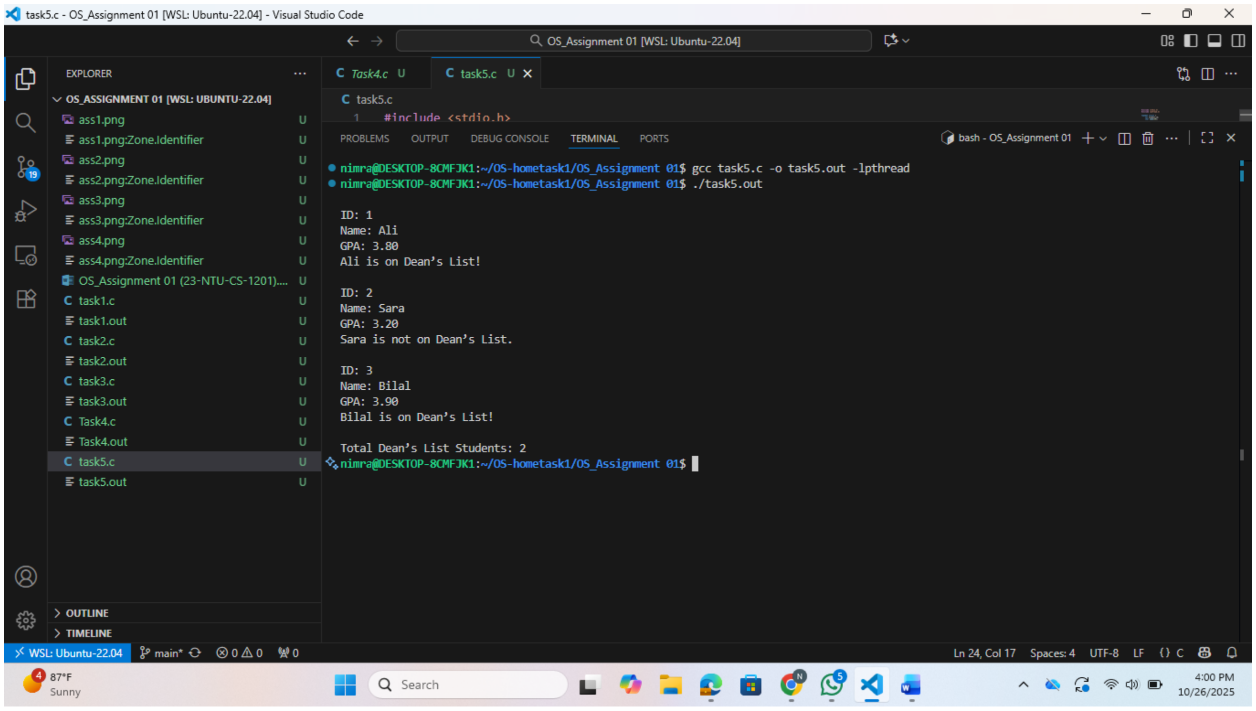
Create a program that simulates a simple student database system.

- 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.



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

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

int count = 0;
pthread_mutex_t lock;

void* check(void* arg) {
    Student* s = (Student*)arg;

    printf("\nID: %d\nName: %s\nGPA: %.2f\n", s->student_id, s->name, s->gpa);

    if (s->gpa >= 3.5) {
        printf("%s is on Dean’s List!\n", s->name);
        pthread_mutex_lock(&lock);
        count++;
        pthread_mutex_unlock(&lock);
    } else {
        printf("%s is not on Dean’s List.\n", s->name);
    }
    return NULL;
}

int main() {
    pthread_t t[3];
    pthread_mutex_init(&lock, NULL);

    Student s1 = {1, "Ali", 3.8};
    Student s2 = {2, "Sara", 3.2};
    Student s3 = {3, "Bilal", 3.9};

    pthread_create(&t[0], NULL, check, &s1);
    pthread_create(&t[1], NULL, check, &s2);
    pthread_create(&t[2], NULL, check, &s3);

    for (int i = 0; i < 3; i++) pthread_join(t[i], NULL);

    printf("\nTotal Dean’s List Students: %d\n", count);
    pthread_mutex_destroy(&lock);
}

```

OUTPUT:



```
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/OS_Assignment 01$ gcc task5.c -o task5.out -lpthread
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/OS_Assignment 01$ ./task5.out

ID: 1
Name: Ali
GPA: 3.80
Ali is on Dean's List!

ID: 2
Name: Sara
GPA: 3.20
Sara is not on Dean's List.

ID: 3
Name: Bilal
GPA: 3.90
Bilal is on Dean's List!

Total Dean's List Students: 2
nimra@DESKTOP-8CMFJK1:~/OS-hometask1/OS_Assignment 01$
```

Section-B: Short Questions

1. Answer all questions briefly and clearly.

2.Define an Operating System in a single line?

Operating System (OS):

An Operating System (OS) is system software that manages computer hardware, software resources, and provides services for computer programs.

3. What is the primary function of the CPU scheduler?

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.

4. List any three states of a process?

Three states of a process are:

1. Ready: The process is waiting to be assigned to the CPU.
2. Running: The process is currently being executed by the CPU.
3. Blocked (or waiting): The process is waiting for some event or I/O operation to complete.

5. What is meant by a Process Control Block (PCB)?

A Process Control Block (PCB) is a data structure in the operating system that contains important information about a specific process, such as its process ID, state, CPU registers, memory management information, and scheduling details.

6. Differentiate between a process and a program.

The difference between a process and a program is

Aspect	Program	Process
Definition	A program is a set of instructions written to perform a specific task.	A process is a program in execution (an active instance of a program).
State	It is passive, stored on disk.	It is active, loaded into memory and being executed.
Resources	Does not require resources like CPU or memory.	Requires resources such as CPU time, memory, and I/O devices.

Example	chrome.exe stored on disk.	A running instance of Chrome browsing a website.

7. What do you understand by context switching?

Context switching is the process of saving the state of a currently running process and restoring the state of another process so that the CPU can switch from executing one process to another.

8. Define CPU utilization and throughput.

CPU Utilization:

It is the percentage of time the CPU is actively executing processes rather than being idle.

Throughput:

It is the number of processes completed by the CPU in a given amount of time.

9. What is the turnaround time of a process?

Turnaround time is the total amount of time taken by a process from the moment it enters the system (arrives) until it finishes execution. It measures how long the process takes to complete.
Formula:
Turnaround Time = Completion Time – Arrival Time

10. How is waiting time calculated in process scheduling?

Waiting time is the total time a process spends waiting in the ready queue before getting CPU time for execution. It shows how long the process had to wait to start running.

Formula:
Waiting Time = Turnaround Time – Burst Time

11. Define response time in CPU scheduling.

Response time is the time between the process arrival and the first time it gets the CPU.
It indicates how quickly the system starts responding to a process.
Formula:
Response Time = Time of First CPU Response – Arrival Time

12. 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.

13. 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.

14. State any two advantages of the Round Robin scheduling algorithm.

1. Fairness: Every process gets an equal share of CPU time using a fixed time quantum.
2. Good response time: It is suitable for time-sharing systems, making it responsive for interactive users. It prevents starvation since all processes get

a turn.

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

A main drawback of SJF is starvation i.e. longer processes may be continuously delayed if short processes keep arriving and it requires knowledge of burst time in advance, which is often not known.

16. Define CPU idle time.

CPU idle time is the period when the CPU is not executing any process — it remains idle because there are no processes ready to run. It represents wasted CPU time that reduces overall efficiency.
Example: This can happen when all processes are waiting for I/O operations to complete.

17. State two common goals of CPU scheduling algorithms.

1. **Maximize CPU utilization:** Keep the CPU as busy as possible — avoid idle time.
2. **Minimize waiting and turnaround time:** So, processes complete faster and the system stays efficient. Other goals include fairness, high throughput, and quick response time.

18. List two possible reasons for process termination.

1. **Normal completion:** The process finishes its task successfully and calls exit().

2. **Error or interruption:** The process is terminated due to errors (like invalid memory access) or by another process e.g., the parent kills it.
   Other reasons: time limit exceeded, user request, or lack of resources.

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

- **exit()** → Called by a process when it completes; it releases all resources and ends the process.
- **wait()** → Used by a parent process to wait for its child process to finish before continuing.
  Example: In UNIX, the parent calls wait() after creating a child using fork().

20. Differentiate between shared memory and message-passing models of inter-process communication (IPC).

Feature	**Shared Memory**	**Message Passing**
Communication Method	Processes share a common memory region.	Processes exchange messages through OS.
Speed	Faster as it direct access to memory.	Slower as it involves kernel for every message.
Synchronization	Must be handled by processes e.g., using semaphores.	OS handles synchronization.
Example	Used in databases or multimedia systems.	Used in distributed systems or microservices.

21. Differentiate between a thread and a process.

Aspect	Process	Thread
Definition	Independent program in execution.	A lightweight sub-unit of a process.
Memory	Has its own memory space.	Shares memory with other threads in the same process.
Creation Time	Slower to create.	Faster to create.
Communication	Inter-process communication needed.	Easier as threads share data directly.

Example: A browser (process) has multiple tabs (threads).

22. Define multithreading.

Multithreading is the ability of a CPU or program to execute multiple threads concurrently within a single process. Each thread performs a different task but shares the same resources like memory.

Example:

In a web browser one thread loads a page, another handles user input, another plays audio.

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

Type	CPU-bound Process	I/O-bound Process
Definition	Spends most time using CPU for computation.	Spends most time waiting for I/O operations.
Example	Data encryption, calculations, simulations.	File reading, database access, user input.
Performance Goal	Needs faster CPU.	Needs faster I/O devices.

24. What are the main responsibilities of the dispatcher?

The dispatcher is part of the CPU scheduler that handles the actual process switch. It:

1. Performs context switching means it saves the old process state and loads the new one.
2. Switches to user mode from kernel mode.
3. Jumps to the correct instruction to start or resume the process. Dispatcher speed affects CPU efficiency, faster switching means less overhead.

25. What is starvation and aging in process scheduling?

- Starvation: When a process never gets CPU time because other high-priority processes keep running again and again.
  Example: A low-priority job keeps waiting forever while new high-priority jobs arrive.
- Aging: A method to stop starvation. It slowly increases the priority of old waiting processes so they will finally get CPU time.
  Example: Every 10 seconds, a waiting process gets a small priority boost.

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

- In Round Robin scheduling, each process gets CPU for a fixed small time — this fixed time is called time quantum.
  Example: If time quantum = 4ms, each process runs for 4ms, then CPU moves to the next one.

27.What happens if time quantum is too large or too small?

- Too large: It becomes like FCFS (First Come First Serve). The response time becomes bad for short or interactive jobs.
- Too small: There will be too many context switches (CPU changes processes too often), wasting CPU time.
  Best case: Choose a balanced time quantum (not too big, not too small).

28. What is Turnaround Ratio (TR/TS)?

- Turnaround Time (TR): Total time from arrival to completion.
- Service Time (TS): Actual CPU time used by the process.
- Turnaround Ratio = TR / TS → shows how long the process waited compared to how much CPU it needed.
  Example: TR = 30ms, TS = 10ms → Ratio = 30/10 = 3.

29.What is the purpose of a ready queue?

- The ready queue holds all processes that are ready to use the CPU but are waiting for their turn.
- The CPU scheduler picks the next process to run from this queue.
  Example: If 3 processes are ready, they all wait in the ready queue until CPU becomes free.

30. Difference between CPU burst and I/O burst?

- CPU burst: The process is using the CPU to do calculations or processing.
- I/O burst: The process is waiting for input/output (like reading from disk or printing).
  Example: A program does some calculations (CPU burst), then saves data to disk (I/O burst).

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

- FCFS (First Come First Serve) and Round Robin (RR) are starvation-free because every process gets a fair chance.
    - In FCFS: whoever comes first, runs first.
    - In RR: every process gets CPU for a small fixed time (time slice), so none are ignored.

But Priority or SJF (Shortest Job First) can cause starvation if some jobs always get skipped.

32. Main steps in process creation in UNIX?

1. Parent process calls fork() → creates a new process (child).
2. Child process gets a new Process ID (PID) and a copy of parent's data.
3. Child can call exec() → replace its program with another one (for example, "ls" command).
4. OS adds the child to the ready queue.
5. Parent can call wait() to wait for the child to finish.
6. When child finishes, OS frees its memory and resources.

Example: The shell uses fork() then exec("ls") parent waits until ls command is done.

33. Define Zombie and Orphan Processes

- **Zombie Process:**
  A process that has **finished execution** but still remains in the process table because its **parent has not collected its exit status**.
  Example: A child process ends, but the parent didn't call wait(), so the child becomes a zombie.

- **Orphan Process:**
  A process whose **parent process has terminated** before it finishes execution. These are adopted by the **init process (PID 1)**.
  Example: If a parent dies while the child is still running, the child becomes an orphan.

34. Differentiate between Priority Scheduling and Shortest Job First (SJF)

Feature	**Priority Scheduling**	Shortest Job First (SJF)
Basis of Selection	Process with **highest priority** runs first. Can be preemptive or non-preemptive.	Process with **shortest CPU burst time** runs first.
Preemption	Can be preemptive or non-preemptive.	Can be preemptive (Shortest Remaining Time First) or non-preemptive.
Problem	May cause **starvation** of low-priority processes.	May cause **starvation** of long jobs.
Example	CPU given to process with priority 1 before priority 3	CPU given to process needing 2 sec before one needing 6 sec.

35. Define Context Switch Time and explain why it is considered overhead

- **Definition:**
  Context Switch Time is the time the CPU takes to **save the state of the current process** and **load the state of the next process** during a process switch.
- **Why it's overhead:**
  Because **no useful work** (no user process execution) is done during this time — CPU only manages switching tasks. So it **reduces efficiency**.

36. List and briefly describe the three levels of schedulers in an Operating System.

Scheduler Type	**Description**
Long-Term Scheduler (Job Scheduler)	Decides **which processes to enter** the ready queue from the job pool (controls degree of multiprogramming).
Medium-Term Scheduler	**Swaps processes in and out** of main memory to control load and improve performance.
Short-Term Scheduler (CPU Scheduler)	Selects **which ready process** will be executed next by the CPU. Works very frequently.

37. Differentiate between User Mode and Kernel Mode in an Operating System.

Feature	**User Mode**	**Kernel Mode**
Access Level	Limited access to system resources.	Full access to hardware and all system resources.
Who Works Here	User applications (like Word, Chrome, etc.).	Operating System code and device drivers.
Caused By	When user programs run normally.	When system calls or interrupts occur.
Example	Running a text editor.	Performing a file save operation (handled by OS).

Section-C: Technical / Analytical Questions (4 marks each)

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

Process Life Cycle

A process life cycle represents the different states a process goes through during its execution, from creation to termination. The operating system manages these states and transitions to ensure smooth execution and resource utilization.

States of a Process

- **New:**
  The process is being created and initialized by the operating system.
- **Ready:**
  The process is loaded into main memory and waiting to be assigned to the CPU.
- **Running:**
  The process is currently being executed by the CPU.
- **Waiting (or blocked):**
  The process cannot continue until some event occurs (e.g., I/O completion).
- **Terminated (or Exit):**
  The process has finished execution and is removed from the system.

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

A context switch occurs when the CPU changes from executing one process (or thread) to another. During this switch, the state (context) of the currently running process must be saved so that it can continue later from the same point. Then, the saved state of the next process is loaded into the CPU registers to resume its execution. Context switching does not perform any useful computation — it is a pure overhead because the CPU time is spent only on saving and restoring states, not on actual process work. The more complex the operating system and process control block (PCB), the longer the context switch takes.

<u>Information Saved and Restored During Context Switch:</u>

When switching processes, the system saves the current process's context in its Process Control Block (PCB) and restores the next process's context from its PCB. The context typically includes:

1. **Process State** – Running, Ready, or Waiting.

2. **Program Counter (PC)** – Address of the next instruction to execute.
3. **CPU Registers** – All CPU register contents (general-purpose and special).
4. **Memory Management Info** – Page tables, base and limit registers.
5. **Accounting Info** – CPU usage, time limits, priority, etc.
6. **I/O Status Info** – Devices allocated, list of open files.

Q # 3: Components of a Process Control Block (PCB)

A Process Control Block (PCB) is a data structure maintained by the operating system that stores all important information about a process. It acts like the identity card or record sheet of a process, i.e. helping the OS manage and switch between processes efficiently.

Main Components of a PCB:

1. **Process State**
   - Shows the current status of the process e.g. *New, Ready, Running, Waiting,* or *Terminated*.
   - Helps the OS know what stage the process is in.
2. **Program Counter (PC)**
   - Holds the address of the next instruction the process will execute.
   - When a context switch happens, this value ensures the process resumes from the correct point.
3. **CPU Registers**
   - Stores all CPU register values used by the process, such as accumulators, index registers, stack pointers, etc.
   - These are saved and restored during a context switch.

4. **CPU Scheduling Information**
    - Includes process priority, scheduling queue pointers, and other data used by the scheduler to decide which process runs next.
5. **Memory Management Information**
    - Contains details about the process's memory allocation, such as base and limit registers, page tables, or segment tables.
    - Used by the OS to manage process address space.
6. **Accounting Information**
    - Keeps track of CPU usage, execution time, job or process ID, and time limits.
    - Useful for performance monitoring and billing (in multi-user systems).
7. **I/O Status Information**
    - Lists I/O devices allocated to the process and the list of open files it is using.
    - Helps the OS manage resources and handle input/output efficiently.

Q # 4: Difference between Long-Term, Medium-Term, and Short-Term Schedulers

Schedulers are special parts of the operating system that decide which process runs, when, and for how long.
They manage how processes move between different states (new, ready, running, waiting, terminated).

1. Long-Term Scheduler (Job Scheduler)

- Purpose: Decides which processes are allowed to enter the ready queue.
- When used: When a new process is created.

- Goal: Controls the degree of multiprogramming (how many processes are in memory).
- Example:
  If 10 jobs are waiting in a batch system, the long-term scheduler may choose only 4 to load into memory for execution.
- Frequency: Runs less often.

Example :
It’s like a gatekeeper — deciding which jobs can enter the CPU area.

2. Medium-Term Scheduler (Swapper)

- Purpose: Temporarily removes (suspends) processes from main memory to reduce load and later brings them back (resumes).
- When used: When the system is overloaded or needs more memory.
- Goal: Improves performance and memory management.
- Example:
  A process is waiting for a long I/O operation — the medium-term scheduler swaps it out from memory and brings another process in.
- Frequency: Runs occasionally.

Example :
It’s like a store manager — moving processes in or out of the memory shelves.

3. Short-Term Scheduler (CPU Scheduler)

- Purpose: Decides which ready process will get the CPU next.
- When used: After every CPU burst or I/O completion.
- Goal: Maximize CPU utilization and responsiveness.

- Example:
  When multiple processes are ready, the short-term scheduler picks one using algorithms like FCFS, SJF, or Round Robin.
- Frequency: Runs very frequently (many times per second).

Example:
It’s like a referee — quickly deciding which player (process) gets the ball (CPU) next.

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

CPU Scheduling Criteria and Their Optimization Goals

CPU scheduling is an important function of the operating system that decides **which process should run on the CPU** at a particular time. To evaluate and compare different CPU scheduling algorithms, several **performance criteria** are used. These criteria help determine how efficiently the CPU and system resources are being used. The main CPU scheduling criteria are **CPU Utilization, Throughput, Turnaround Time, Waiting Time,** and **Response Time**.

1. CPU Utilization

Definition:
CPU utilization refers to the percentage of time the CPU is actively working (not idle). It measures how effectively the CPU is being used.

Explanation:
A good scheduling algorithm keeps the CPU as busy as possible by reducing idle time.
If the CPU is idle for a long period, it means the scheduling method is inefficient.

Example:
If out of 1 second, the CPU was busy for 0.9 seconds, then the CPU utilization is 90%.

Optimization Goal:
To **maximize CPU utilization**, ideally between **80% to 100%**, so that the processor remains productive most of the time.

2. Throughput

Definition:
Throughput is the total number of processes that are completed (executed) in a specific amount of time.

Explanation:
Higher throughput means more work is being done in less time, which indicates better performance.
Scheduling algorithms aim to increase the number of completed processes per second or per minute.

Example:
If 10 processes are completed in 5 seconds, the throughput is 2 processes per second.

Optimization Goal:
To **maximize throughput** so that more processes finish execution in less time.

3. Turnaround Time

Definition:
Turnaround time is the **total time taken** from the moment a process enters the system (submitted) until it is fully completed.

Formula:

$$\text{Turnaround Time} = \text{Waiting Time} + \text{Execution (CPU) Time}$$

Explanation:
It includes all the time spent by the process in waiting, executing, and performing

I/O operations.
Lower turnaround time means processes finish faster.

Example:
If a process is submitted at time 0 and finishes at time 20 seconds, then its turnaround time is 20 seconds.

Optimization Goal:
To **minimize turnaround time**, ensuring that each process completes as soon as possible.

4. Waiting Time

Definition:
Waiting time is the total time a process spends waiting in the **ready queue** before it gets CPU time.

Explanation:
Every process has to wait for the CPU to become available.
Good scheduling algorithms try to reduce this waiting period to improve overall system performance.

Example:
If a process waits 10 seconds before it gets CPU time, its waiting time is 10 seconds.

Optimization Goal:
To **minimize waiting time** so that processes do not stay idle in the queue for too long.

5. Response Time

Definition:
Response time is the time taken from when a process is submitted until the **first response** is produced (i.e., the first time it gets the CPU).

Explanation:
This criterion is especially important for **interactive systems**, such as user interfaces or online applications, where users expect quick feedback.
Lower response time improves user satisfaction and system responsiveness.

Example:
If you open a software application and it starts showing output after 2 seconds, then the response time is 2 seconds.

Optimization Goal:
To **minimize response time** so that users get faster feedback and the system feels more responsive.

<u>Section-D: CPU Scheduling Calculations</u>

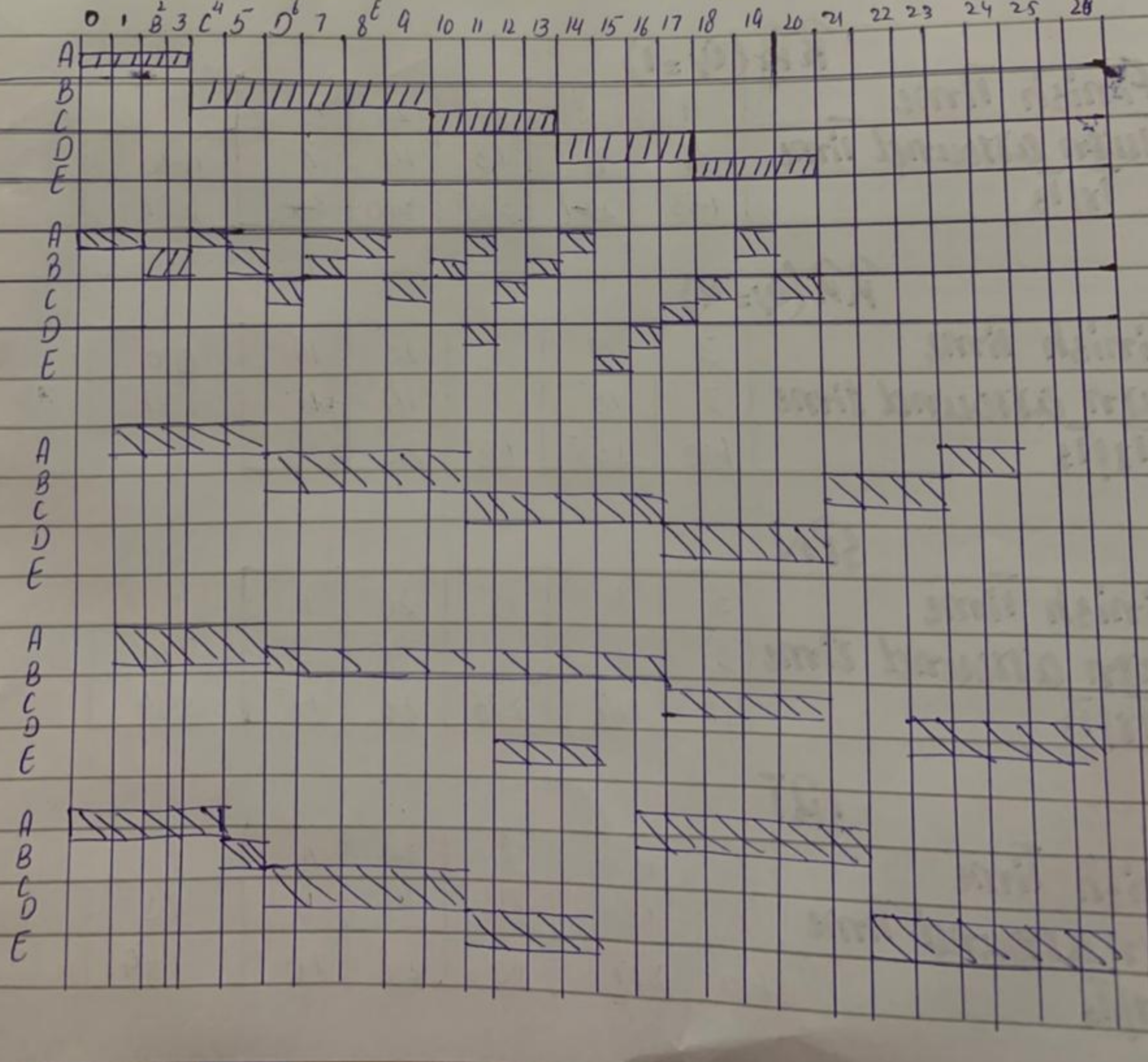
• Perform the following calculations for each part (A–C).

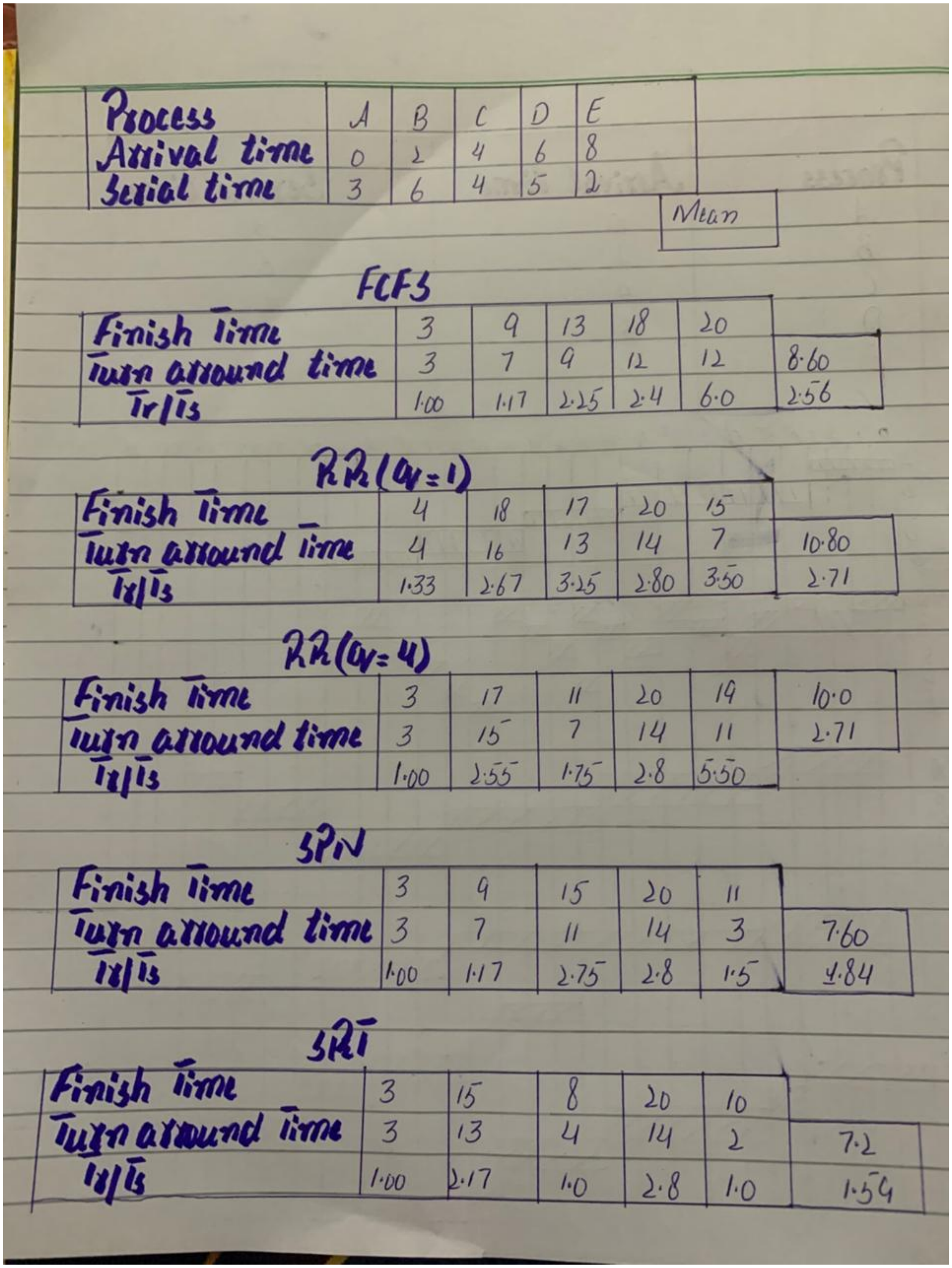
• a) Draw Gantt charts for FCFS, RR (Q=4), SJF, and SRTF.

• b) Compute Waiting Time, Turnaround Time, TR/TS ratio, and CPU Idle Time.

•c) Compare average values and identify which algorithm performs best.

Process	Arrival time	Service time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2





Process	A	B	C	D	E
Arrival time	0	2	4	6	8
Serial time	3	6	4	5	2

Mean

FCFS

Finish time	3	9	13	18	20	
Turn arround time	3	7	9	12	12	8.60
Tr/Ts	1.00	1.17	2.25	2.4	6.0	2.56

RR (q=1)

Finish Time	4	18	17	20	15	
Turn arround time	4	16	13	14	7	10.80
Tr/Ts	1.33	2.67	3.25	2.80	3.50	2.71

RR (q=4)

Finish time	3	17	11	20	19	10.0
turn arround time	3	15	7	14	11	2.71
Tr/Ts	1.00	2.55	1.75	2.8	5.50	

SPN

Finish time	3	9	15	20	11	
Turn arround time	3	7	11	14	3	7.60
Tr/Ts	1.00	1.17	2.75	2.8	1.5	1.84

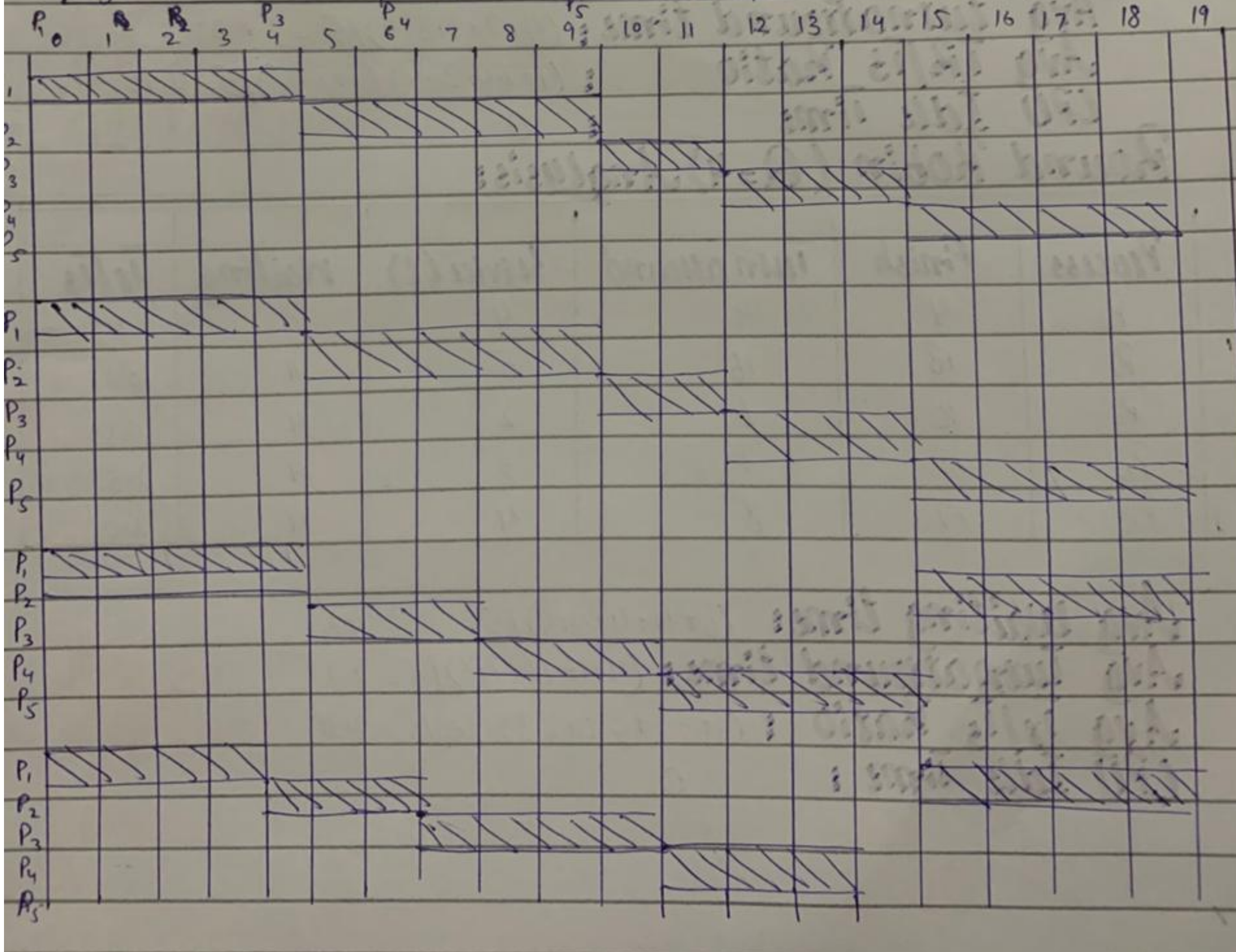
SRT

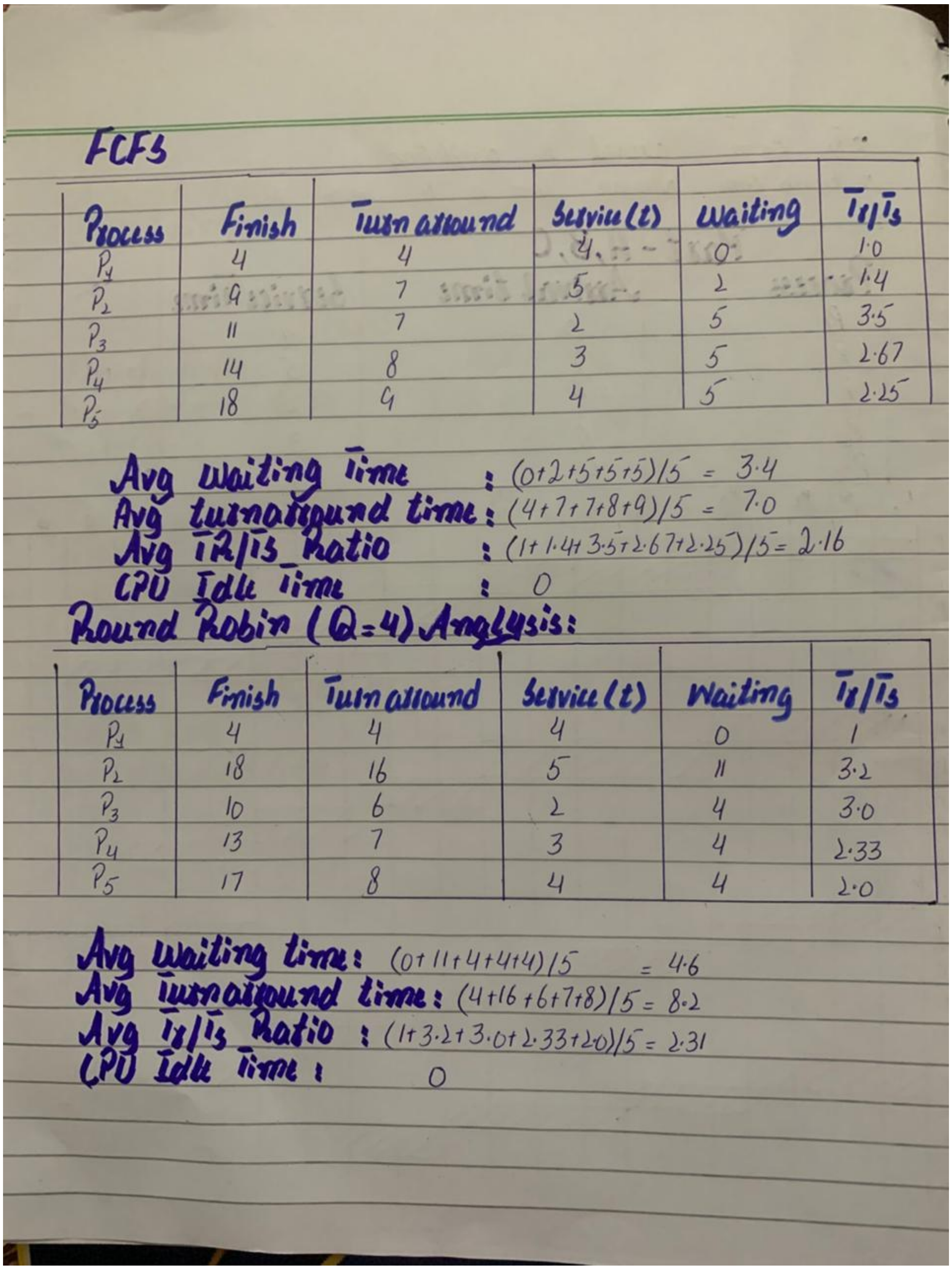
Finish time	3	15	8	20	10	
Turn arround time	3	13	4	14	2	7.2
Tr/Ts	1.00	2.17	1.0	2.8	1.0	1.59

∴ T_r From arrival to completion
∴ Finish time: Measure from 0 to finish process.

Part - A,B,C

Process	Arrival time	Service Time
P_1	0	4
P_2	2	5
P_3	4	2
P_4	6	3
P_5	9	4





FCFS

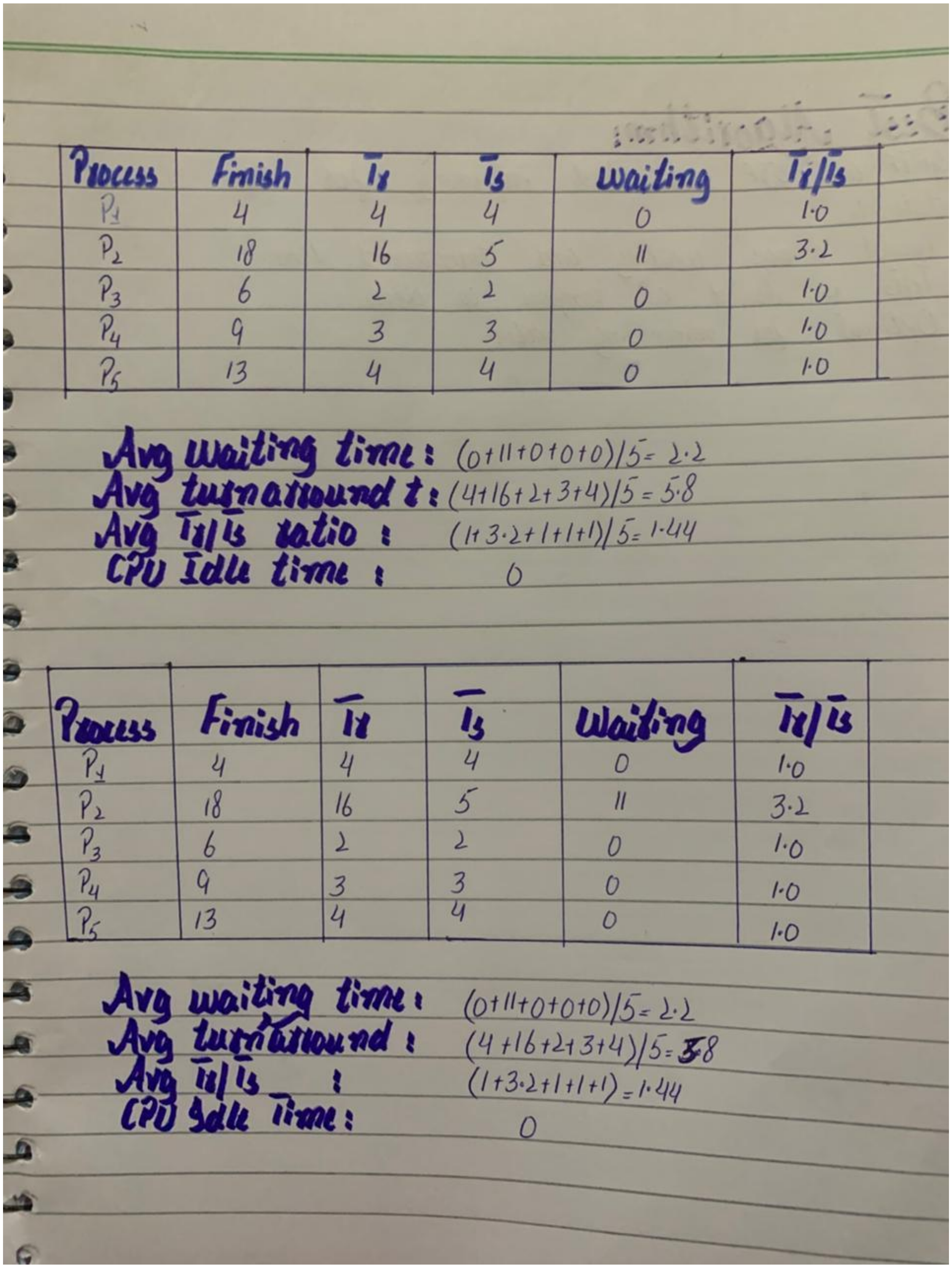
Process	Finish	Turn around	Service (t)	Waiting	Tr/Ts
P_1	4	4	4	0	1.0
P_2	9	7	5	2	1.4
P_3	11	7	2	5	3.5
P_4	14	8	3	5	2.67
P_5	18	9	4	5	2.25

Avg waiting Time : (0+2+5+5+5)/5 = 3.4
Avg turnaround time : (4+7+7+8+9)/5 = 7.0
Avg TR/TS_Ratio : (1+1.4+3.5+2.67+2.25)/5 = 2.16
CPU Idle Time : 0

Round Robin (Q=4) Analysis:

Process	Finish	Turn around	Service (t)	Waiting	Tr/Ts
P_1	4	4	4	0	1
P_2	18	16	5	11	3.2
P_3	10	6	2	4	3.0
P_4	13	7	3	4	2.33
P_5	17	8	4	4	2.0

Avg waiting time: (0+11+4+4+4)/5 = 4.6
Avg Turnaround time: (4+16+6+7+8)/5 = 8.2
Avg Tr/Ts_Ratio : (1+3.2+3.0+2.33+2.0)/5 = 2.31
CPU Idle Time : 0

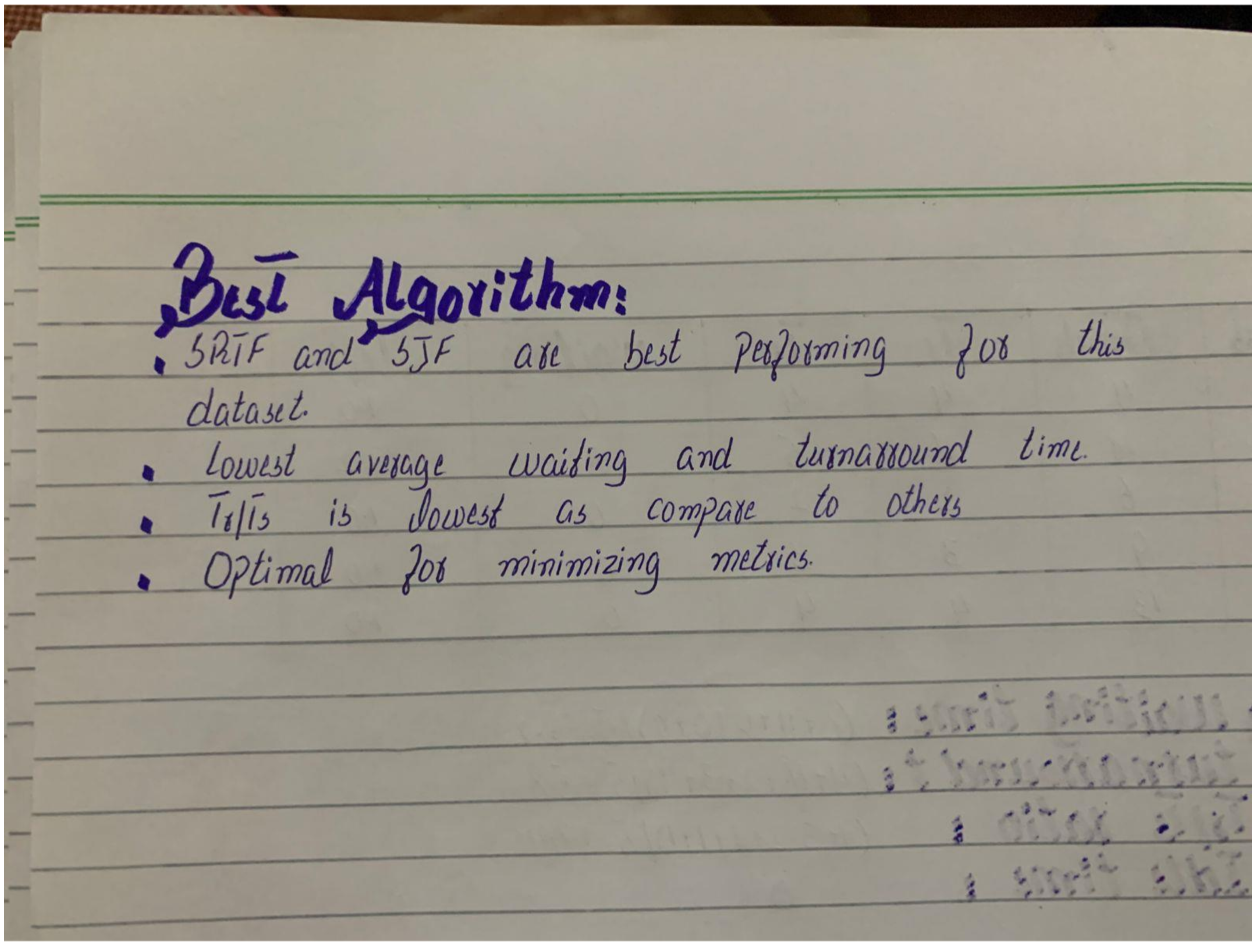


Process	Finish	$\overline{T_r}$	$\overline{T_s}$	waiting	$\overline{T_r}/\overline{T_s}$
P_1	4	4	4	0	1.0
P_2	18	16	5	11	3.2
P_3	6	2	2	0	1.0
P_4	9	3	3	0	1.0
P_5	13	4	4	0	1.0

Avg waiting time: $(0+11+0+0+0)/5 = 2.2$
Avg turnaround t: $(4+16+2+3+4)/5 = 5.8$
Avg $\overline{T_r}/\overline{T_s}$ ratio: $(1+3.2+1+1+1)/5 = 1.44$
CPU Idle time: 0

Process	Finish	$\overline{T_r}$	$\overline{T_s}$	Waiting	$\overline{T_r}/\overline{T_s}$
P_1	4	4	4	0	1.0
P_2	18	16	5	11	3.2
P_3	6	2	2	0	1.0
P_4	9	3	3	0	1.0
P_5	13	4	4	0	1.0

Avg waiting time: $(0+11+0+0+0)/5 = 2.2$
Avg turnaround: $(4+16+2+3+4)/5 = 5.8$
Avg $\overline{T_r}/\overline{T_s}$: $(1+3.2+1+1+1) = 1.44$
CPU Idle Time: 0



Best Algorithm:

- SRTF and SJF are best performing for this dataset.
- Lowest average waiting and turnarround time.
- Tr/Ts is lowest as compare to others
- Optimal for minimizing metrics.