National Textile University

Department of Computer Science

Subject: Operating System

Submitted to: Sir Nasir

Submitted by: Maha

Reg. number: 23-NTU-CS-1170

Assignment 1

Semester:5th

Section-A: Programming Tasks

Task 1 – Thread Information Display

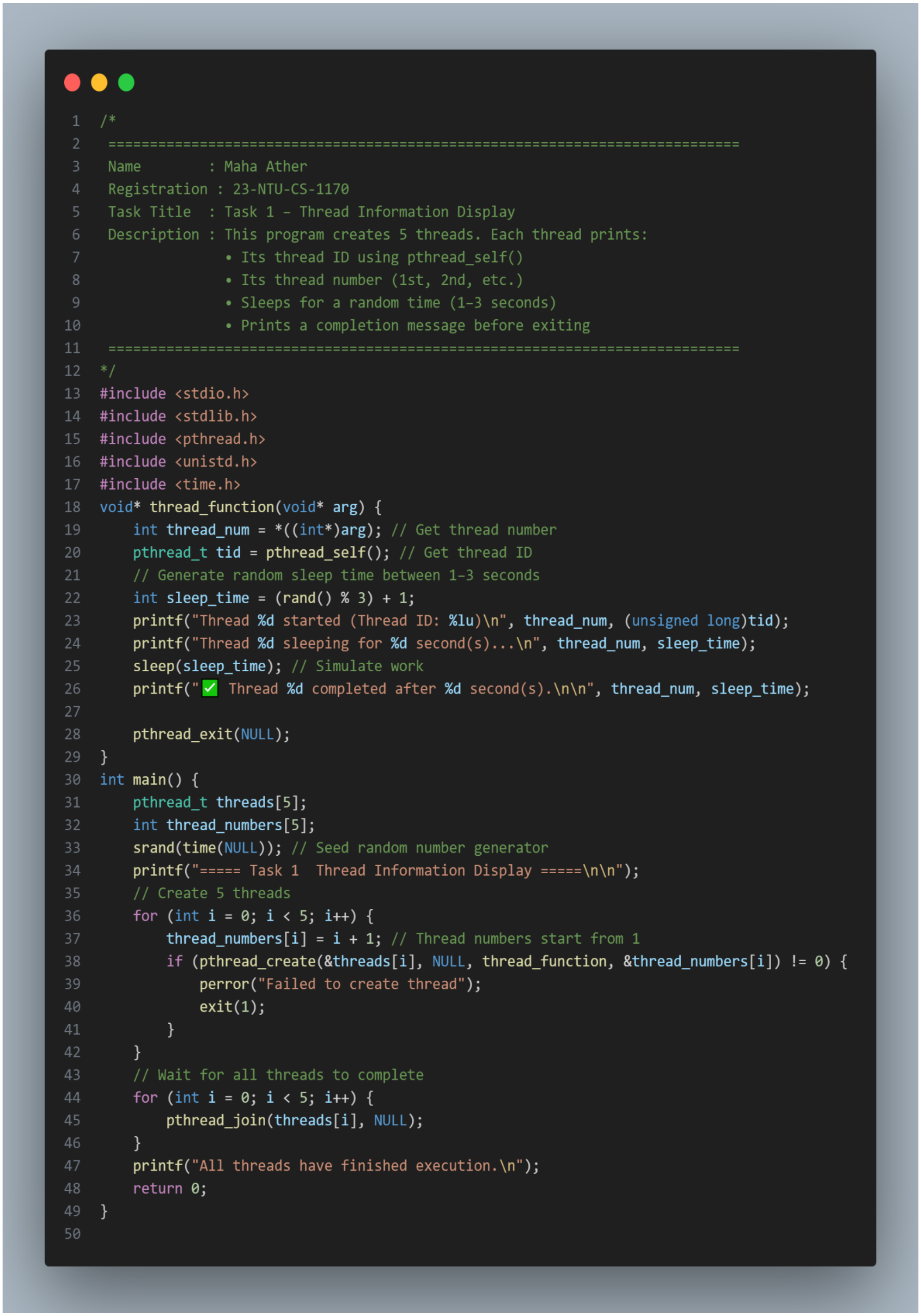
Write a program that creates 5 threads.

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.

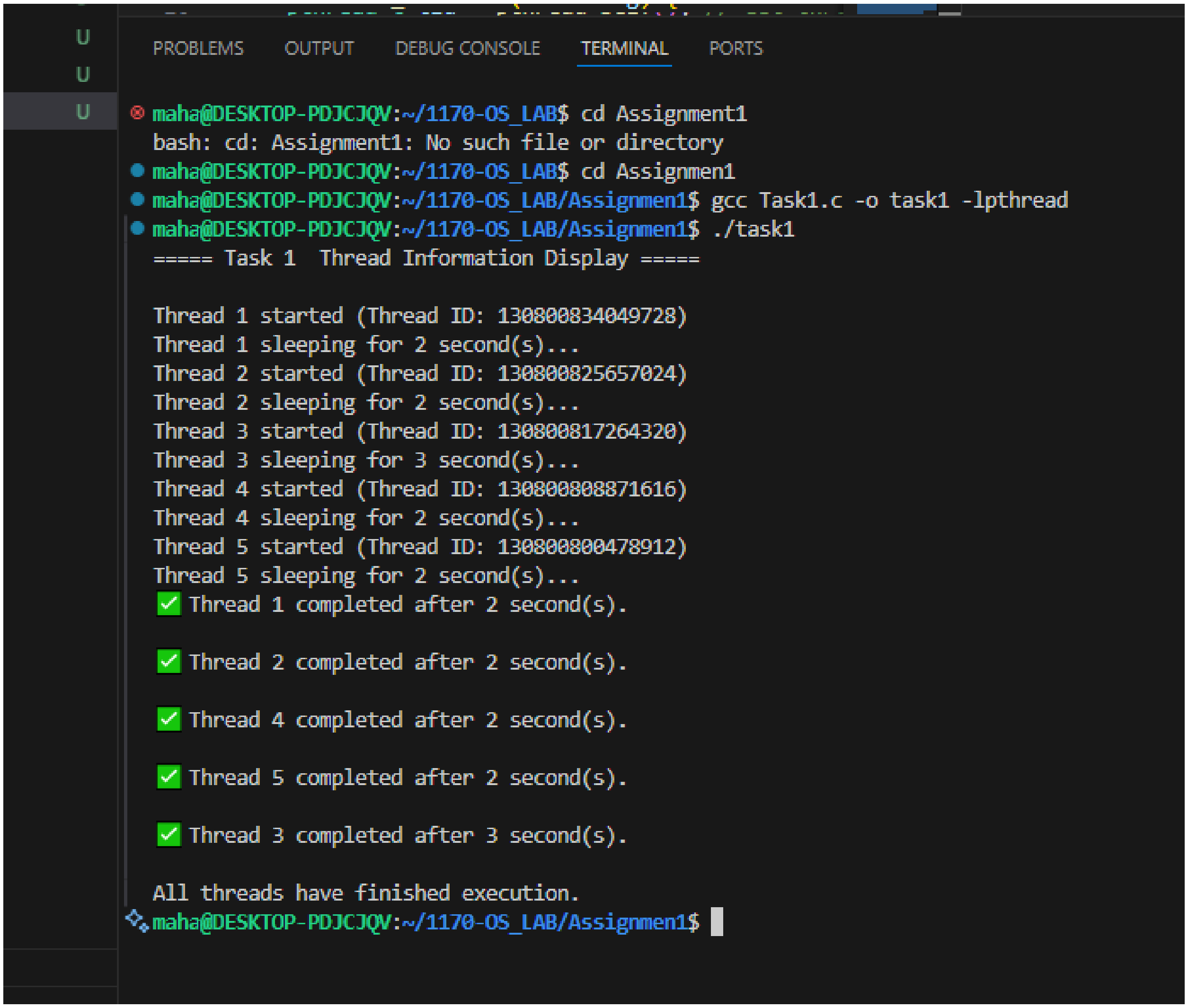
Expected Output: Threads complete in different orders due to random sleep times.

Code:



```c
/*
 ============================================================================
 Name        : Maha Ather
 Registration : 23-NTU-CS-1170
 Task Title  : Task 1 – Thread Information Display
 Description : This program creates 5 threads. Each thread prints:
               • Its thread ID using pthread_self()
               • Its thread number (1st, 2nd, etc.)
               • Sleeps for a random time (1–3 seconds)
               • Prints a completion message before exiting
 ============================================================================
*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <time.h>
void* thread_function(void* arg) {
    int thread_num = *((int*)arg); // Get thread number
    pthread_t tid = pthread_self(); // Get thread ID
    // Generate random sleep time between 1–3 seconds
    int sleep_time = (rand() % 3) + 1;
    printf("Thread %d started (Thread ID: %lu)\n", thread_num, (unsigned long)tid);
    printf("Thread %d sleeping for %d second(s)...\n", thread_num, sleep_time);
    sleep(sleep_time); // Simulate work
    printf("✅ Thread %d completed after %d second(s).\n\n", thread_num, sleep_time);

    pthread_exit(NULL);
}
int main() {
    pthread_t threads[5];
    int thread_numbers[5];
    srand(time(NULL)); // Seed random number generator
    printf("===== Task 1  Thread Information Display =====\n\n");
    // Create 5 threads
    for (int i = 0; i < 5; i++) {
        thread_numbers[i] = i + 1; // Thread numbers start from 1
        if (pthread_create(&threads[i], NULL, thread_function, &thread_numbers[i]) != 0) {
            perror("Failed to create thread");
            exit(1);
        }
    }
    // Wait for all threads to complete
    for (int i = 0; i < 5; i++) {
        pthread_join(threads[i], NULL);
    }
    printf("All threads have finished execution.\n");
    return 0;
}

```

Terminal:



```
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB$ cd Assignment1
bash: cd: Assignment1: No such file or directory
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB$ cd Assignmen1
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/Assignmen1$ gcc Task1.c -o task1 -lpthread
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/Assignmen1$ ./task1
===== Task 1  Thread Information Display =====

Thread 1 started (Thread ID: 130800834049728)
Thread 1 sleeping for 2 second(s)...
Thread 2 started (Thread ID: 130800825657024)
Thread 2 sleeping for 2 second(s)...
Thread 3 started (Thread ID: 130800817264320)
Thread 3 sleeping for 3 second(s)...
Thread 4 started (Thread ID: 130800808871616)
Thread 4 sleeping for 2 second(s)...
Thread 5 started (Thread ID: 130800800478912)
Thread 5 sleeping for 2 second(s)...
✅ Thread 1 completed after 2 second(s).

✅ Thread 2 completed after 2 second(s).

✅ Thread 4 completed after 2 second(s).

✅ Thread 5 completed after 2 second(s).

✅ Thread 3 completed after 3 second(s).

All threads have finished execution.
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/Assignmen1$
```

Task 2 – Personalized Greeting Thread

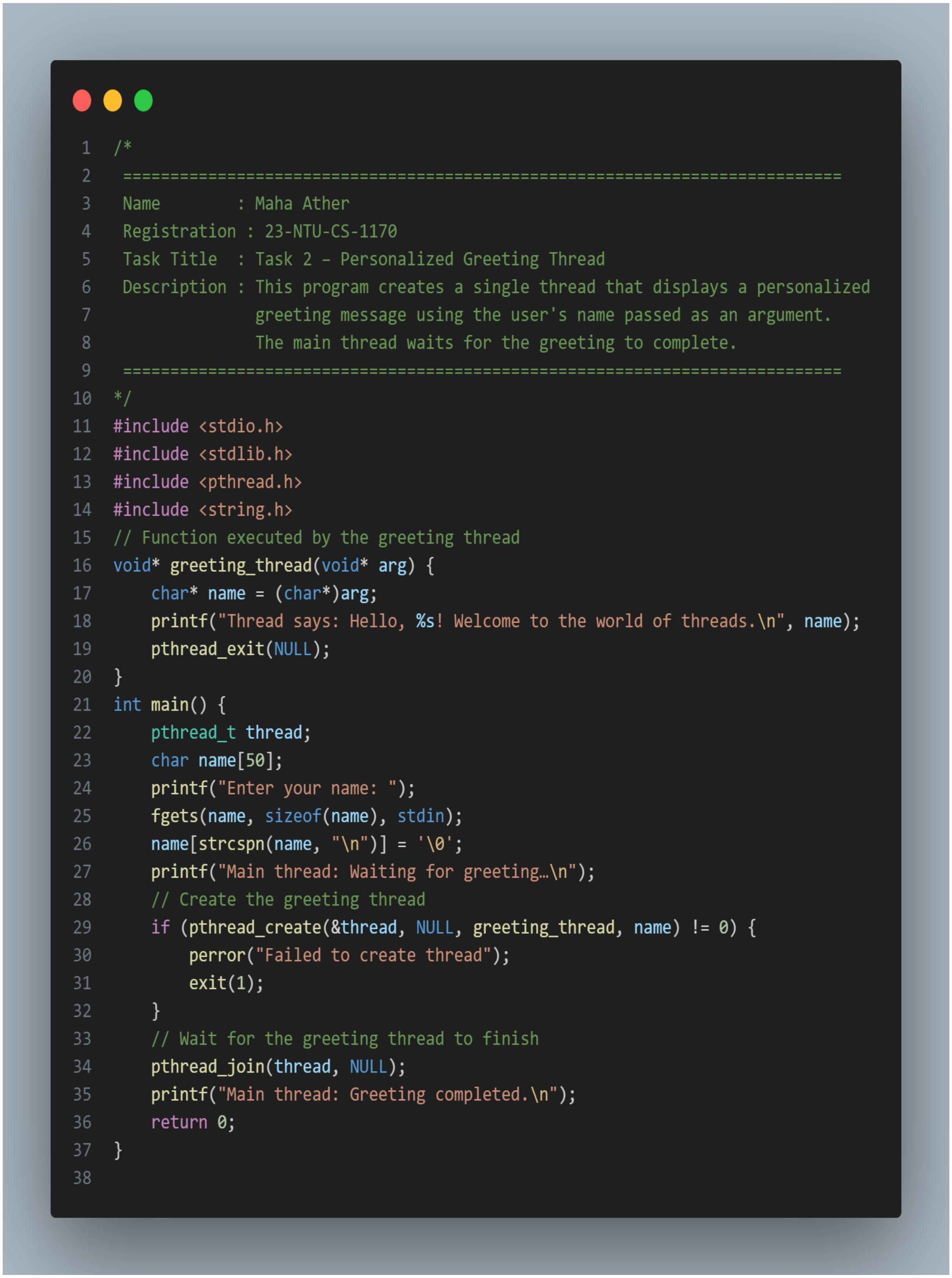
Write a C program that:

- 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:

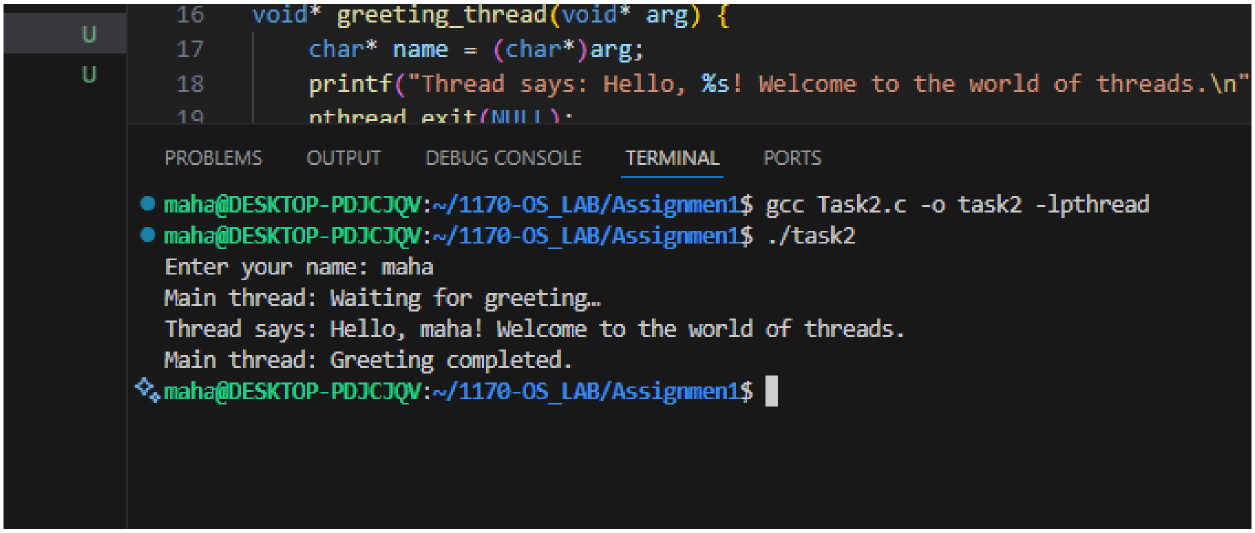
Main thread: Waiting for greeting… Thread says: Hello, Ali! Welcome to the world of threads. Main thread: Greeting completed.

Code:



```c
/*
 ===========================================================================
 Name        : Maha Ather
 Registration : 23-NTU-CS-1170
 Task Title  : Task 2 – Personalized Greeting Thread
 Description : This program creates a single thread that displays a personalized
               greeting message using the user's name passed as an argument.
               The main thread waits for the greeting to complete.
 ===========================================================================
*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
// Function executed by the greeting thread
void* greeting_thread(void* arg) {
    char* name = (char*)arg;
    printf("Thread says: Hello, %s! Welcome to the world of threads.\n", name);
    pthread_exit(NULL);
}
int main() {
    pthread_t thread;
    char name[50];
    printf("Enter your name: ");
    fgets(name, sizeof(name), stdin);
    name[strcspn(name, "\n")] = '\0';
    printf("Main thread: Waiting for greeting…\n");
    // Create the greeting thread
    if (pthread_create(&thread, NULL, greeting_thread, name) != 0) {
        perror("Failed to create thread");
        exit(1);
    }
    // Wait for the greeting thread to finish
    pthread_join(thread, NULL);
    printf("Main thread: Greeting completed.\n");
    return 0;
}

```

Terminal:



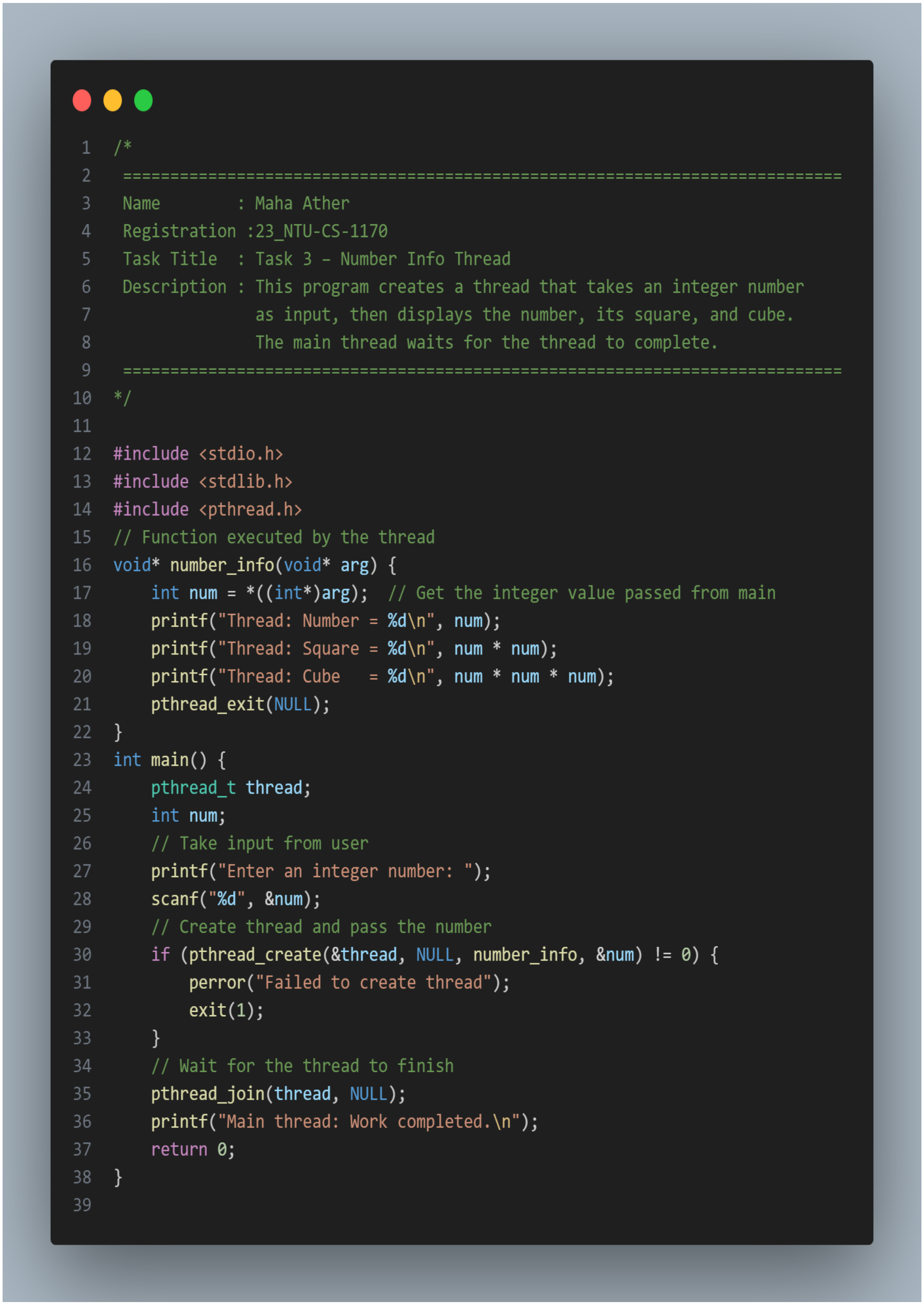
```
void* greeting_thread(void* arg) {
    char* name = (char*)arg;
    printf("Thread says: Hello, %s! Welcome to the world of threads.\n"
    pthread_exit(NULL);
```

```
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/Assignmen1$ gcc Task2.c -o task2 -lpthread
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/Assignmen1$ ./task2
Enter your name: maha
Main thread: Waiting for greeting…
Thread says: Hello, maha! Welcome to the world of threads.
Main thread: Greeting completed.
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/Assignmen1$
```

Task 3 – Number Info Thread Write a program that:

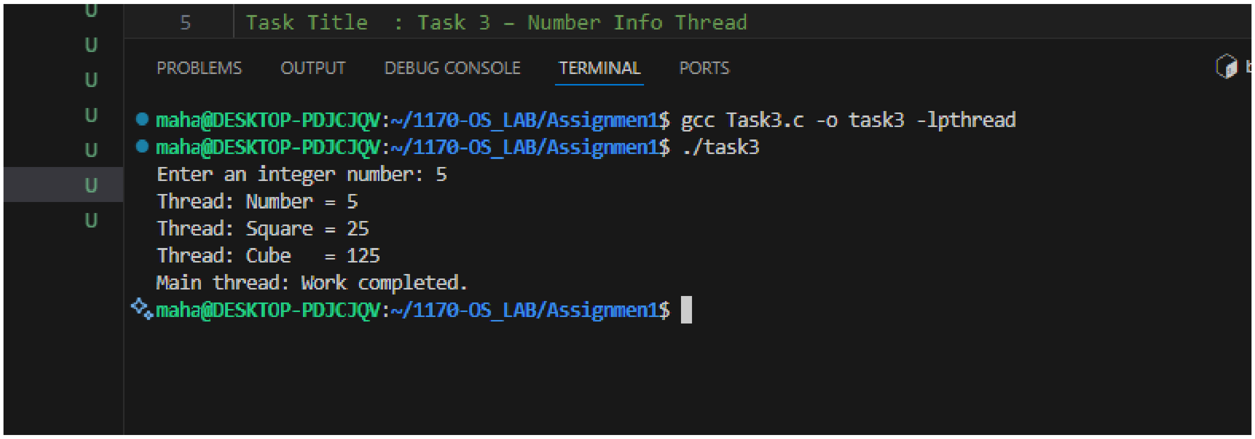
- Takes an integer input from the user.
- Creates a thread and passes this integer to it.
- The thread prints the number, its square, and cube.
- The main thread waits until completion and prints “Main thread: Work completed.”

Code:



```c
/*
 ============================================================================
 Name        : Maha Ather
 Registration :23_NTU-CS-1170
 Task Title  : Task 3 – Number Info Thread
 Description : This program creates a thread that takes an integer number
               as input, then displays the number, its square, and cube.
               The main thread waits for the thread to complete.
 ============================================================================
*/

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
// Function executed by the thread
void* number_info(void* arg) {
    int num = *((int*)arg);  // Get the integer value passed from main
    printf("Thread: Number = %d\n", num);
    printf("Thread: Square = %d\n", num * num);
    printf("Thread: Cube   = %d\n", num * num * num);
    pthread_exit(NULL);
}
int main() {
    pthread_t thread;
    int num;
    // Take input from user
    printf("Enter an integer number: ");
    scanf("%d", &num);
    // Create thread and pass the number
    if (pthread_create(&thread, NULL, number_info, &num) != 0) {
        perror("Failed to create thread");
        exit(1);
    }
    // Wait for the thread to finish
    pthread_join(thread, NULL);
    printf("Main thread: Work completed.\n");
    return 0;
}

```

Terminal:



```
Task Title  : Task 3 – Number Info Thread
```

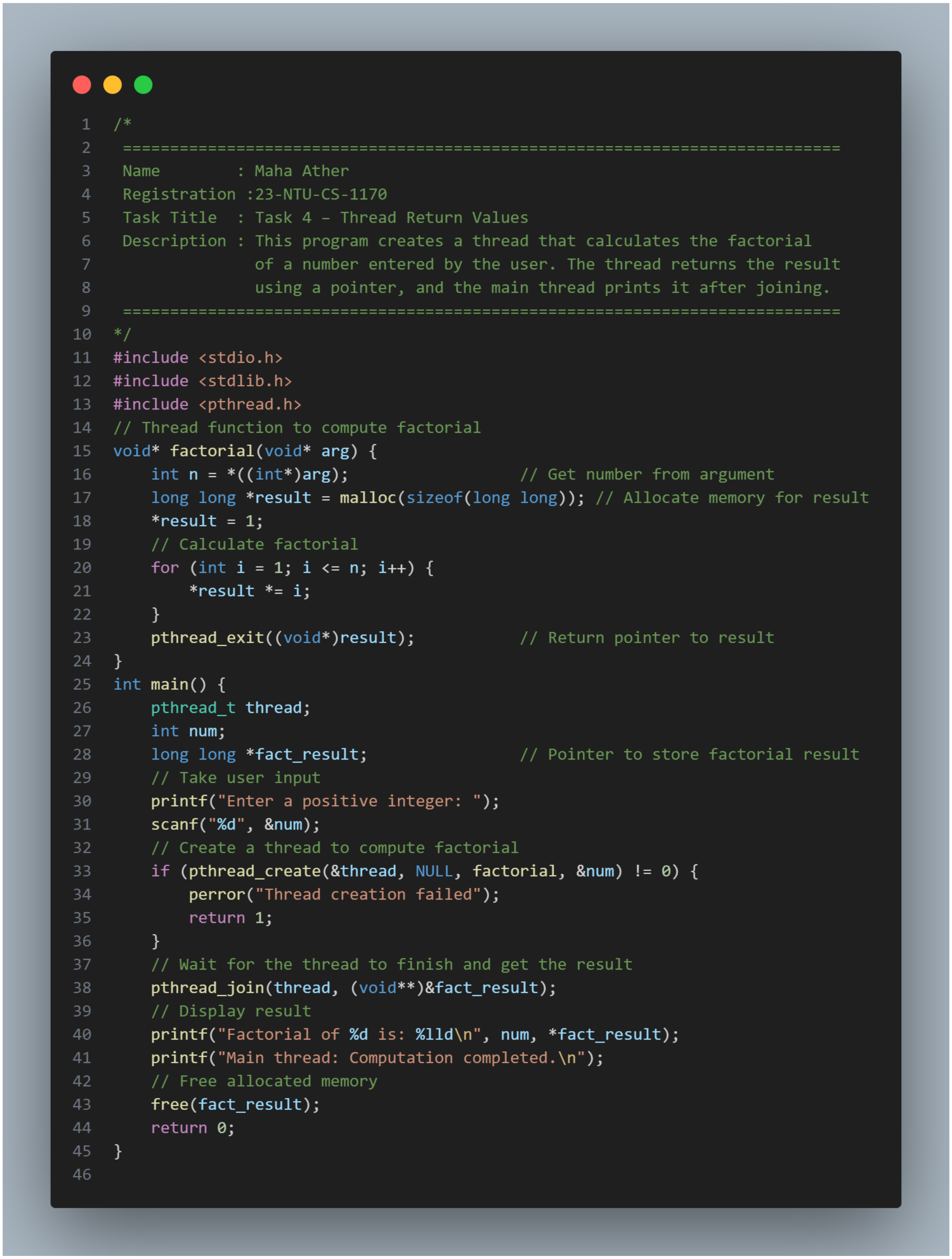
```
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/Assignmen1$ gcc Task3.c -o task3 -lpthread
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/Assignmen1$ ./task3
Enter an integer number: 5
Thread: Number = 5
Thread: Square = 25
Thread: Cube   = 125
Main thread: Work completed.
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/Assignmen1$
```

Task 4 – Thread Return Values

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

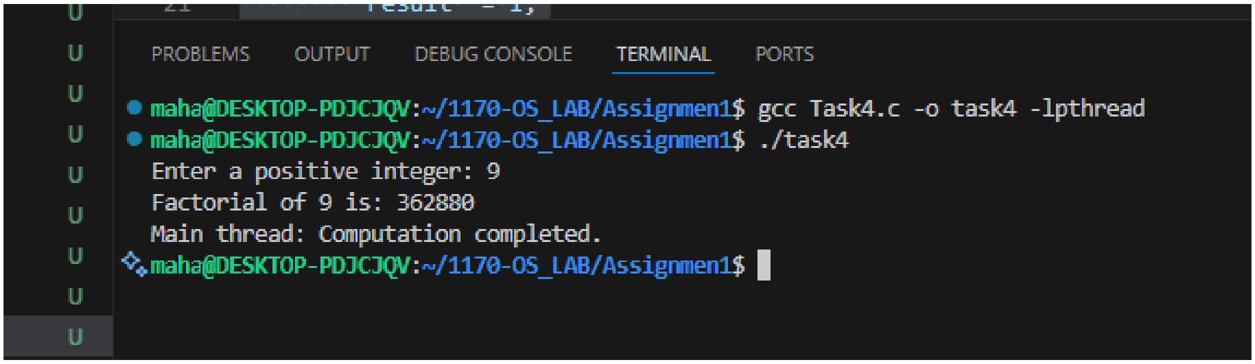
- The thread should return the result using a pointer.
- The main thread prints the result after joining.

Code:



```c
/*
 ============================================================================
 Name        : Maha Ather
 Registration :23-NTU-CS-1170
 Task Title  : Task 4 – Thread Return Values
 Description : This program creates a thread that calculates the factorial
               of a number entered by the user. The thread returns the result
               using a pointer, and the main thread prints it after joining.
 ============================================================================
*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
// Thread function to compute factorial
void* factorial(void* arg) {
    int n = *((int*)arg);                  // Get number from argument
    long long *result = malloc(sizeof(long long)); // Allocate memory for result
    *result = 1;
    // Calculate factorial
    for (int i = 1; i <= n; i++) {
        *result *= i;
    }
    pthread_exit((void*)result);           // Return pointer to result
}
int main() {
    pthread_t thread;
    int num;
    long long *fact_result;                // Pointer to store factorial result
    // Take user input
    printf("Enter a positive integer: ");
    scanf("%d", &num);
    // Create a thread to compute factorial
    if (pthread_create(&thread, NULL, factorial, &num) != 0) {
        perror("Thread creation failed");
        return 1;
    }
    // Wait for the thread to finish and get the result
    pthread_join(thread, (void**)&fact_result);
    // Display result
    printf("Factorial of %d is: %lld\n", num, *fact_result);
    printf("Main thread: Computation completed.\n");
    // Free allocated memory
    free(fact_result);
    return 0;
}
```

Terminal:



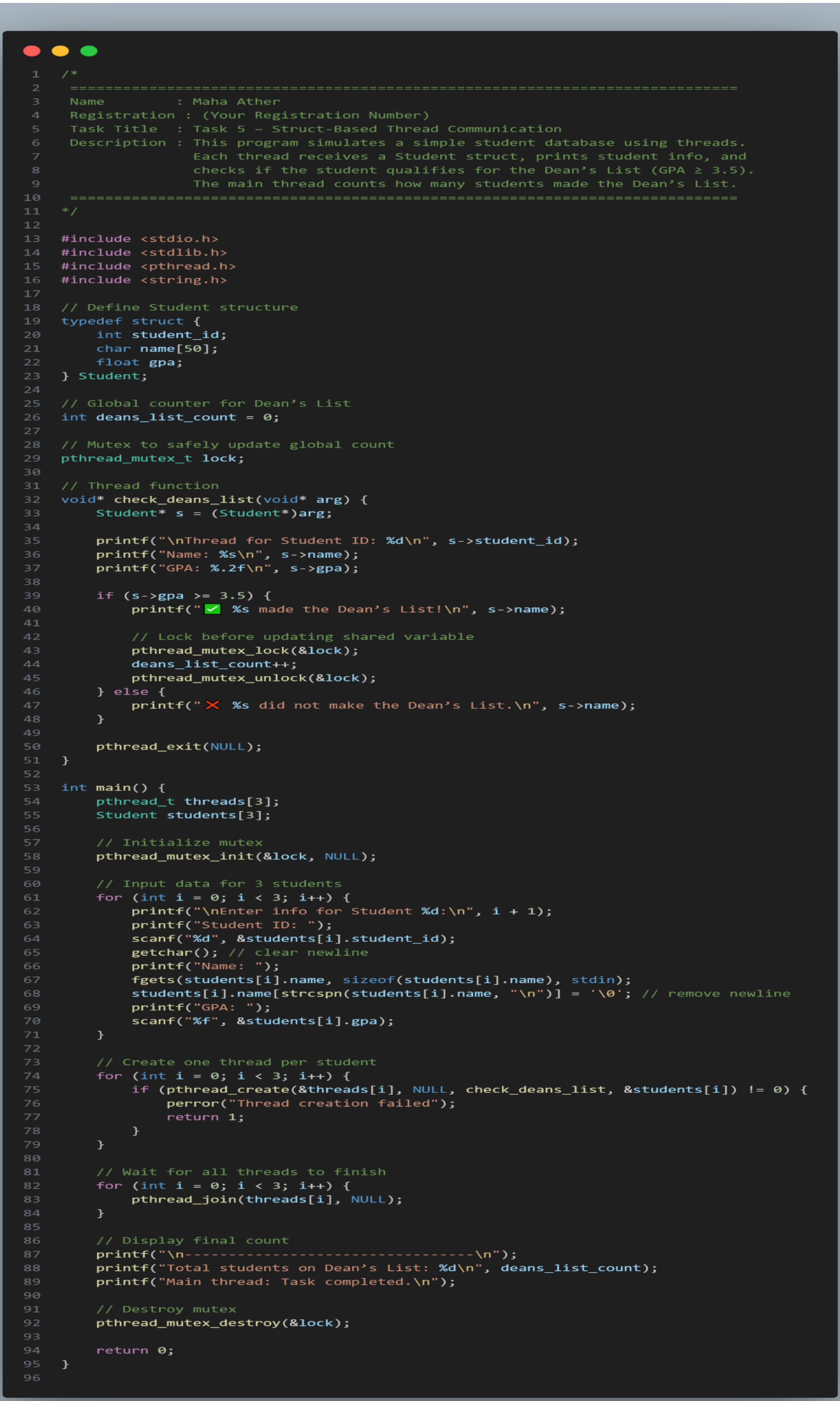
```
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/Assignmen1$ gcc Task4.c -o task4 -lpthread
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/Assignmen1$ ./task4
Enter a positive integer: 9
Factorial of 9 is: 362880
Main thread: Computation completed.
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/Assignmen1$
```

Task 5 – Struct-Based Thread Communication

Create a program that simulates a simple student database system.

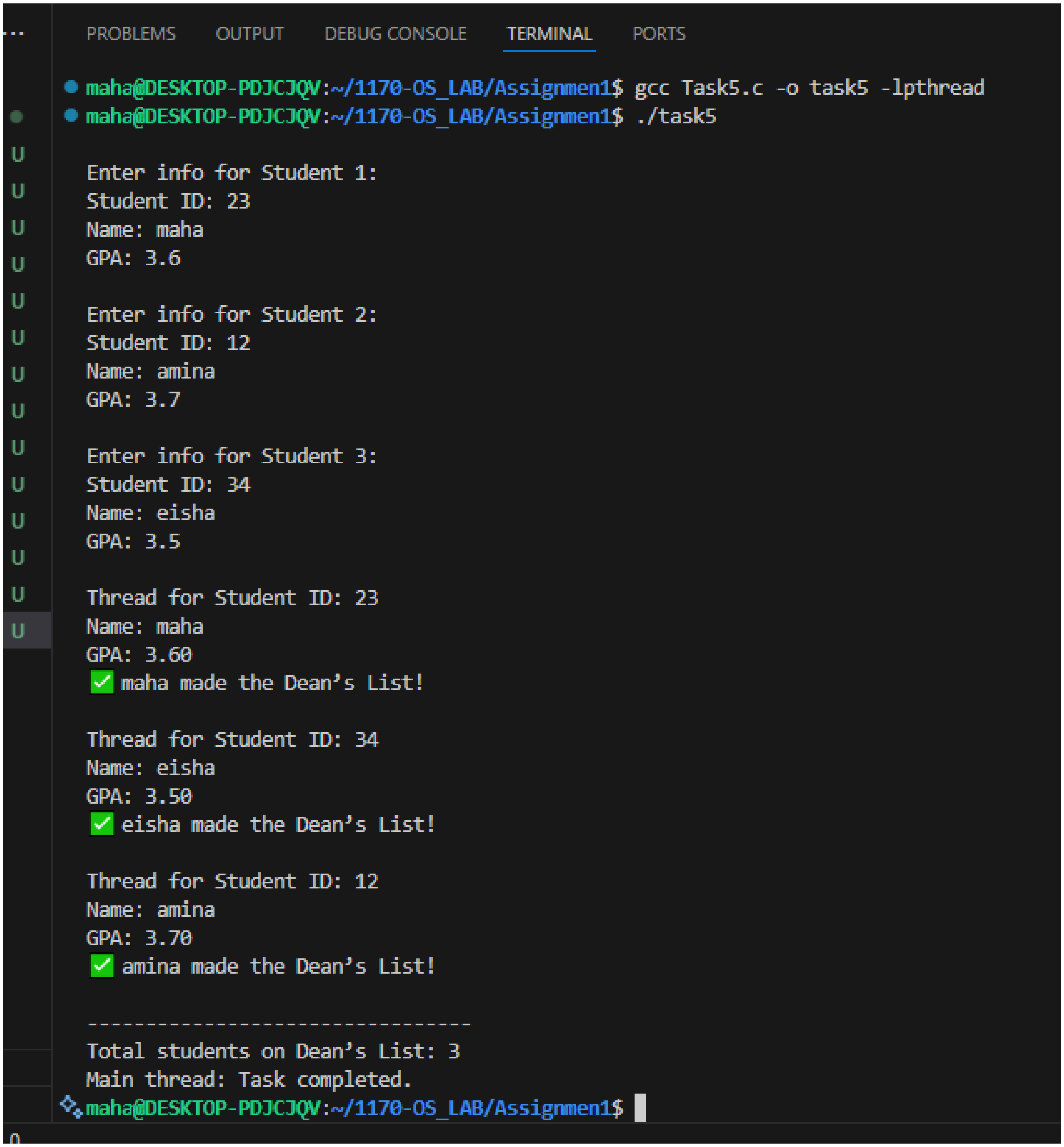
- Define a struct: `typedef struct { int student_id; char name[50]; float gpa; } Student;`
- Create 3 threads, each receiving a different Student struct.
- Each thread prints student info and checks Dean's List eligibility (GPA ≥ 3.5).
- The main thread counts how many students made the Dean's List.

Code:



```c
/*
 ============================================================================
 Name         : Maha Ather
 Registration : (Your Registration Number)
 Task Title   : Task 5 – Struct-Based Thread Communication
 Description : This program simulates a simple student database using threads.
              Each thread receives a Student struct, prints student info, and
              checks if the student qualifies for the Dean’s List (GPA ≥ 3.5).
              The main thread counts how many students made the Dean’s List.
 ============================================================================
*/

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

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

// Global counter for Dean’s List
int deans_list_count = 0;

// Mutex to safely update global count
pthread_mutex_t lock;

// Thread function
void* check_deans_list(void* arg) {
    Student* s = (Student*)arg;

    printf("\nThread for Student ID: %d\n", s->student_id);
    printf("Name: %s\n", s->name);
    printf("GPA: %.2f\n", s->gpa);

    if (s->gpa >= 3.5) {
        printf("✅ %s made the Dean’s List!\n", s->name);

        // Lock before updating shared variable
        pthread_mutex_lock(&lock);
        deans_list_count++;
        pthread_mutex_unlock(&lock);
    } else {
        printf("❌ %s did not make the Dean’s List.\n", s->name);
    }

    pthread_exit(NULL);
}

int main() {
    pthread_t threads[3];
    Student students[3];

    // Initialize mutex
    pthread_mutex_init(&lock, NULL);

    // Input data for 3 students
    for (int i = 0; i < 3; i++) {
        printf("\nEnter info for Student %d:\n", i + 1);
        printf("Student ID: ");
        scanf("%d", &students[i].student_id);
        getchar(); // clear newline
        printf("Name: ");
        fgets(students[i].name, sizeof(students[i].name), stdin);
        students[i].name[strcspn(students[i].name, "\n")] = '\0'; // remove newline
        printf("GPA: ");
        scanf("%f", &students[i].gpa);
    }

    // Create one thread per student
    for (int i = 0; i < 3; i++) {
        if (pthread_create(&threads[i], NULL, check_deans_list, &students[i]) != 0) {
            perror("Thread creation failed");
            return 1;
        }
    }

    // Wait for all threads to finish
    for (int i = 0; i < 3; i++) {
        pthread_join(threads[i], NULL);
    }

    // Display final count
    printf("\n---------------------------------\n");
    printf("Total students on Dean’s List: %d\n", deans_list_count);
    printf("Main thread: Task completed.\n");

    // Destroy mutex
    pthread_mutex_destroy(&lock);

    return 0;
}
```

Terminal:



```
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/Assignmen1$ gcc Task5.c -o task5 -lpthread
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/Assignmen1$ ./task5

Enter info for Student 1:
Student ID: 23
Name: maha
GPA: 3.6

Enter info for Student 2:
Student ID: 12
Name: amina
GPA: 3.7

Enter info for Student 3:
Student ID: 34
Name: eisha
GPA: 3.5

Thread for Student ID: 23
Name: maha
GPA: 3.60
✅ maha made the Dean’s List!

Thread for Student ID: 34
Name: eisha
GPA: 3.50
✅ eisha made the Dean’s List!

Thread for Student ID: 12
Name: amina
GPA: 3.70
✅ amina made the Dean’s List!

---------------------------------
Total students on Dean’s List: 3
Main thread: Task completed.
maha@DESKTOP-PDJCJQV:~/1170-OS_LAB/Assignmen1$
```

Section-B: Short Questions

1. Define an Operating System in a single line.
    - An Operating System (OS) is system software that acts as an interface between the user and computer hardware, managing resources and controlling system operations.

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

- The primary function of the CPU scheduler is to select one process from the ready queue and assign the CPU to it for execution.

3. List any three states of a process.
    - Ready, Running, and Waiting (or Blocked).

4. What is meant by a Process Control Block (PCB)?
    - PCB control block is a data which is used by OS for process control. Each process have PCB which store its status

5. Differentiate between a process and a program.
    - A program is a passive collection of instructions stored on disk, while a process is an active instance of a program in execution.

6. What do you understand by context switching?
    - When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch
    - Context of a process represented in the PCB
    - Context-switch time is pure overhead the system does no useful work while switching
    - The more complex the OS and the PCB ➔ the longer the context switch

7. Define CPU utilization and throughput.
    - CPU Utilization:

      Keep CPU busy as much as possible
    - Throughput:

      Amount of processes executed by CPU per unit time
8. What is the turnaround time of a process?
    - Turnaround time:

      Amount of time used by process to complete its execution
9. How is waiting time calculated in process scheduling?
    - Waiting time is calculated as the total time a process spends in the ready queue waiting for CPU allocation.
10. Define response time in CPU scheduling.
    - Response Time:

      The amount of time a CPU used to create a first response when a request is submitted
11. What is preemptive scheduling?
    - preemptive scheduling

      Preemptive scheduling allows the CPU to be taken away from a running process If a higher-priority process arrives or after a time slice expires.

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

      Non-preemptive scheduling means once a process starts executing, it cannot be stopped until it finishes or voluntarily releases the CPU.

13. State any two advantages of the Round Robin scheduling algorithm.
    - It gives equal opportunity to all processes and ensures better responsiveness in time-sharing systems.
14. Mention one major drawback of the Shortest Job First (SJF) algorithm.
    - SJF can cause starvation for longer processes if shorter ones keep arriving.
15. Define CPU idle time.
    - CPU idle time is the period when the CPU is not executing any process.
16. State two common goals of CPU scheduling algorithms.
    - Maximize CPU utilization
    - minimize waiting or turnaround time.
17. List two possible reasons for process termination.
    - Normal completion of process
    - Arithmetic error
    - Data misuse
    - Time runout
    - I/O failure
18. Explain the purpose of the wait() and exit() system calls.
    - Exit():

      When a child process completes its work then it calls exit() to terminate it self
    - Wait():

      Then calls wait() to return resources to parent process
      If it doesn't call wait then child process becomes zombie and parent doesn't wait for child process completion

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

Shared Memory Model	Message Passing Model
Processes communicate by sharing a common memory space.	Processes communicate by sending and receiving messages.
Faster communication for large data transfers.	Slower for large data but safer and easier to implement.
Requires synchronization (e.g., semaphores) to avoid conflicts.	No need for explicit synchronization; handled by the OS.

Suitable for processes on the same system.	Suitable for processes on different systems or networks.

20. Differentiate between a thread and a process.

Process	Thread
A process is an independent program in execution.	A thread is a smaller unit of a process that can run concurrently.
Has its own memory and resources.	Shares memory and resources with other threads of the same process.
Creation and switching are slower.	Creation and switching are faster.
Failure of one process does not affect others.	Failure of one thread can affect other threads in the same process.

21. Define multithreading.

- Multithreading means running multiple threads (smaller tasks) at the same time within a single process.
- Each thread is a lightweight subprocess — it shares the same memory and resources as the main process.
- It helps a program do multiple things simultaneously, improving efficiency and speed.

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

CPU-Bound Process	I/O-Bound Process
Spends most of its time using the CPU for computations.	Spends most of its time waiting for input/output operations.
Requires high processing power.	Requires frequent I/O operations.
Example: Scientific calculations or data analysis.	Example: File transfer or database read/write.
Less number of I/O operations.	More number of I/O operations.

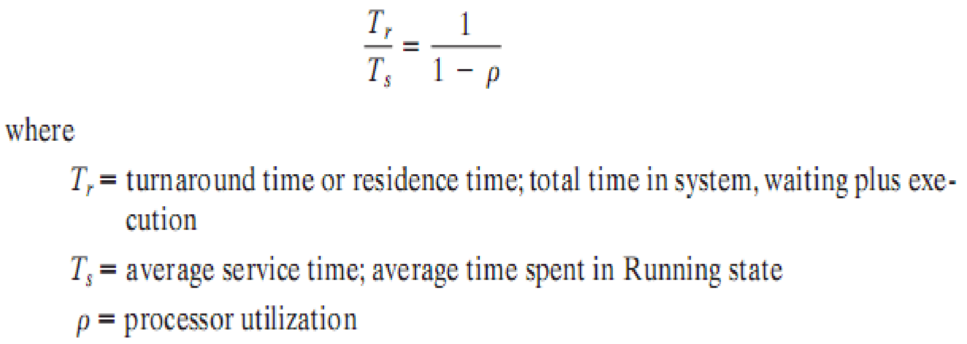
23. What are the main responsibilities of the dispatcher?
    - The dispatcher is responsible for context switching, switching the CPU to user mode, and jumping to the correct program location to start execution
24. Define starvation and aging in process scheduling.
    - Starvation occurs when a process waits indefinitely for CPU allocation. Aging is a technique that increases a process's priority the longer it waits to prevent starvation.
25. What is a time quantum (or time slice)?
    - A time quantum is the fixed amount of CPU time given to each process in Round Robin scheduling.

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

- If too large, response time increases
- if too small, too many context switches occur, reducing CPU efficiency.

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

- The turnaround ratio is the turnaround time divided by the service (or burst) time of a process.



$$\frac{T_r}{T_s} = \frac{1}{1 - \rho}$$

where

T_r = turnaround time or residence time; total time in system, waiting plus execution

T_s = average service time; average time spent in Running state

ρ = processor utilization

28. What is the purpose of a ready queue?

- It stores all the processes that are ready and waiting to use the CPU.
- The CPU scheduler selects processes from the ready queue for execution.
- It helps manage process scheduling in a fair and organized way.
- Processes move to the ready queue after being created or after waiting for I/O.
- It ensures efficient CPU utilization by keeping the CPU busy when possible.

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

CPU Burst	I/O Burst
The time period during which a process is executing instructions on the CPU.	The time period during which a process is performing input/output operations.
Involves computation and processing tasks.	Involves reading from or writing to devices like disk or keyboard.
CPU is actively used.	CPU remains idle or executes another process.
Example: Performing calculations or logic operations.	Example: Reading a file or printing a document.

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

- Round Robin scheduling is starvation-free because every process gets CPU time in a cyclic order

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

- The parent process calls fork() to create a child process
- which then uses exec() to load a new program
- the parent may use wait() to synchronize

32. Define zombie and orphan processes.

- A zombie process has finished execution but still has an entry in the process table. An orphan process continues execution after its parent has terminated.

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

Priority Scheduling	*Shortest Job First (SJF)*
Processes are scheduled based on their assigned priority value.	Processes are scheduled based on the shortest burst time.
A process with higher priority gets the CPU first.	A process with the smallest CPU burst time gets the CPU first.
May lead to starvation of low-priority processes.	May lead to starvation of long processes.
Priority can be assigned internally or externally.	Burst time must be known in advance.

34. Define context switch time and explain why it is considered overhead.

- Context switch time is the time taken by the CPU to save the state of the currently running process and load the state of the next process to be executed.
- It is considered overhead because:
  No useful work is done during a context switch.
  It consumes CPU time and system resources.
  Frequent context switching can reduce overall CPU efficiency.

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

Three Levels of Schedulers in an Operating System:

- **Long-Term Scheduler (Job Scheduler):**
  Decides which processes are admitted into the ready queue from secondary storage. It controls the degree of multiprogramming.
- **Medium-Term Scheduler:**
  Temporarily removes processes from main memory (suspension) and later reintroduces them to improve memory utilization and CPU performance.
- **Short-Term Scheduler (CPU Scheduler):**
  Selects which process in the ready queue should be executed next by the CPU. It runs most frequently.

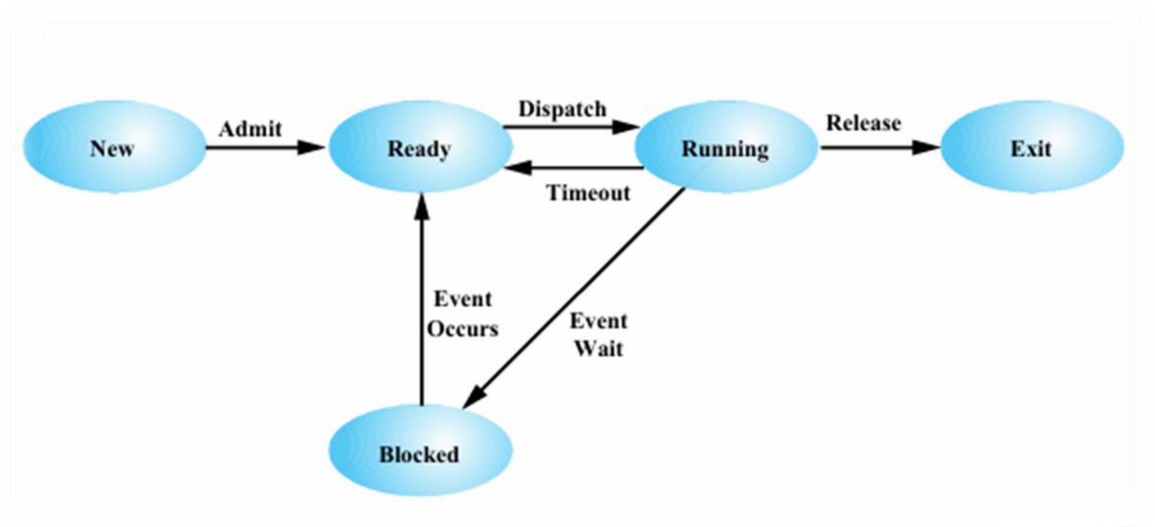
36. Differentiate between User Mode and Kernel Mode in an Operating System.

User Mode	Kernel Mode
In this mode, user applications run with limited access to system resources.	In this mode, the operating system has full access to all hardware and resources.
Direct hardware access is not allowed.	Direct hardware access is allowed.
Errors in user mode affect only the running application.	Errors in kernel mode can crash the entire system.
Used for running user-level programs like MS Word or browsers.	Used for running OS-level functions like process and memory management.

Section-C: Technical / Analytical Questions

Question NO.01:

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



States:
- New: The process is being created.
- Ready: The process is waiting in memory for the CPU.
- Running: The process’s instructions are being executed on the CPU.
- Blocked: The process is waiting for an external event before it can continue.
- Exit: The process has finished its execution.

Transitions:
- Admit: The OS moves the process from New into the Ready queue.
- Dispatch: The OS selects the process from Ready and gives it the CPU.
- Timeout: The OS interrupts the process to give the CPU to another one.
- Event Wait: The process requests a resource that is not immediately available.

- Event Occurs: The event the process was waiting for (e.g., I/O) completes.
- Release: The process finishes its execution and terminates.

Question no.2

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

Context Switch Overhead

A **context switch** occurs when the CPU switches from one process or thread to another. This action is performed by the operating system to enable **multitasking** and efficient use of the CPU. During this switch, the CPU must save the state of the currently running process and load the state of the next process that will run.

However, this switching process does not perform any actual computation for user programs — it only involves saving and loading data. The time and resources consumed in this activity are known as **context switch overhead**.

This overhead reduces CPU efficiency, as valuable processing time is spent managing processes instead of executing them. The frequency of context switches depends on the **scheduling algorithm** and the **time quantum** used by the system.

Reasons for Context Switching:

- A process voluntarily releases the CPU (for example, when waiting for I/O).
- The running process exceeds its time quantum in preemptive scheduling.
- A higher-priority process becomes ready to run.
- The running process terminates or is blocked.

Why It Is Considered Overhead:

- It consumes CPU time but does not contribute to any actual process execution.
- It involves multiple memory and register operations.
- Frequent context switches can lead to slower performance and increased latency.
- It causes cache invalidation, reducing CPU cache efficiency.

Information Saved and Restored During a Context Switch:

- **Program Counter (PC):** Stores the address of the next instruction to execute.
- **CPU Registers:** Hold the temporary data and instruction states.
- **Process State:** Indicates whether the process is ready, running, or waiting.
- **Stack Pointer:** Keeps track of function calls and local variables.
- **Memory Management Information:** Includes page tables or base/limit registers to restore memory context.

- **I/O Status Information:** Tracks ongoing input/output operations and open files.
- **Accounting Information:** Includes CPU usage statistics and process priority.

Minimizing Context Switch Overhead:

- Using efficient CPU scheduling algorithms (e.g., SJF, priority scheduling).
- Avoiding unnecessary preemption.
- Increasing time quantum in round-robin systems (but not too much).
- Grouping related tasks to reduce frequent switching.

Question NO.03:

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

Components of a Process Control Block (PCB)

A **Process Control Block (PCB)** is a data structure maintained by the operating system that contains all the essential information about a specific process. It acts as the identity card of a process and helps the OS manage and control processes efficiently.

When a process is created, a PCB is also created for it. When the process is terminated, its PCB is deleted.

Main Components of a PCB:

- **Process ID (PID):**
A unique identifier assigned to each process. It helps the operating system distinguish between multiple running processes.

- **Process State:**
Indicates the current state of the process such as New, Ready, Running, Waiting, or Terminated.

- **Program Counter (PC):**
Holds the address of the next instruction that the CPU should execute for this process.

- **CPU Registers:**
Store the contents of all CPU registers so that the process can resume correctly after being paused or switched.

- **Memory Management Information:**
Contains details like base and limit registers, page tables, or segment tables that define the memory area allocated to the process.

- **Accounting Information:**
Keeps track of CPU usage, process priority, process ID of the parent, and time limits. This helps the OS monitor resource usage and scheduling.

- **I/O Status Information:**
Includes a list of I/O devices allocated to the process, open file tables, and information about pending I/O requests.

- **Process Scheduling Information:**
Contains data used by the CPU scheduler, such as process priority, scheduling queues, and pointers to other PCBs in the queue.

Question NO.04:

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

Aspect	*Long-Term Scheduler (Job Scheduler)*	*Short-Term Scheduler (CPU Scheduler)*	*Medium-Term Scheduler (Swapper)*
Function	Selects processes from secondary storage (job pool) and loads them into memory.	Selects one of the ready processes for CPU execution.	Temporarily removes and later reintroduces processes to control load.
Type / Name	Job Scheduler	CPU Scheduler	Process Swapping Scheduler
Speed	Slowest of all schedulers.	Fastest; runs most frequently.	Intermediate speed between long and short-term schedulers.
Multiprogramming Control	Controls the degree of multiprogramming.	Gives less control over the degree of multiprogramming.	Reduces the degree of multiprogramming when memory is overloaded.
Presence in Systems	Barely present or nonexistent in time-sharing systems.	Always present; key component of time-sharing systems.	Used mainly in time-sharing systems for swapping.
Key Operation	Admits new jobs into memory.	Dispatches ready processes to CPU.	Suspends and later resumes processes.
Re-entry Capability	Can re-enter processes into memory for	Executes ready processes.	Re-introduces suspended processes

continuation of execution.		into memory for continuation.

Question NO.05:

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

CPU Scheduling Criteria and Their Optimization Goals

CPU scheduling determines which process should be executed by the CPU at a given time. To evaluate and compare scheduling algorithms, several criteria are used.

1. CPU Utilization

- Definition: It measures how busy the CPU is, i.e., the percentage of time the CPU is executing processes rather than being idle.
- Goal: Maximize CPU utilization so the processor remains active as much as possible.
- Example: If CPU utilization is 90%, the CPU is active 90% of the time and idle 10%.

2. Throughput

- Definition: The number of processes completed within a specific period of time.
- Goal: Maximize throughput by completing more processes per unit time.
- Example: If 10 processes finish in one second, the throughput is 10 processes per second.

3. Turnaround Time

- Definition: The total time taken from the submission of a process to its completion.
  Formula: Turnaround Time = Completion Time – Arrival Time
- Goal: Minimize turnaround time so that processes finish faster.
- Example: If a process arrives at 2s and completes at 12s, the turnaround time is 10s.

4. Waiting Time

- Definition: The total time a process spends waiting in the ready queue before execution.
  Formula: Waiting Time = Turnaround Time – Burst Time
- Goal: Minimize waiting time to reduce delay in process execution.
- Example: If a process waits 5 seconds before execution, its waiting time is 5s.

5. Response Time

- Definition: The time between the submission of a request and the first response produced by the system.
- Goal: Minimize response time to improve the performance of interactive systems.
- Example: In an online application, faster response time means the system reacts quickly to user input.

Summary of Optimization Goals

CPU Utilization – Maximize
Throughput – Maximize
Turnaround Time – Minimize
Waiting Time – Minimize
Response Time – Minimize

Section-D: CPU Scheduling Calculations

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

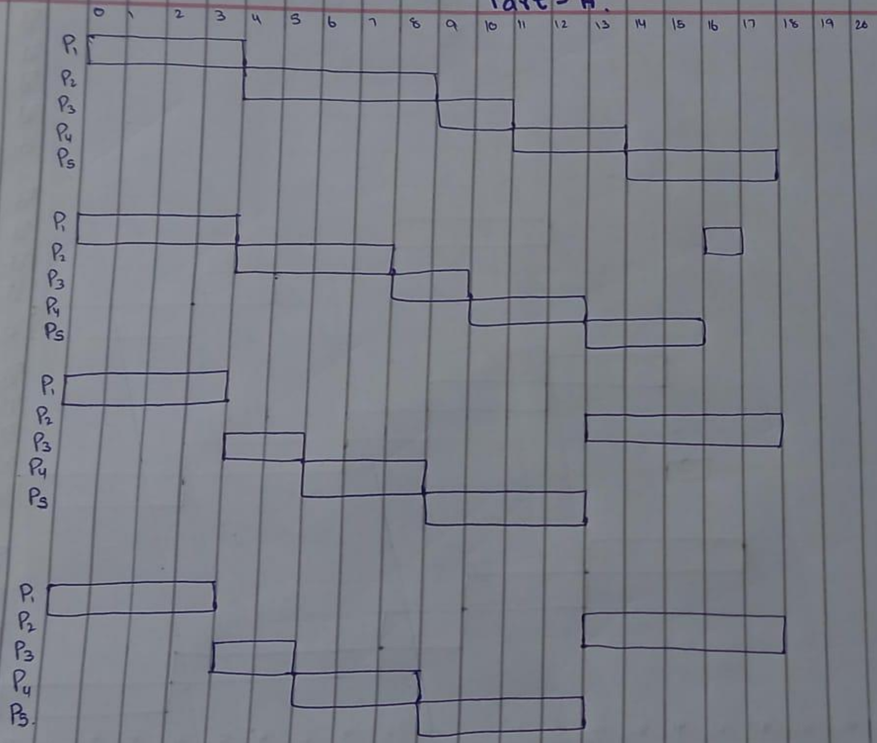
- Draw Gantt charts for FCFS, RR (Q=4), SJF, and SRTF.
- Compute Waiting Time, Turnaround Time, TR/TS ratio, and CPU Idle Time.
- Compare average values and identify which algorithm performs best.

Part (A):

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

23-NTU-CS-1170

Part - A.



FCFS:

Process	Finish	Turnarround	Service	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.

- Average Waiting Time : (0+2+5+5+5)/5 = 3.4
- Average Turn around: (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):

Process	Finish	Turnaround	Service	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	2.37
P_5	17	8	4	4	2.0

- Avg waiting Time: (0+11+4+4+4)|5 = 4.6
- Avg Turnaround : (4+16+6+7+8)|5 = 8.2
- Avg Tr/Ts Ratio: (1+3.2+3.0+2.37+2.0)/5 = 2.31
- CPU Idle Time: 0

SJF Algorithm:

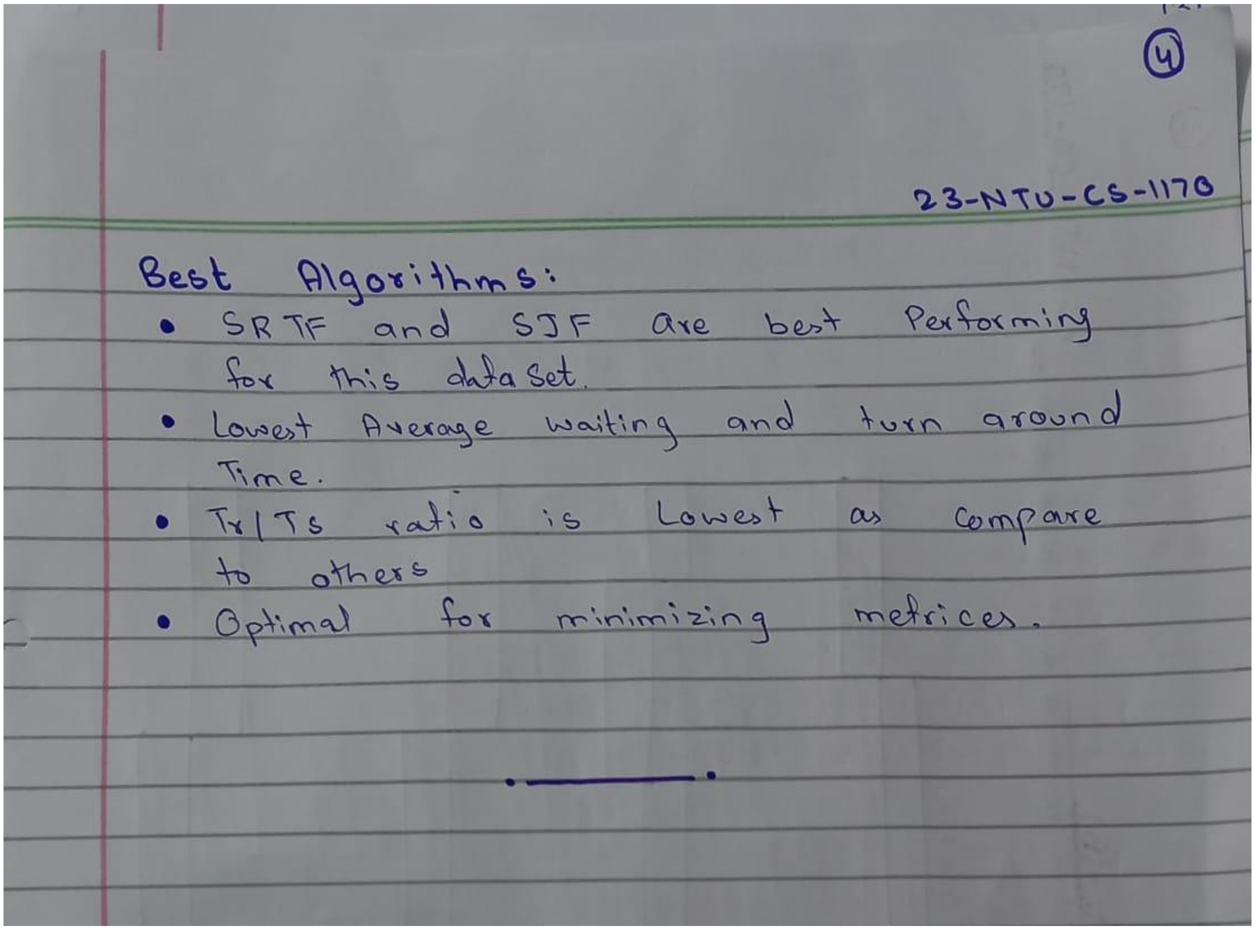
Process	Finish	Tr	Ts	Waiting	Tr/Ts
P_1	4	4	4	0	1.0
P_2	18	16	5	11	3.2
P_3	6	2	2	0	1.0
P_4	9	3	3	0	1.0
P_5	13	4	4	0	1.0

- Avg waiting Time: (0+11+0+0+0) / 5 = 2.2
- Avg Turnaround: (4+16+2+3+4) / 5 = 5.8
- Avg Tr/Ts ratio: (1+3.2+1+1+1) / 5 = 1.44
- CPU Idle Time: 0

SRTF Analysis:

Process	Finish	Tr	Ts	waiting	Tr/Ts
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 Tr/Ts ratio: (1+3.2+1+1+1) / 5 = 1.44
- CPU Idle Time: 0



4

23-NTU-CS-1170

Best Algorithms:

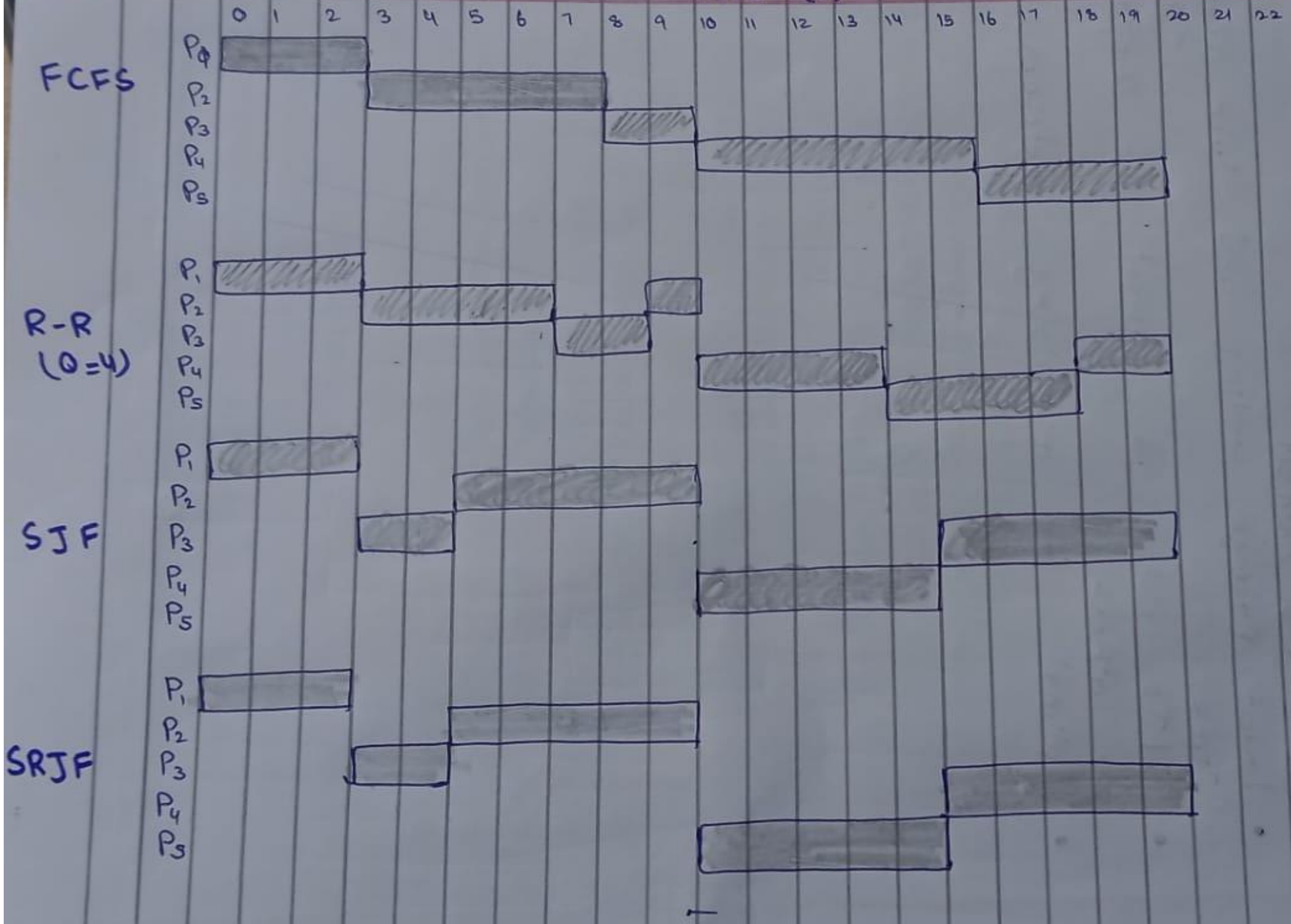
- SRTF and SJF are best Performing for this data set.
- Lowest Average waiting and turn around Time.
- Tr/Ts ratio is Lowest as compare to others
- Optimal for minimizing metrices.

Part (b):

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

Part - (B)

23-NTU-CS-1170



FCFS Analysis:

Process	Finish	Tr	Ts	waiting	Tr/Ts
P_1	3	3	3	0	1.0
P_2	8	7	5	2	1.4
P_3	10	7	2	5	3.5
P_4	16	7	6	1	1.17
P_5	20	10	4	6	2.5.

- Avg waiting Time: (0+2+5+1+6)/5 = 2.8.
- Avg Turnaround: (3+7+7+7+10)/5 = 6.8
- Avg Tr/Ts: (1+1.4+3.5+1.17+2.5)/5 = 1.91
- CPU utilization: 0

Round Robin (Q=4)

Process	Finish	Tr	Ts	Waiting	Tr/Ts
P_1	3	3	3	0	1.0
P_2	10	9	5	4	1.8
P_3	9	6	2	4	3.0
P_4	20	11	6	5	1.83
P_5	18	8	4	4	2.0

- Avg waiting Time: (0+4+4+5+4)/5 = 3.4.
- Avg Turnaround: (3+9+6+11+8)/5 = 7.4
- Avg Tr/Ts: (1+1.8+3.0+1.83+2.0)/5 = 1.93
- CPU utilization: 0.

SJF Analysis:

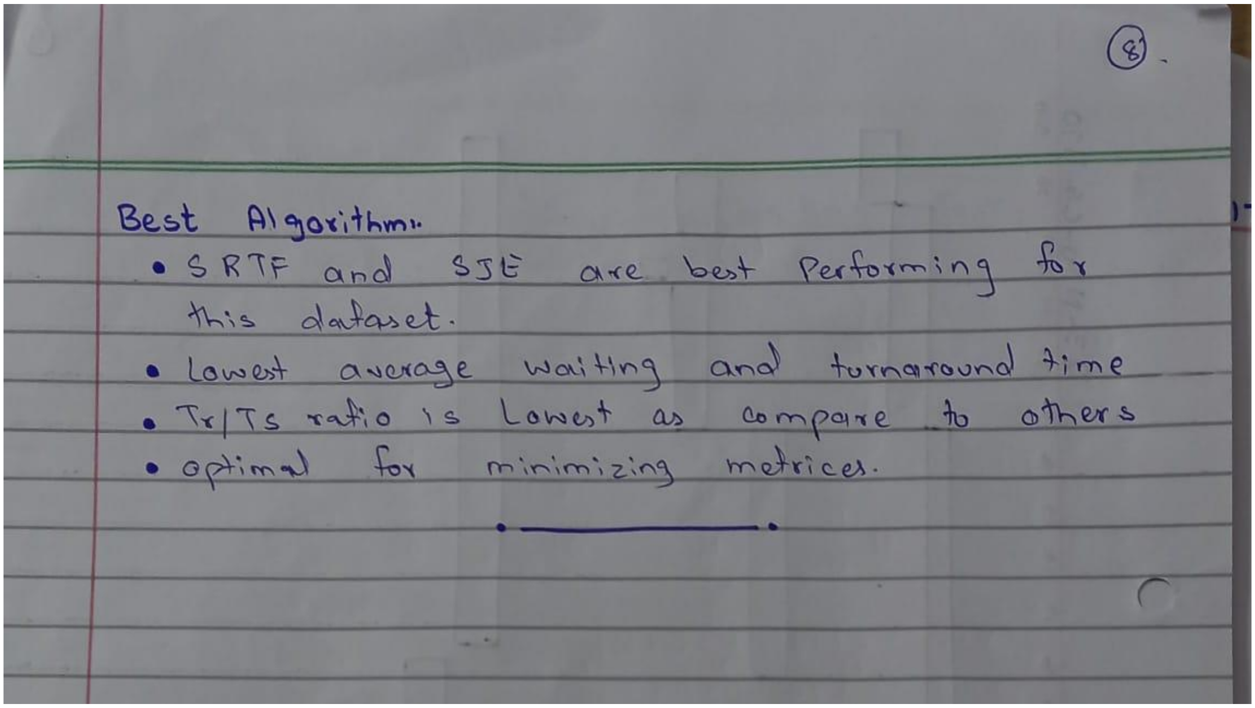
Process	Finish	Tr	Ts	waiting	Tr/Ts
P_1	3	3	3	0	1.0
P_2	10	9	5	4	1.8
P_3	5	2	2	0	1.0
P_4	20	11	6	5	1.83
P_5	14	14	4	0	1.0

- Avg waiting-Time: (0+4+0+5+0)/5 = 1.8.
- Avg Turn around: (3+9+2+11+14)/5 = 5.8.
- Avg Tr/Ts Ratio: (1+1.8+1+1.83+1)/5 = 1.33
- CPU idle Time: 0.

SRTF Analysis:

Process	Finish	Tr	Ts	waiting	Tr/Ts
P_1	3	3	3	0	1.0
P_2	10	9	5	4	1.8
P_3	5	2	2	0	1.0
P_4	20	11	6	5	1.83
P_5	14	4	4	0	1.0

- Avg waiting-Time: (0+4+0+5+0)/5 = 1.8
- Avg Turn around: (3+9+2+11+4)/5 = 5.8
- Avg Tr/Ts ratio: (1.0+1.8+1.0+1.83+1.0)/5 = 1.33
- CPU idle Time: 0.



8

Best Algorithm:-

- SRTF and SJF are best Performing for this dataset.
- Lowest average waiting and turnaround time
- Tr/Ts ratio is Lowest as compare to others
- optimal for minimizing metrices.

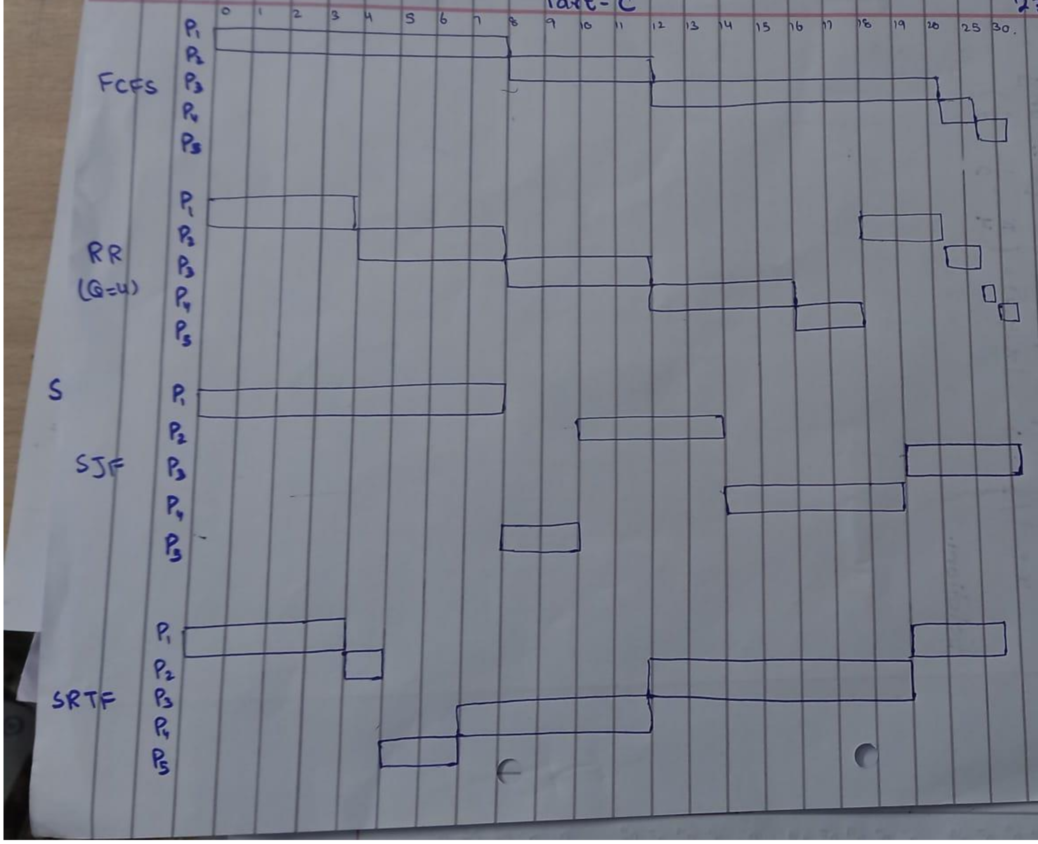
Part (c)

Process	Arrival Time	Service Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5
P5	4	2

⑨

Part-C

23-NTU-CS-1170



FCFS Analysis:

Process	Finish	Tr	Ts	waiting	Tr/Ts
P_1	8	8	8	0	1.0
P_2	12	11	4	7	2.75
P_3	21	19	9	10	2.11
P_4	26	23	5	18	4.60
P_5	28	24	2	22	12.0

- Average waiting Time: 11.4
- Average Tr: 17.
- Average Tr/Ts: 4.49
- CPU utilization idle: 0

Round Robin (Q=4)

Process	Finish	Tr	Ts	waiting	Tr/Ts
P_1	22	22	8	14	2.75
P_2	8	7	4	3	1.75
P_3	28	26	9	17	2.89
P_4	27	24	5	19	4.8
P_5	18	14	2	12	7.0

- Average Waiting Time= 13.0
- Average Tr: 18.6.
- Average Tr/Ts: 3.84
- CPU idle Time: 0

SJF Analysis

Proces	Finish	Tr	Ts	Waiting	Tr|Ts
P1	8	8	8	0	1.0
P2	14	13	4	9	3.25
P3	28	26	9	17	2.89
P4	19	16	5	11	3.2
P5	10	6	2	4	3.0

- Average Waiting= 8.2
- Average Tr = 13.8
- Average Tr|Ts= 2.67
- CPU idle Time= 0

SRJF Analysis

Proces	Finish	Tr	Ts	waiting	Tr|Ts.
P1	19	19	8	11	2.38
P2	5	4	4	0	1.0
P3	28	26	9	17	2.89
P4	12	9	5	4	1.8
P5	7	3	2	1	1.5.

- Average waiting= 6.6
- Average Tr = 12.2
- Average Tr|Ts= 1.91
- CPU idle Time= 0.

Best Performing Algorithm:-

- SRTF is best Performing algo for this data set.
- Least Average Turnaround and waiting Time.
- Optimize Performance.