National Textile University

Department of Computer Science

Subject

Operating System

Submitted to:

Sir Nasir Mehmood

Submitted by:

Hussain Naweed

Registration Number

23-NTU-CS-1165

Assignment No.

01

Semester

5th

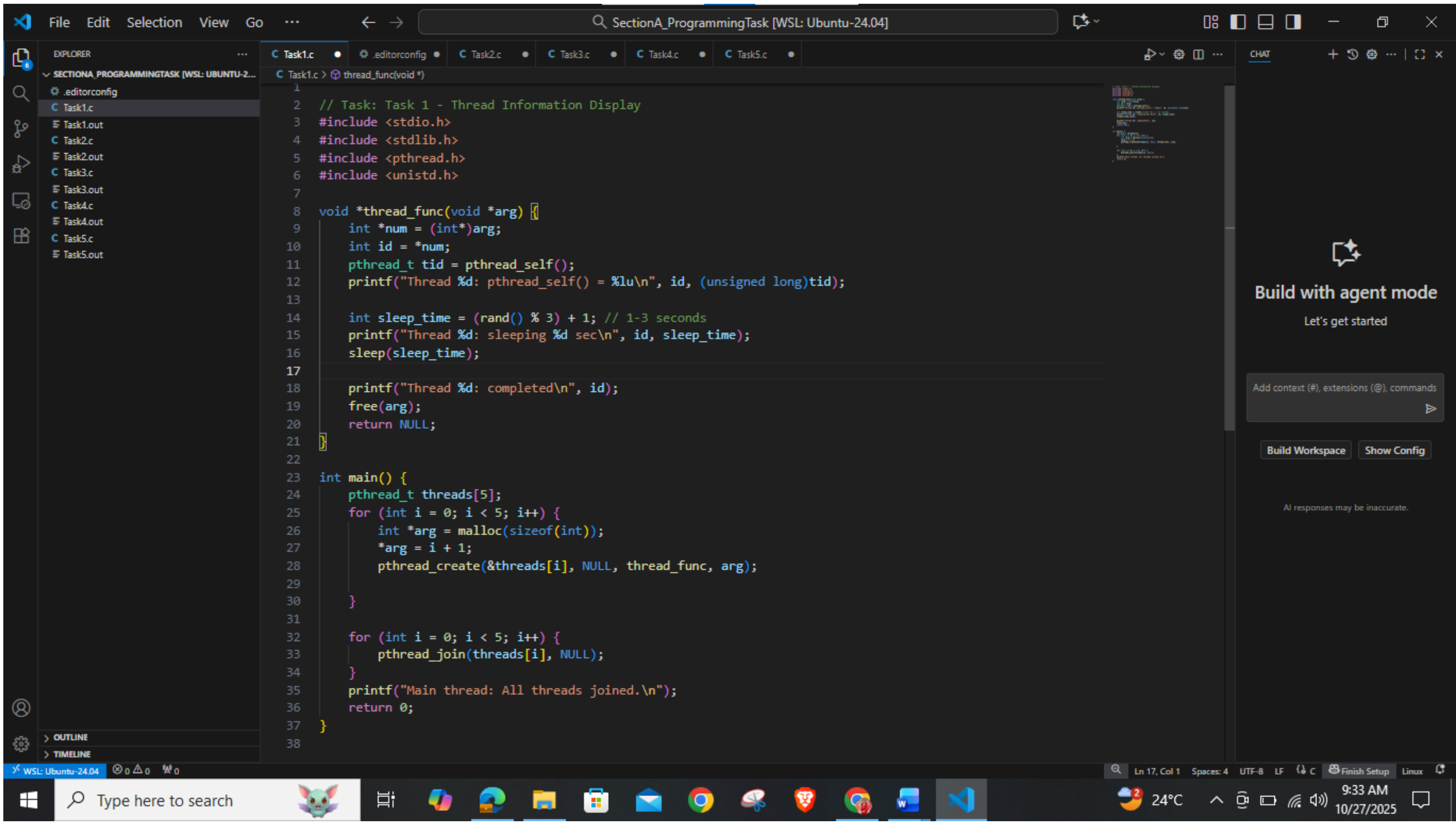
Section-A: Programming Tasks

Task 1 – Thread Information Display:

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

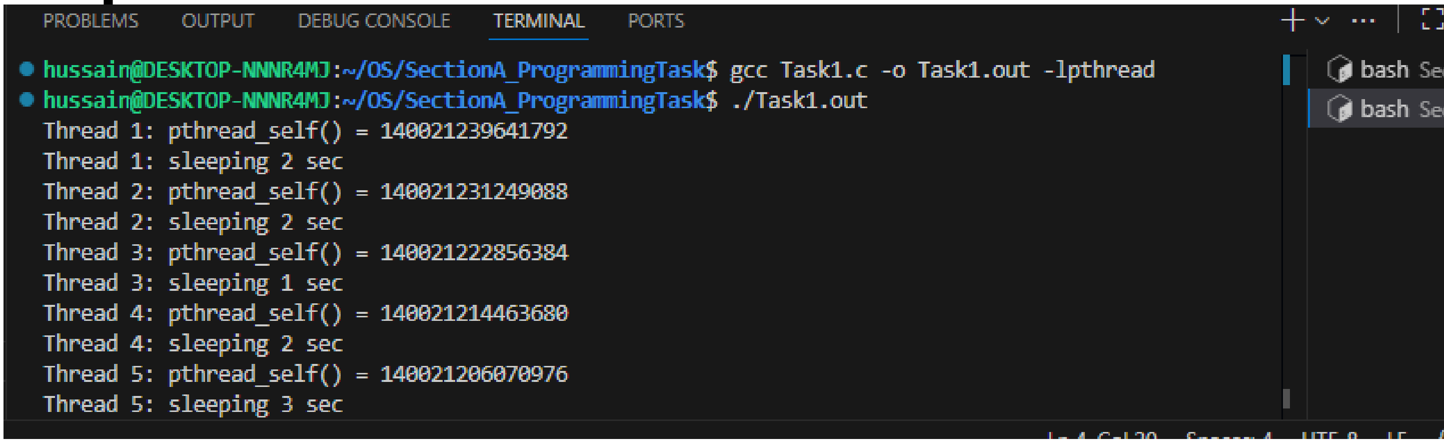
- Print its thread ID using `pthread_self()`.
- Display its thread number (1st, 2nd, etc.).
- Sleep for a random time between 1-3 seconds.
- Print a completion message before exiting.

Code:



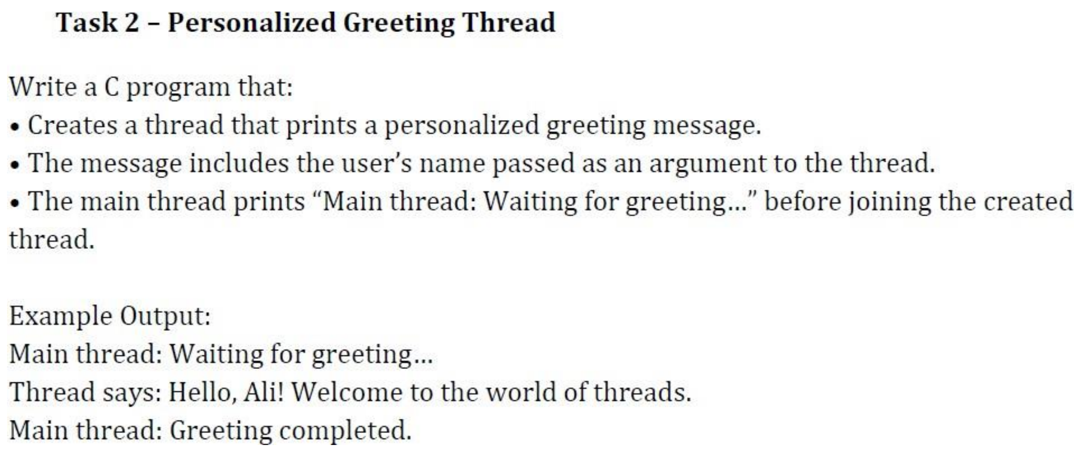
```c

// Task: Task 1 - Thread Information Display
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

void *thread_func(void *arg) {
    int *num = (int*)arg;
    int id = *num;
    pthread_t tid = pthread_self();
    printf("Thread %d: pthread_self() = %lu\n", id, (unsigned long)tid);

    int sleep_time = (rand() % 3) + 1; // 1-3 seconds
    printf("Thread %d: sleeping %d sec\n", id, sleep_time);
    sleep(sleep_time);

    printf("Thread %d: completed\n", id);
    free(arg);
    return NULL;
}

int main() {
    pthread_t threads[5];
    for (int i = 0; i < 5; i++) {
        int *arg = malloc(sizeof(int));
        *arg = i + 1;
        pthread_create(&threads[i], NULL, thread_func, arg);

    }

    for (int i = 0; i < 5; i++) {
        pthread_join(threads[i], NULL);
    }
    printf("Main thread: All threads joined.\n");
    return 0;
}

```

Output:



```
● hussain@DESKTOP-NNNR4MJ:~/OS/SectionA_ProgrammingTask$ gcc Task1.c -o Task1.out -lpthread
● hussain@DESKTOP-NNNR4MJ:~/OS/SectionA_ProgrammingTask$ ./Task1.out
  Thread 1: pthread_self() = 140021239641792
  Thread 1: sleeping 2 sec
  Thread 2: pthread_self() = 140021231249088
  Thread 2: sleeping 2 sec
  Thread 3: pthread_self() = 140021222856384
  Thread 3: sleeping 1 sec
  Thread 4: pthread_self() = 140021214463680
  Thread 4: sleeping 2 sec
  Thread 5: pthread_self() = 140021206070976
  Thread 5: sleeping 3 sec
```

Task_2 Personalize Greeting Thread:



Task 2 – Personalized Greeting Thread

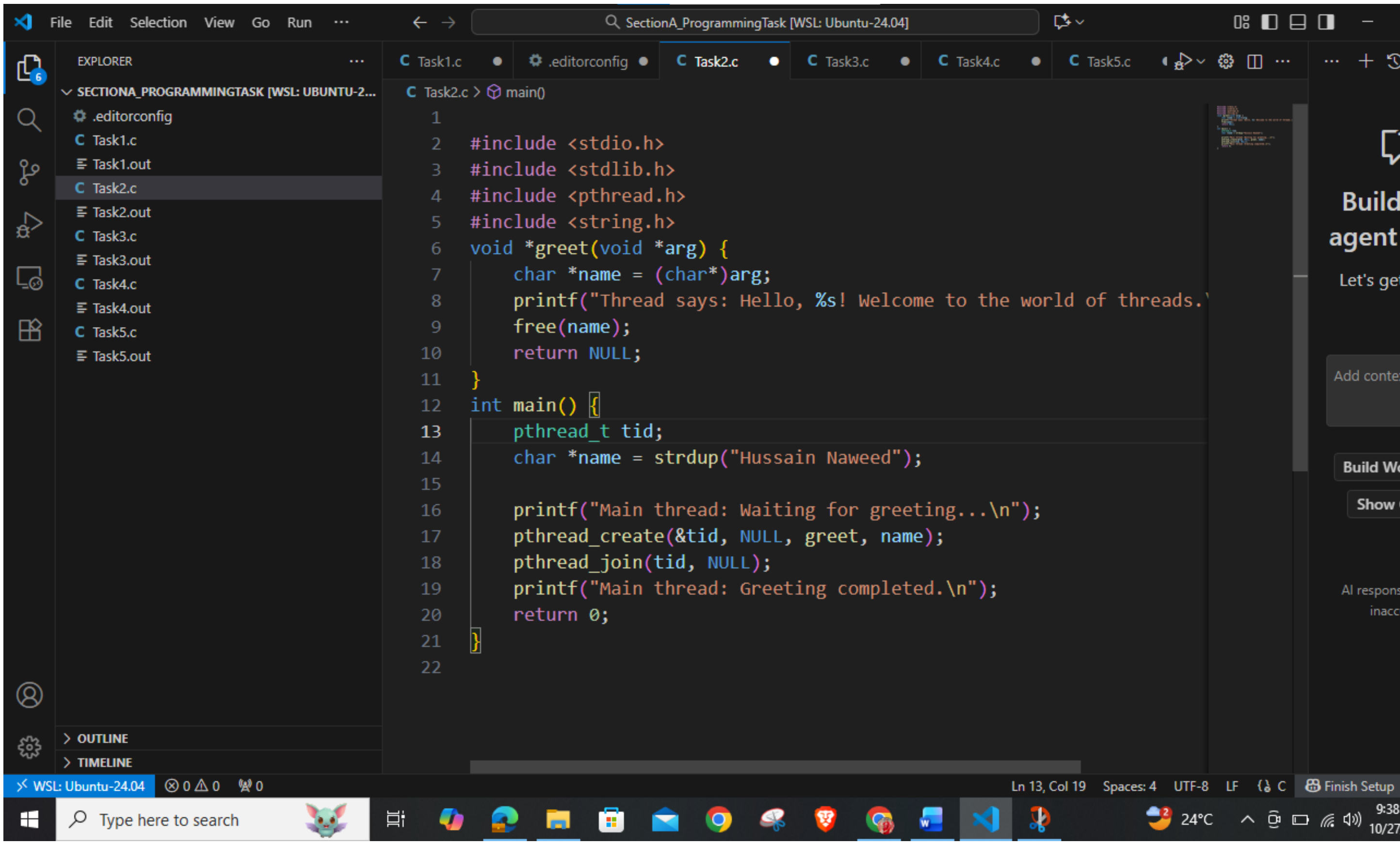
Write a C program that:

- 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:

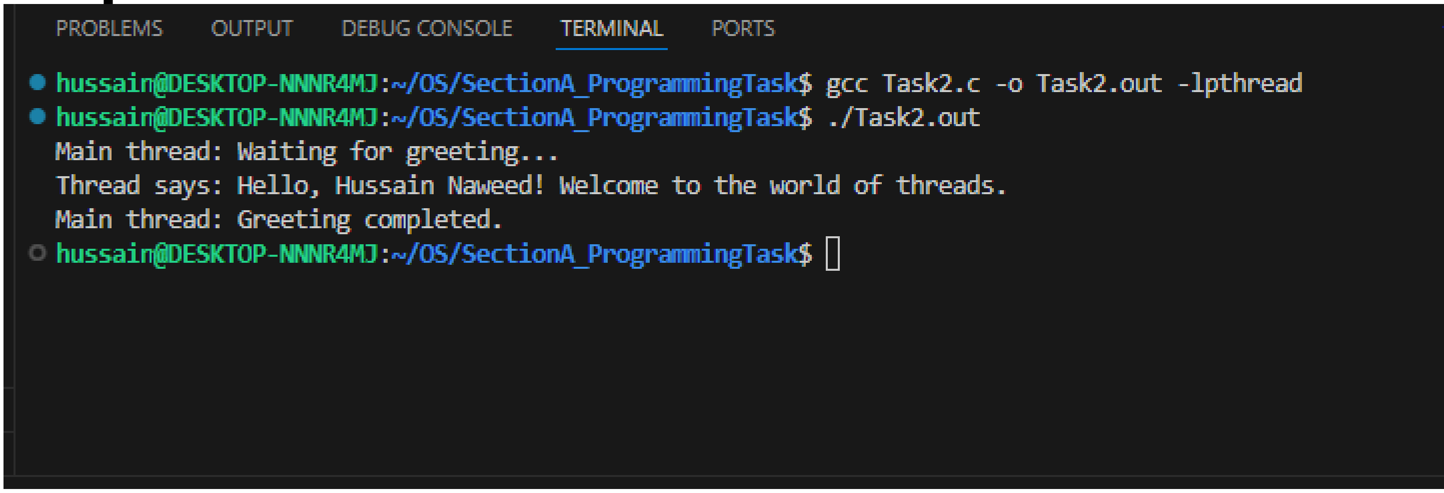
```
Main thread: Waiting for greeting...
Thread says: Hello, Ali! Welcome to the world of threads.
Main thread: Greeting completed.
```

Code :



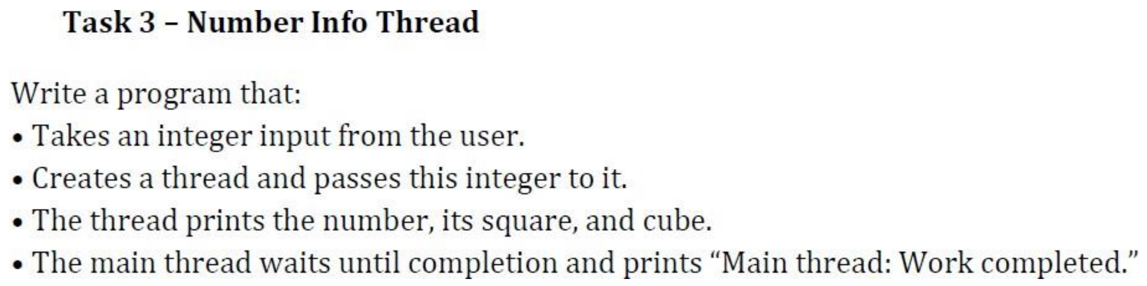
```

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
void *greet(void *arg) {
    char *name = (char*)arg;
    printf("Thread says: Hello, %s! Welcome to the world of threads.
    free(name);
    return NULL;
}
int main() {
    pthread_t tid;
    char *name = strdup("Hussain Naweed");

    printf("Main thread: Waiting for greeting...\n");
    pthread_create(&tid, NULL, greet, name);
    pthread_join(tid, NULL);
    printf("Main thread: Greeting completed.\n");
    return 0;
}

```

Output:



```
hussain@DESKTOP-NNNR4MJ:~/OS/SectionA_ProgrammingTask$ gcc Task2.c -o Task2.out -lpthread
hussain@DESKTOP-NNNR4MJ:~/OS/SectionA_ProgrammingTask$ ./Task2.out
Main thread: Waiting for greeting...
Thread says: Hello, Hussain Naweed! Welcome to the world of threads.
Main thread: Greeting completed.
hussain@DESKTOP-NNNR4MJ:~/OS/SectionA_ProgrammingTask$ 
```

Task_3 Number informed Thread:

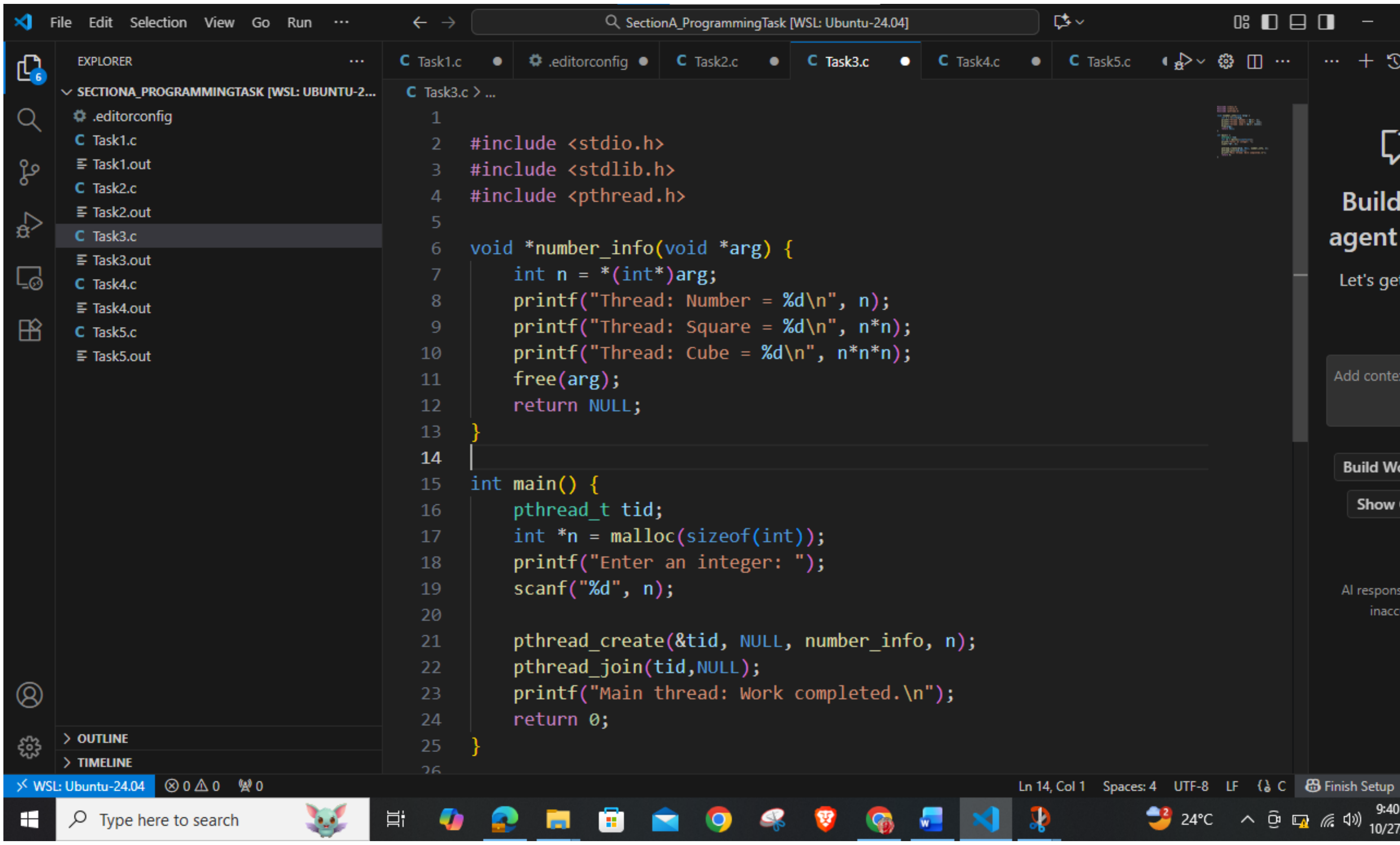


Task 3 – Number Info Thread

Write a program that:

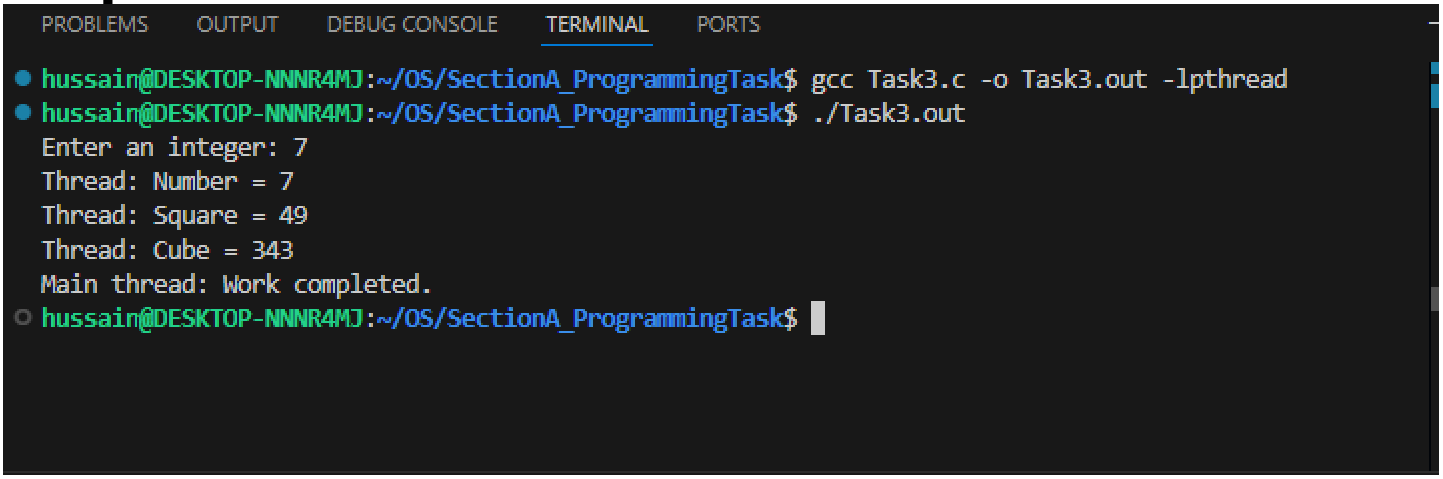
- Takes an integer input from the user.
- Creates a thread and passes this integer to it.
- The thread prints the number, its square, and cube.
- The main thread waits until completion and prints “Main thread: Work completed.”

Code :



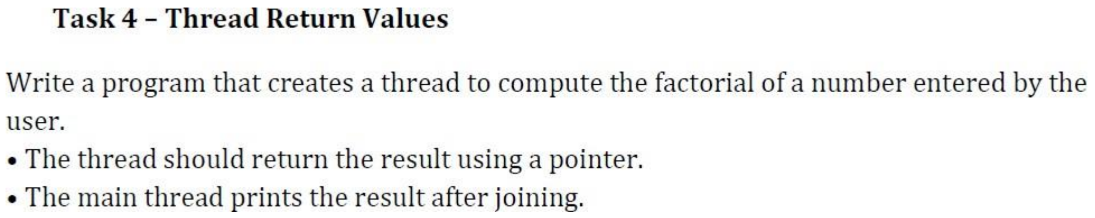
```c

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

void *number_info(void *arg) {
    int n = *(int*)arg;
    printf("Thread: Number = %d\n", n);
    printf("Thread: Square = %d\n", n*n);
    printf("Thread: Cube = %d\n", n*n*n);
    free(arg);
    return NULL;
}

int main() {
    pthread_t tid;
    int *n = malloc(sizeof(int));
    printf("Enter an integer: ");
    scanf("%d", n);

    pthread_create(&tid, NULL, number_info, n);
    pthread_join(tid,NULL);
    printf("Main thread: Work completed.\n");
    return 0;
}
```

Output:



```
hussain@DESKTOP-NNNR4MJ:~/OS/SectionA_ProgrammingTask$ gcc Task3.c -o Task3.out -lpthread
hussain@DESKTOP-NNNR4MJ:~/OS/SectionA_ProgrammingTask$ ./Task3.out
Enter an integer: 7
Thread: Number = 7
Thread: Square = 49
Thread: Cube = 343
Main thread: Work completed.
hussain@DESKTOP-NNNR4MJ:~/OS/SectionA_ProgrammingTask$ 
```

Task_4 Thread Return Values:

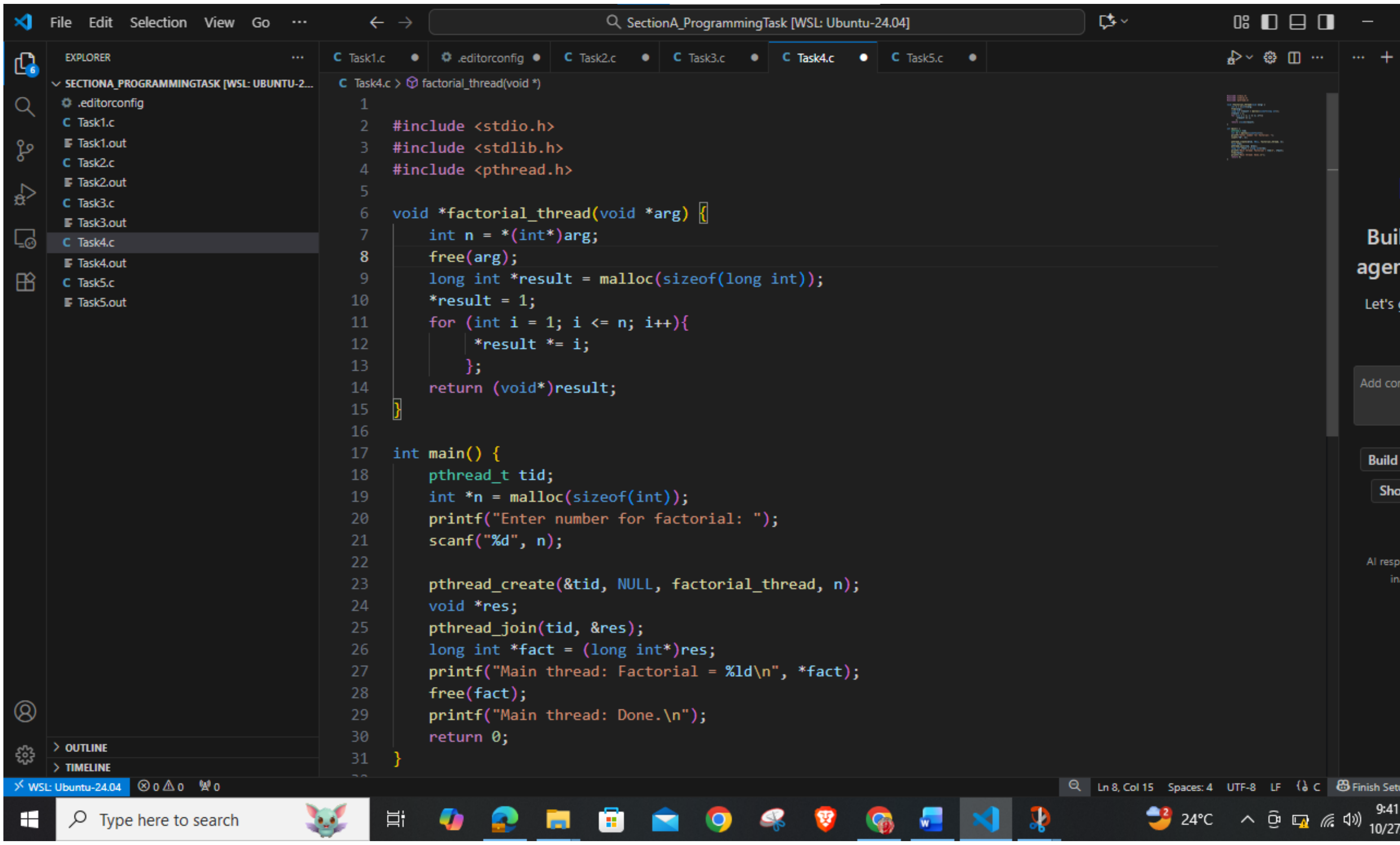


Task 4 – Thread Return Values

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

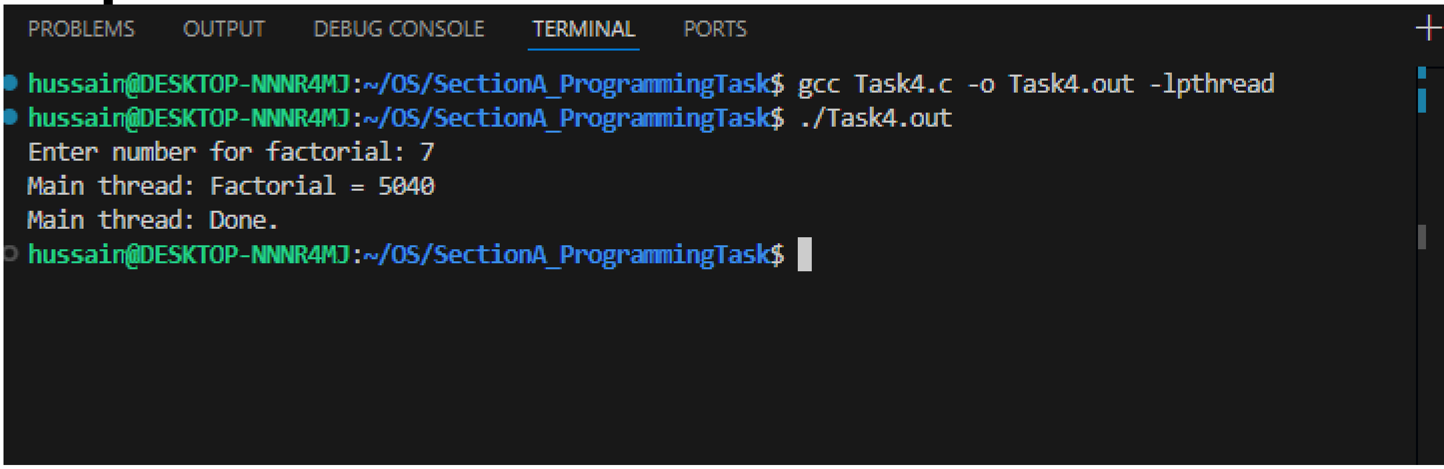
- The thread should return the result using a pointer.
- The main thread prints the result after joining.

Code :



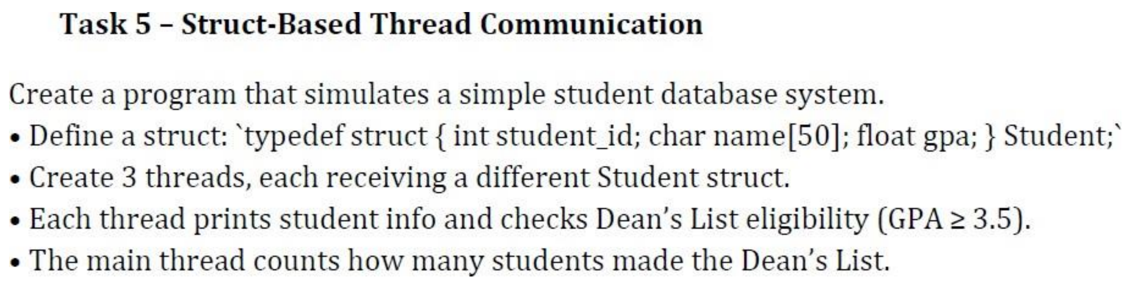
```c

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

void *factorial_thread(void *arg) {
    int n = *(int*)arg;
    free(arg);
    long int *result = malloc(sizeof(long int));
    *result = 1;
    for (int i = 1; i <= n; i++){
        *result *= i;
    };
    return (void*)result;
}

int main() {
    pthread_t tid;
    int *n = malloc(sizeof(int));
    printf("Enter number for factorial: ");
    scanf("%d", n);

    pthread_create(&tid, NULL, factorial_thread, n);
    void *res;
    pthread_join(tid, &res);
    long int *fact = (long int*)res;
    printf("Main thread: Factorial = %ld\n", *fact);
    free(fact);
    printf("Main thread: Done.\n");
    return 0;
}
```

Output:



```
hussain@DESKTOP-NNNR4MJ:~/OS/SectionA_ProgrammingTask$ gcc Task4.c -o Task4.out -lpthread
hussain@DESKTOP-NNNR4MJ:~/OS/SectionA_ProgrammingTask$ ./Task4.out
Enter number for factorial: 7
Main thread: Factorial = 5040
Main thread: Done.
hussain@DESKTOP-NNNR4MJ:~/OS/SectionA_ProgrammingTask$
```

Task-5 Structure base thread communication:

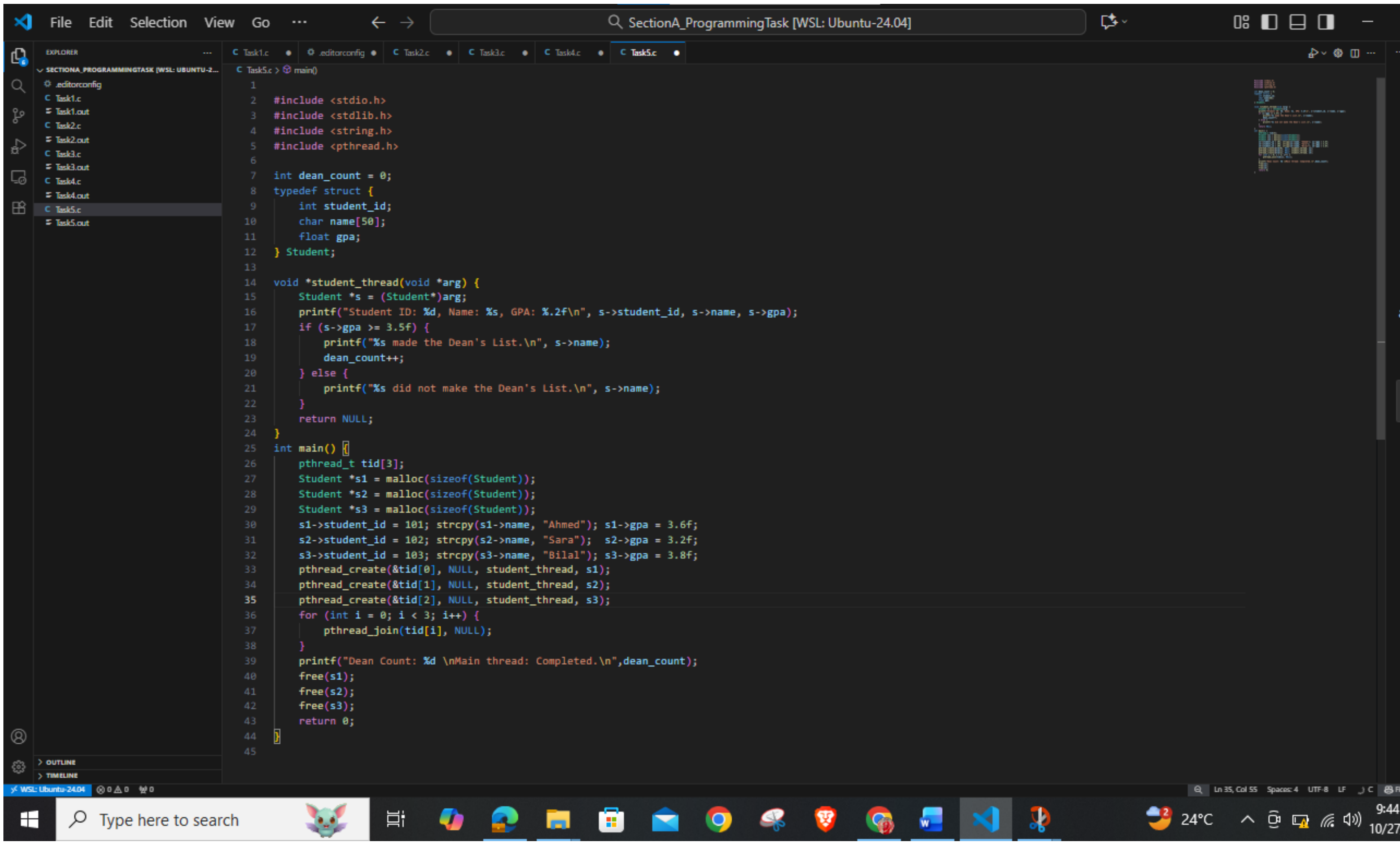


Task 5 – Struct-Based Thread Communication

Create a program that simulates a simple student database system.

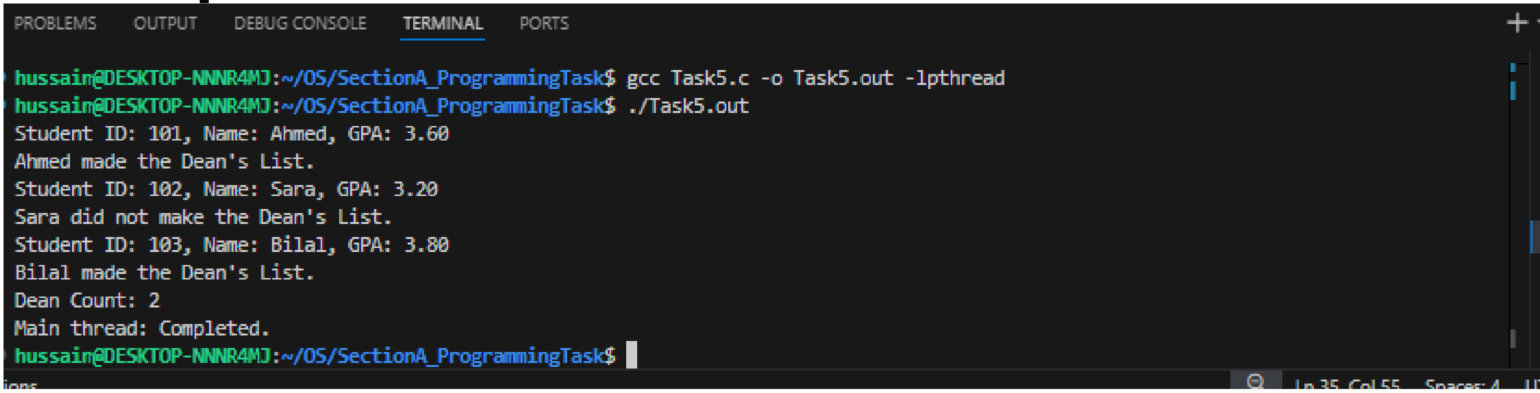
- Define a struct: `typedef struct { int student_id; char name[50]; float gpa; } Student;`
- Create 3 threads, each receiving a different Student struct.
- Each thread prints student info and checks Dean’s List eligibility (GPA ≥ 3.5).
- The main thread counts how many students made the Dean’s List.

Code:



```c

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

int dean_count = 0;
typedef struct {
    int student_id;
    char name[50];
    float gpa;
} Student;

void *student_thread(void *arg) {
    Student *s = (Student*)arg;
    printf("Student ID: %d, Name: %s, GPA: %.2f\n", s->student_id, s->name, s->gpa);
    if (s->gpa >= 3.5f) {
        printf("%s made the Dean's List.\n", s->name);
        dean_count++;
    } else {
        printf("%s did not make the Dean's List.\n", s->name);
    }
    return NULL;
}
int main() {
    pthread_t tid[3];
    Student *s1 = malloc(sizeof(Student));
    Student *s2 = malloc(sizeof(Student));
    Student *s3 = malloc(sizeof(Student));
    s1->student_id = 101; strcpy(s1->name, "Ahmed"); s1->gpa = 3.6f;
    s2->student_id = 102; strcpy(s2->name, "Sara");  s2->gpa = 3.2f;
    s3->student_id = 103; strcpy(s3->name, "Bilal"); s3->gpa = 3.8f;
    pthread_create(&tid[0], NULL, student_thread, s1);
    pthread_create(&tid[1], NULL, student_thread, s2);
    pthread_create(&tid[2], NULL, student_thread, s3);
    for (int i = 0; i < 3; i++) {
        pthread_join(tid[i], NULL);
    }
    printf("Dean Count: %d \nMain thread: Completed.\n",dean_count);
    free(s1);
    free(s2);
    free(s3);
    return 0;
}

```

Output:



```
hussain@DESKTOP-NNNR4MJ:~/OS/SectionA_ProgrammingTask$ gcc Task5.c -o Task5.out -lpthread
hussain@DESKTOP-NNNR4MJ:~/OS/SectionA_ProgrammingTask$ ./Task5.out
Student ID: 101, Name: Ahmed, GPA: 3.60
Ahmed made the Dean's List.
Student ID: 102, Name: Sara, GPA: 3.20
Sara did not make the Dean's List.
Student ID: 103, Name: Bilal, GPA: 3.80
Bilal made the Dean's List.
Dean Count: 2
Main thread: Completed.
hussain@DESKTOP-NNNR4MJ:~/OS/SectionA_ProgrammingTask$
```

Section :B Short Questions :

1. Define an Operating System:

An **Operating System (OS)** is a special type of system software that acts as an intermediary between computer hardware and the user. It manages all hardware components, software applications, and system resources, ensuring that programs run efficiently and users can interact with the machine smoothly.

2. Primary Function of the CPU Scheduler:

The **main purpose** of the CPU scheduler is to determine which process from the ready queue should be assigned to the CPU for execution next. It ensures effective CPU utilization by choosing the most appropriate process at any given moment.

3. Three States of a Process:

A process in an operating system can exist in the following main states:

- **Ready:** The process is loaded into main memory and waiting to be assigned the CPU.
- **Running:** The process is currently being executed by the CPU.
- **Waiting (Blocked):** The process cannot proceed until a certain event occurs, such as the completion of an I/O operation.

4. Process Control Block (PCB):

A **Process Control Block (PCB)** is a crucial data structure maintained by the operating system that contains all vital information about a specific process. This information includes the process identification number (PID), its current state, register contents, memory usage details, and other data necessary for process management. The OS uses the PCB to track and control the execution of every process in the system.

5. Difference Between a Process and a Program:

A **program** refers to a passive collection of instructions stored on a disk or in secondary storage. In contrast, a **process** represents the active execution of that program in main memory, including its current state, data, and resources. In simple terms, a program becomes a process when it is loaded and running in memory.

6. Context Switching:

Context switching occurs when the CPU changes from executing one process to another. During this operation, the OS saves the current process's state (such as register values and the program counter) into its PCB and loads the saved state of the next process. This mechanism allows multiple processes to share the CPU effectively, creating the illusion of simultaneous execution.

7. CPU Utilization and Throughput:

- **CPU Utilization:** This metric represents how effectively the CPU is being used. It indicates the percentage of time the processor is actively working on executing processes rather than remaining idle.
- **Throughput:** It measures the total number of processes completed within a specific time frame. Higher throughput implies that the system is completing more tasks efficiently.

8. Turnaround Time:

Turnaround time is defined as the total time taken from the moment a process is submitted to the operating system until it finishes its execution. It includes all phases such as waiting time in the ready queue, execution time, and any time spent in I/O operations.

9. Waiting Time Calculation:

The **waiting time** for a process is the total duration that it remains in the ready queue waiting for CPU allocation. It can be computed using the following formula:

$$\text{Waiting Time} = \text{Turnaround Time} - \text{Burst Time}$$

Where **burst time** represents the actual CPU execution time required by the process.

10. Response Time:

Response time is the time interval between the submission of a process and the production of its first response or output. In interactive systems, minimizing response time is critical because it directly affects how quickly users perceive feedback from the system.

11. Preemptive Scheduling:

Preemptive scheduling is a type of CPU scheduling method where the operating system has the authority to interrupt a currently running process and allocate the CPU to another process before the first one completes its execution. This allows the system to handle high-priority or time-sensitive processes efficiently, ensuring better responsiveness and fairness in multitasking environments.

12. Non-Preemptive Scheduling:

In **non-preemptive scheduling**, once a process starts execution on the CPU, it continues to run until it either completes its execution or voluntarily releases the CPU (for example, when waiting for I/O). The operating system does not forcibly remove the process from the CPU during this time. This approach is simpler but can lead to longer waiting times for other processes.

13. Two Advantages of Round Robin Scheduling:

1. **Fair CPU Distribution:** Every process receives an equal share of CPU time in a cyclic order, which ensures fairness among all processes.
2. **Reduced Waiting Time for Short Jobs:** Shorter tasks get frequent CPU access, decreasing their waiting time and improving response in time-sharing systems.

14. Major Drawback of the SJF (Shortest Job First) Algorithm:

A key disadvantage of the **Shortest Job First (SJF)** scheduling algorithm is the problem of **starvation**. When shorter jobs keep arriving continuously, longer processes may be postponed indefinitely, never getting a chance to execute.

15. CPU Idle Time:

CPU idle time refers to the duration when the processor is not executing any process. During this period, the CPU remains inactive or unused, often waiting for new tasks or for I/O operations to complete. Minimizing idle time helps in achieving better CPU utilization.

16. Two Common Goals of CPU Scheduling Algorithms:

1. **Maximize CPU Utilization:** Ensure that the processor remains busy executing tasks as much as possible.
2. **Minimize Waiting and Turnaround Time:** Reduce the time processes spend waiting in the ready queue and the total time taken to complete execution.

17. Two Reasons for Process Termination:

1. **Normal Completion:** The process successfully completes its execution and releases all allocated resources.
2. **Abnormal Termination:** The process is forcefully stopped because of an error, failure, or termination request from another process (for instance, through a kill command).

18. Purpose of wait() and exit() System Calls:

- **wait()**: This system call is used by a parent process to pause its execution until one or more of its child processes finish. It allows synchronization between parent and child processes.
- **exit()**: The exit() system call is used by a process to terminate its execution. It releases the resources held by the process and sends an exit status back to the parent process.

19. Shared Memory vs. Message Passing:

Aspect	**Shared Memory**	**Message Passing**
Communication Method	Processes share a common memory region to exchange data.	Processes exchange information by sending and receiving messages.
Speed	Faster communication but requires proper synchronization (e.g., using semaphores).	Slower in comparison but safer and easier to implement.
System Type	Used mainly in tightly coupled systems.	Suitable for distributed systems.
Example	Inter-process communication in UNIX using shared	Communication in client-server or

Aspect	Shared Memory	Message Passing
	segments.	distributed applications.

20. Thread vs. Process:

Aspect	Thread	Process
Definition	A lightweight execution unit within a process.	A heavy, independent program in execution.
Memory Usage	Threads share the same memory and resources within the process.	Each process has its own separate memory and resources.
Speed	Context switching between threads is faster.	Switching between processes takes more time.
Dependency	Multiple threads can run concurrently in a single process.	Processes run independently without sharing memory.
Example	Web browser tabs as multiple threads within one application.	Running two separate applications like Word and Chrome.

21. Multithreading:

Multithreading is the capability of a single process to execute multiple threads concurrently within a shared memory space. Each thread runs independently but shares resources like data and code with other threads. This improves performance and resource utilization, particularly in systems where multiple operations can be performed simultaneously (e.g., web servers handling many requests).

22. CPU-bound vs. I/O-bound Processes:

Aspect	CPU-bound Process	I/O-bound Process
Main Activity	Spends most of its time performing computations.	Spends most of its time waiting for I/O operations.
Resource Demand	Requires more CPU time and less I/O time.	Needs more I/O time and less CPU processing.
Examples	Scientific simulations, encryption, complex calculations.	Reading files, printing, web communication.

23. What Are the Main Responsibilities of the Dispatcher?

The **dispatcher** is responsible for handing control of the CPU to the process selected by the scheduler. It performs the context switching operation by saving the state of the currently running process, loading the state of the next one, and then starting its execution. Essentially, it manages the transition from one process to another within the CPU.

24. Define Starvation and Aging in Process Scheduling:

- **Starvation:** A condition in which a process remains in the waiting state for an extended period because other processes continuously get preference for CPU time.
- **Aging:** A technique used to prevent starvation by gradually increasing the priority of waiting processes, ensuring that all eventually get CPU time.

25. What Is a Time Quantum (or Time Slice)?

A **time quantum** (also called a time slice) is a small, fixed interval of CPU time that each process receives in **Round Robin scheduling**. After this time expires, the process is preempted and placed back at the end of the ready queue, ensuring fair distribution of CPU time.

26. What Happens When the Quantum Is Too Large or Too Small?

- **If the time quantum is too long:** The system behaves like the **First-Come, First-Served (FCFS)** algorithm, leading to slower response times.
- **If the time quantum is too short:** The CPU will perform frequent context switches, causing higher overhead and reducing overall efficiency.

27. Define the Turnaround Ratio (TR/TS):

The **turnaround ratio** compares the total time a process takes to complete (turnaround time) with the CPU time it actually required (service time).

Formula:

$$\text{Turnaround Ratio} = \frac{\text{Turnaround Time}}{\text{Service Time}}$$

A higher ratio indicates more time spent waiting relative to processing.

28. What Is the Purpose of a Ready Queue?

The **ready queue** contains all processes that are prepared to execute but are currently waiting for the CPU. When the CPU becomes available, the scheduler selects one process from this queue based on the scheduling algorithm in use.

29. Differentiate Between a CPU Burst and an I/O Burst:

- **CPU Burst:** A period during which a process actively uses the CPU for computation.
- **I/O Burst:** A phase when the process waits for input/output operations (like reading a file or sending data to a printer).

Processes typically alternate between CPU bursts and I/O bursts throughout their lifetime.

30. Which Scheduling Algorithm Is Starvation-Free, and Why?

The **Round Robin scheduling** algorithm is considered **starvation-free** because it ensures that every process gets CPU time in a cyclic order. Each process receives a fixed time slice, so none of them can be indefinitely delayed or ignored by the scheduler.

31. Outline the Main Steps Involved in Process Creation in UNIX:

The following steps occur during process creation in a UNIX-based system:

1. The **parent process** invokes the fork() system call to create a new child process.
2. The **child process** may use the exec() system call to replace its memory space with a new program.
3. The **parent** can call wait() to pause its execution until the child completes.
4. Once finished, the **child** process terminates using exit(), and its exit status is returned to the parent.

This parent-child relationship allows UNIX to manage processes hierarchically.

32. Define Zombie and Orphan Processes:

- **Zombie Process:** A process that has finished execution but remains in the process table because its parent hasn’t collected its termination status using the wait() call.
- **Orphan Process:** A process whose parent has terminated before it did. These processes are automatically adopted by the **init** (or systemd) process, which handles their cleanup.

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

Aspect	**Priority Scheduling**	**Shortest Job First (SJF)**
Scheduling Basis	CPU is assigned based on process priority.	CPU is given to the process with the shortest CPU burst time.
Starvation Risk	Low-priority processes may starve.	Long jobs may experience starvation.
Priority Type	Determined by importance or aging mechanism.	Determined solely by the expected CPU burst time.
Focus	Importance or urgency of a process.	Efficiency and fast completion of short jobs.

34. Define Context Switch Time and Explain Why It Is Considered Overhead:

Context switch time is the duration the CPU takes to save the state of one process and load the state of another during a context switch.
It is considered **overhead** because no real user computation occurs during this period — the CPU is busy with administrative tasks rather than executing user instructions. Excessive context switching can reduce overall system performance.

35. List and Briefly Describe the Three Levels of Schedulers in an Operating System:

Scheduler Type	**Main Function**	**When It Operates**	**Example/Description**

Scheduler Type	**Main Function**	**When It Operates**	**Example/Description**
Long-Term Scheduler	Decides which new processes should be brought into the ready queue (controls the degree of multiprogramming).	When new jobs are submitted.	Adds new user jobs from secondary storage to main memory.
Medium-Term Scheduler	Temporarily suspends or resumes processes to manage memory usage.	When the system is overloaded or memory is scarce.	Swaps background or idle processes out and restores them later.
Short-Term Scheduler	Selects which ready process gets CPU next.	Runs very frequently (milliseconds).	Chooses processes based on algorithms like Round Robin or Priority.

36. Differentiate Between User Mode and Kernel Mode:

User Mode	**Kernel Mode**
Executes user-level programs such as editors, browsers, and games.	Runs operating system core functions and kernel routines.
Has restricted access to system resources for safety.	Has full and unrestricted access to hardware and memory.
Requires system calls to request privileged operations.	Executes privileged instructions directly.
Example: Running a text editor.	Example: Managing device drivers or process scheduling.

Section-C: Technical / Analytical Questions

1. Describe the Complete Life Cycle of a Process with a Neat Diagram Showing Transitions Between New, Ready, Running, Waiting, and Terminated States.

Answer:
The **process life cycle** illustrates the different stages that a process passes through from its creation until completion in an operating system. These states represent how a process moves and interacts with system resources during its lifetime.

Process States:

1. **New:**
   In this state, the process is just being created. The operating system allocates essential resources such as memory and creates a **Process Control Block (PCB)** to store process information.
2. **Ready:**
   The process is fully prepared for execution but is waiting in the **ready queue** for the CPU to be assigned by the scheduler.

3. **Running:**
   At this stage, the process is being executed by the CPU and actively performing instructions.

4. **Waiting (or Blocked):**
   The process cannot proceed further until a specific event occurs — for example, completion of an **I/O operation** or receiving required input.

5. **Terminated (or Exit):**
   The process has completed its execution, and the operating system releases all resources (like memory and open files) allocated to it.

State Transitions:

Processes move between these states as system events occur:

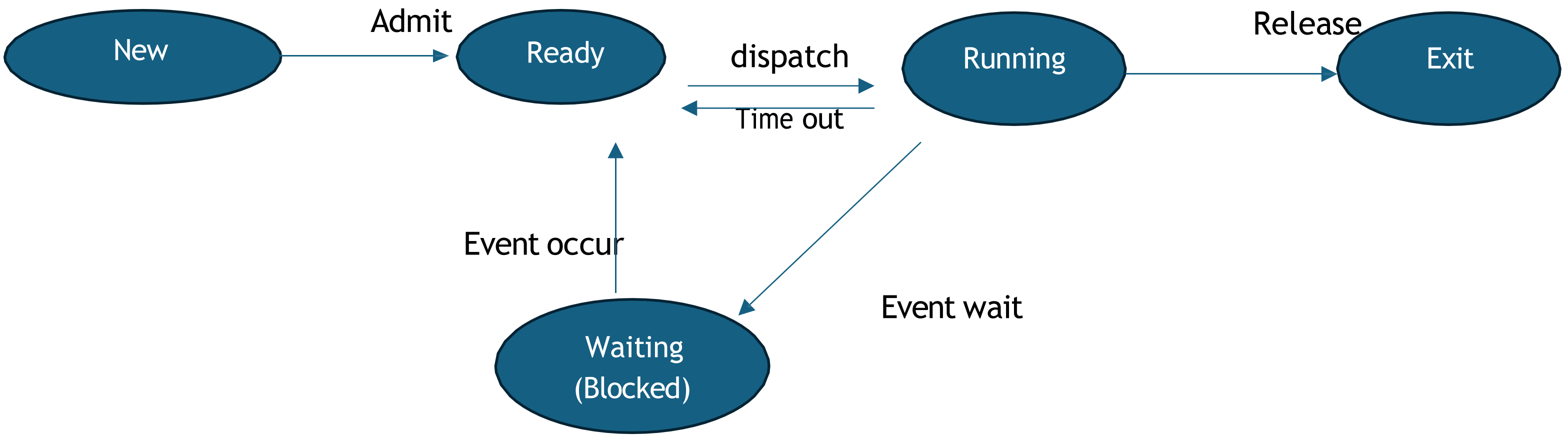
- **New → Ready:**
  When a new process is admitted to the system by the long-term scheduler, it is placed in the ready queue.

- **Ready → Running:**
  The short-term scheduler dispatches the process from the ready queue to the CPU for execution.

- **Running → Waiting:**
  If the process requests I/O or waits for an event, it transitions to the waiting state.

- **Waiting → Ready:**
  Once the waiting event (like I/O completion) occurs, the process moves back to the ready queue.

Running → Terminated:
When the process finishes execution or is killed, it transitions to the terminated state, and the OS deallocates its resources.



2. Write a Short Note on Context Switch Overhead and Describe What Information Must Be Saved and Restored.

Answer:

Definition of Context Switch:

A **context switch** takes place when the CPU stops executing one process (or thread) and switches to another. This mechanism enables **multitasking** by allowing multiple processes to share the CPU efficiently. During a switch, the system saves the current process's state and restores the state of the next process so that each process can continue from where it left off.

When Context Switches Occur:

Context switches happen in several scenarios, such as:

- A process moves from **Running** → **Waiting** (for I/O or an event).
- A process is **preempted** when its time quantum expires.
- A **higher-priority process** becomes ready.
- The current process **terminates**.

Context Switch Overhead:

The **time consumed by the CPU** during a context switch is called **context switch overhead**.
This time is considered overhead because:

- No actual process execution or user work happens during this period.
- The CPU is occupied saving and restoring process information.
- The time cost depends on hardware design, the operating system, and the number of processes running.

Typically, this takes a few microseconds to milliseconds — though frequent switching can reduce CPU efficiency.

Information Saved and Restored During a Context Switch:

When switching between processes, the operating system must store and later restore certain process data, including:

1. **CPU Registers:**
   General-purpose and special-purpose registers (like data registers, address registers, and flag registers) that hold temporary values used by the CPU.
2. **Program Counter (PC):**
   Contains the address of the next instruction the process will execute when it resumes.
3. **Stack Pointer:**
   Points to the current position in the process's stack, maintaining function calls, local variables, and return addresses.
4. **Process State:**
   Indicates whether the process is running, ready, or waiting. This information is recorded in the **PCB (Process Control Block)**.

5. **Memory and I/O Information:**
   Includes page tables, memory mapping data, and I/O device statuses, ensuring the process continues correctly after resuming.

3. List and Explain the Components of a Process Control Block (PCB).

Answer:
A **Process Control Block (PCB)** is a critical data structure used by the operating system to store all relevant details about a process. Each process has its own PCB, which helps the OS manage and control its execution.

Role of PCB:

The PCB acts as the process's identity card—it stores essential information that allows the operating system to schedule processes, switch between them, and track their status.

Storage Location:

PCBs are stored in the **kernel's memory area**, typically organized in a table called the **process table**.

Main Components of a PCB:

1. **Process ID (PID):**
   A unique number assigned to each process for identification.
2. **Process State:**
   Describes the current status of the process (e.g., New, Ready, Running, Waiting, or Terminated).
3. **Program Counter:**
   Holds the address of the next instruction to execute, ensuring continuity during process resumption.
4. **CPU Registers:**
   Store temporary computational data and must be saved during context switching.
5. **Memory Management Information:**
   Contains details about the process's memory allocation, such as page tables or segment tables.
6. **Accounting Information:**
   Includes process priority, CPU usage time, job number, and other statistics used for scheduling and monitoring.
7. **I/O Status Information:**
   Tracks open files, input/output devices in use, and pending I/O requests for the process.

4. Differentiate Between Long-Term, Medium-Term, and Short-Term Schedulers with Examples.

Answer:

A **scheduler** in an operating system decides which processes should run, when they should run, and how system resources should be allocated to them. There are **three major types** of schedulers, each with different functions and timing.

Type of Scheduler	Main Function	When It Works	Example
Long-Term Scheduler (Job	Determines which new processes should be brought from disk (secondary storage) into main	When new processes	Admitting new batch jobs

Type of Scheduler	Main Function	When It Works	Example
Scheduler)	memory. Controls the **degree of multiprogramming**.	are created.	into the ready queue.
Medium-Term Scheduler (Swapper)	Temporarily suspends and later resumes processes to manage memory more efficiently.	When memory is full or overloaded.	Swapping background or inactive processes out to disk and restoring them later.
Short-Term Scheduler (CPU Scheduler)	Selects which ready process should be executed next by the CPU.	Frequently (usually every few milliseconds).	Choosing the next process using Round Robin or Priority Scheduling.

Summary:

- The **long-term scheduler** controls which processes enter memory.
- The **medium-term scheduler** suspends and resumes processes to manage limited resources.
- The **short-term scheduler** makes rapid decisions on which process runs next.

5. Explain CPU Scheduling Criteria (Utilization, Throughput, Turnaround, Waiting, and Response) and Their Optimization Goals.

Answer:
CPU scheduling performance is evaluated using several key **criteria** that determine how efficiently processes are handled by the operating system.

1. CPU Utilization:

This criterion measures how effectively the CPU is being used. It reflects the percentage of time the processor is doing productive work instead of being idle.

Optimization Goal:
Maximize CPU utilization to ensure minimal idle time.

Scheduler Relation:

- **Long-Term Scheduler:** Balances CPU-bound and I/O-bound processes to prevent CPU overload or underuse.
- **Short-Term Scheduler:** Keeps the CPU busy by continuously allocating ready processes.

Example:
If a server maintains around 85% CPU utilization, it indicates efficient CPU usage.

2. Throughput:

Throughput refers to the total number of processes that complete execution per unit of time (e.g., processes per second).

Optimization Goal:
Increase throughput to ensure more processes are completed efficiently.

Scheduler Relation:

- **Long-Term Scheduler:** Controls how many processes enter memory to maintain an optimal workload.
- **Short-Term Scheduler:** Prioritizes quick tasks to improve completion rates.

Example:
A payroll system completing 100 jobs per hour has higher throughput than one completing only 50.

3. Turnaround Time:

The total time taken from process submission until its completion. It includes waiting time, execution time, and I/O operations.

Optimization Goal:
Minimize turnaround time for faster process completion.

Scheduler Relation:

- **Long-Term Scheduler:** Affects turnaround time by controlling process admission.
- **Short-Term Scheduler:** Prioritizes shorter jobs to finish quickly.
- **Medium-Term Scheduler:** Can affect turnaround time by swapping processes in and out of memory.

Example:
If a grading program takes 5 seconds from start to finish, its turnaround time is 5 seconds.

4. Waiting Time:

The cumulative amount of time a process spends waiting in the ready queue for CPU access.

Optimization Goal:
Minimize waiting time to improve responsiveness.

Scheduler Relation:

- **Short-Term Scheduler:** Directly impacts waiting time by how it chooses ready processes.
- **Long-Term Scheduler:** Affects waiting time indirectly by managing how many processes enter the system.

Example:
In a time-sharing system, if a process waits only 200 ms before execution, it has an efficient waiting time.

5. Response Time:

The time from when a process is submitted until it produces its first output or starts execution.

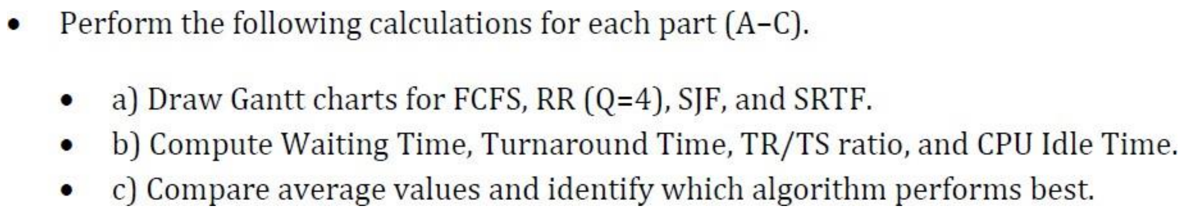
Optimization Goal:
Reduce response time for better interactivity, especially in user-facing systems.

Scheduler Relation:

- **Short-Term Scheduler:** Ensures interactive processes are dispatched quickly.
- **Long-Term Scheduler:** Influences how soon a process enters the ready queue.

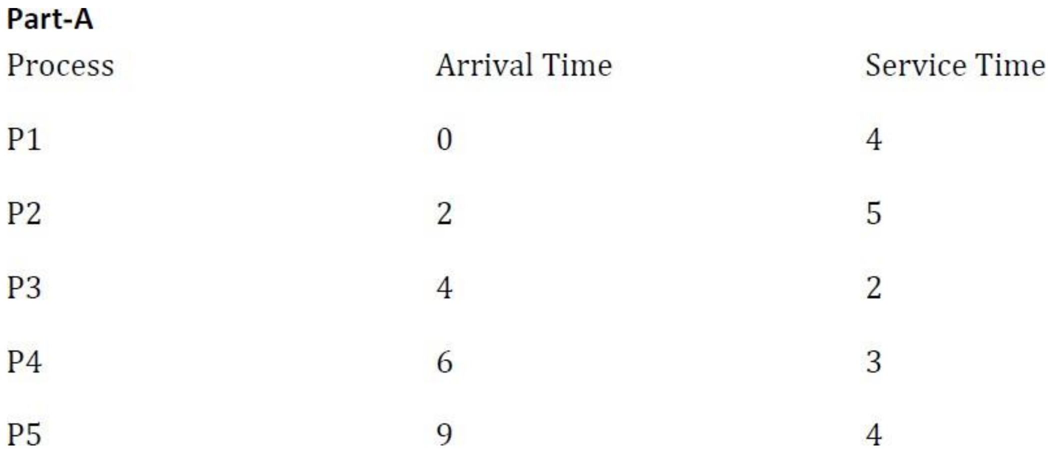
Example:
If a web page starts loading 100 milliseconds after a user clicks a link, the response time is 100 ms.

Section-D: CPU Scheduling Calculations



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

Part _ A :

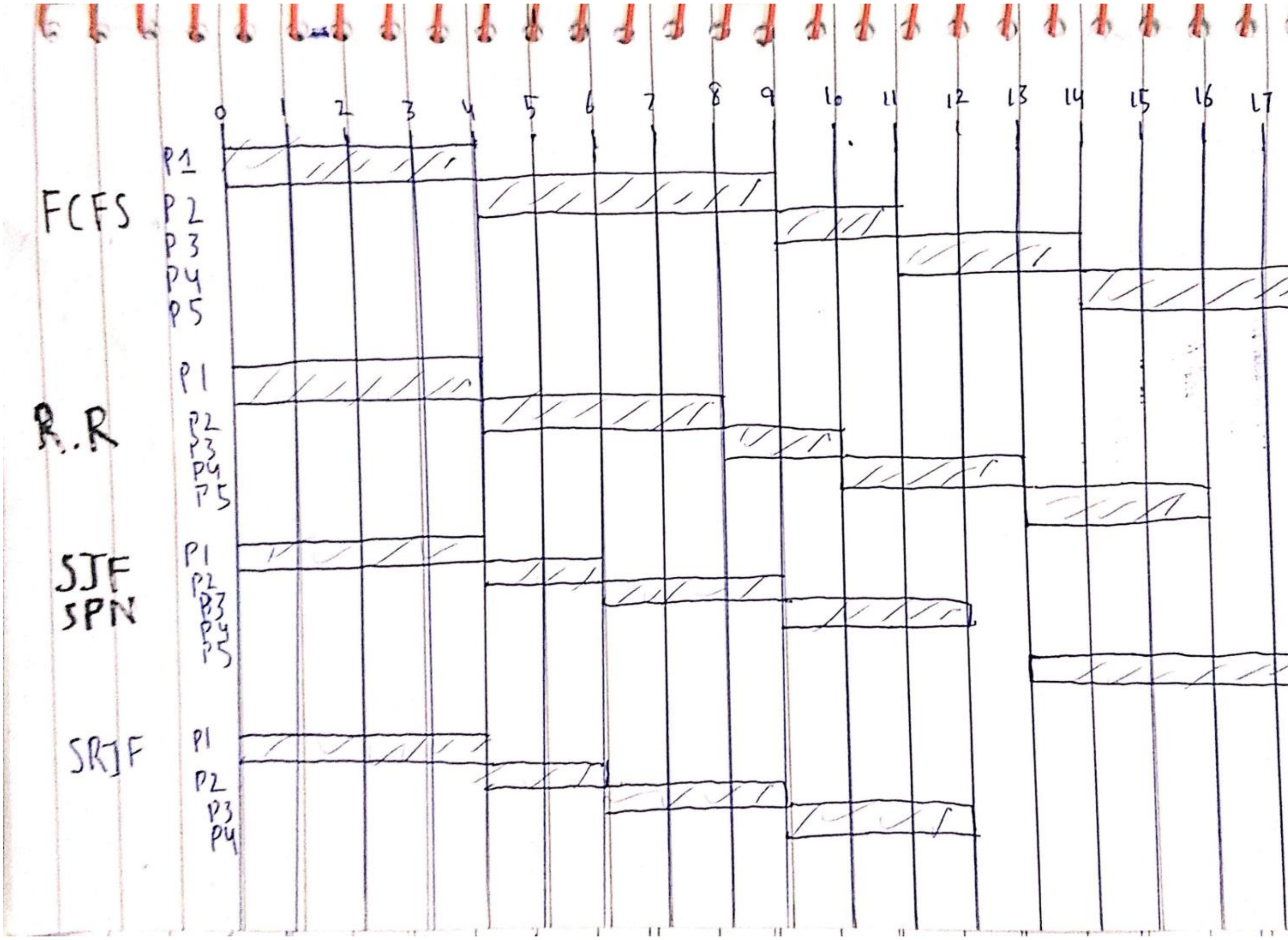


Part-A

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

Solution:

Grant Chat :


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
FCFS
P1
P2
P3
P4
P5
R.R
P1
P2
P3
P4
P5
SJF
SPN
P1
P2
P3
P4
P5
SRJF
P1
P2
P3
P4

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	2.6
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 TK/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	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	2.33
P_5	17	8	4	4	2.0

- By waiting time: (0+11+4+4+4)/5=4.6
- Avg Turnaround: (4+16+6+7+8)/5=8.2

Avg Tr/Trs Ration: (1+3.2+3+2.33+2)/5
=2.31

- CPU Idle time: 0

SJF 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

- By 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	32
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 Idea time: 0

Best Algorithms:-

SRTF and SJF are the best performing for this dataset.

- Lowest:- Average waiting and turnaround time
- Tr/Ts is the lowest as compared to others
- Optional for minimizing metrics

SRTF Analysis:

Process	Finish	Tr	Ts	Waiting	Tr/Ts
P_1	4	4	4	0	1.0
P_2	18	16	5	11	32
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 Idea time: 0

Best Algorithms:-

SRTF and SJF are the best performing for this dataset.

- Lowest:- Average waiting and turnaround time
- Tr/Ts is the lowest as compared to others
- Optional for minimizing metrics

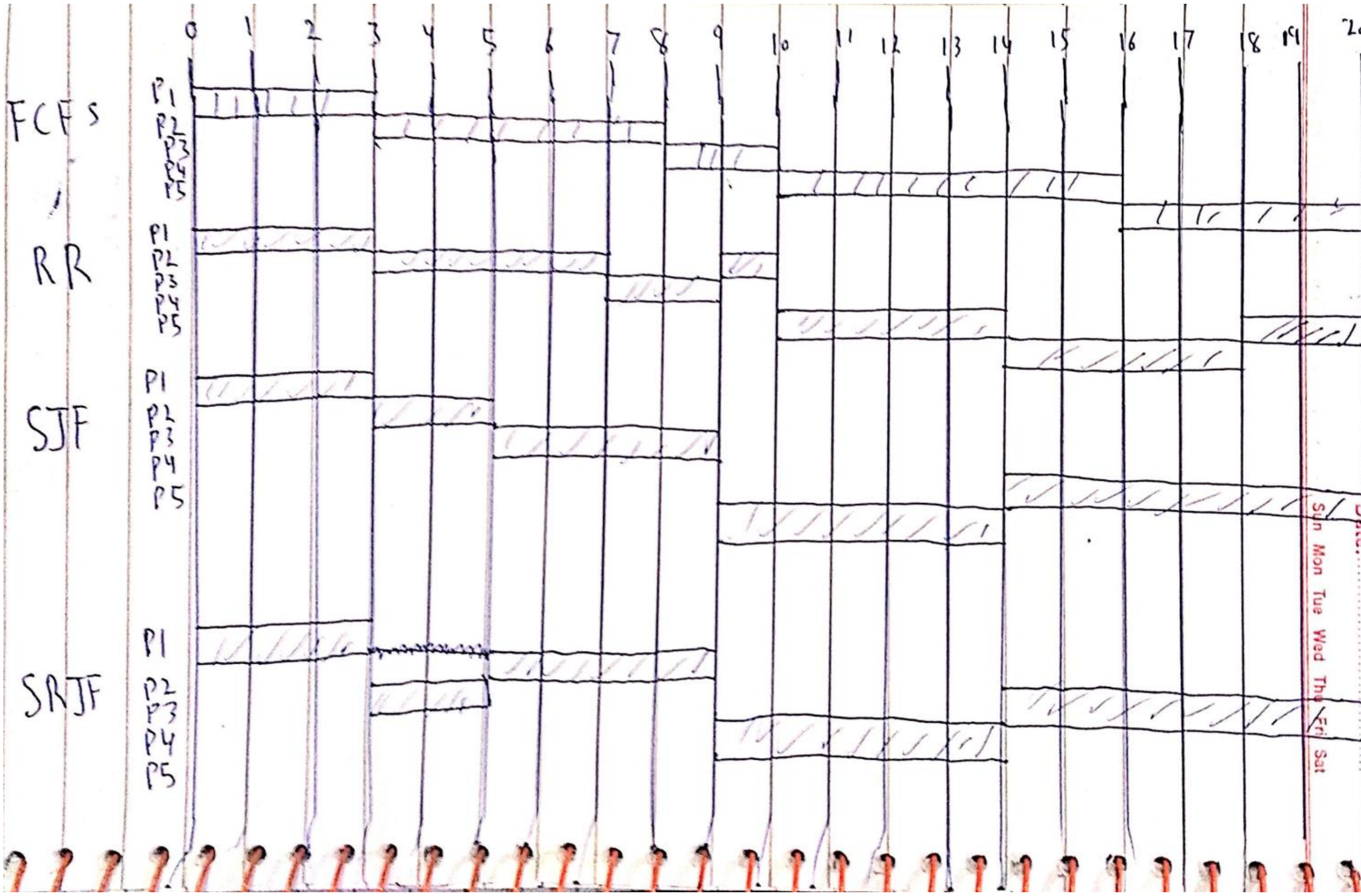
Part_B:

Part-B

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

Solution :

Chat :



Part B

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 time: (3+7+7+7+10)/5=6.8
- Avg Tr/Ts: (1+1.4+3.5+1.17+2.5)/5=1.96
- CPU Idle time: 0

Round Robin (Q=4) 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	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 Ratio: (1+1.8+3+1.83+2)/5= 9.3
- CPU Idle time: 0

SJF Analysis:

Process	Finish	Tr	Ts	Waiting	Tr/Ts
P_1	3	3	3	0	1.0
P_2	10	4	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 Turnaround: (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:

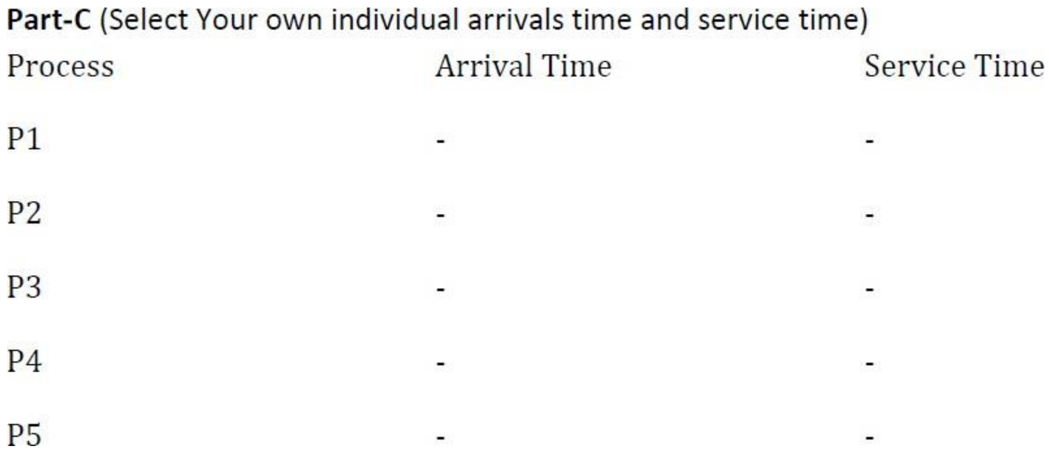
Process	Finish:	Tr	Ts	Waiting	Tr/Ts
P_1	3	3	3	0	1.0
P_2	16	9	5	4	1.8
P_3	5	2	2	0	1.0
P_4	20	11	6	5	1.83
P_5	14	4	4	0	1.0

- Avg Waiting: (0+4+0+5+0)/5 = 1.8
- Avg Turnaround: (3+9+2+11+4)/5 = 5.8
- Avg Tr/Ts: (1+1.8+1+1.83+1) /5 = 1.3
- CPU Idletime: 0

Best Algoritm:-

- SRTF and SJF are best performing for this dataset
- Lowest average waiting and turnaround time
- Tr/Ts is lowest as compared to others.
- ~~[illegible]~~ Optimal for minimizing metrices

Part C :

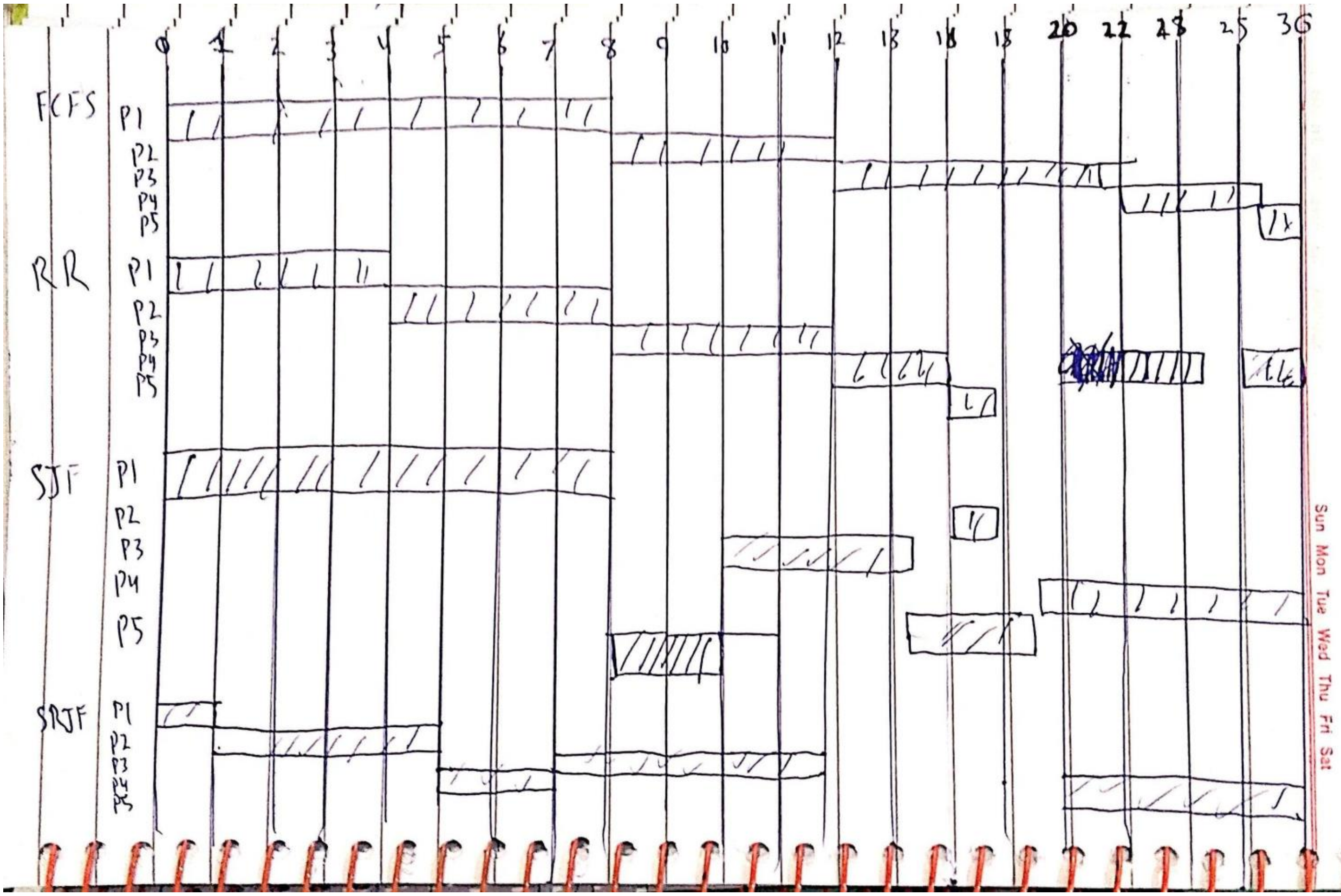


Part-C (Select Your own individual arrivals time and service time)

Process	Arrival Time	Service Time
P1	-	-
P2	-	-
P3	-	-
P4	-	-
P5	-	-

Solution :

Chat :


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
20
22
23
25
36
FCFS
P1
P2
P3
P4
P5
RR
P1
P2
P3
P4
P5
SJF
P1
P2
P3
P4
P5
SRJF
P1
P2
P3
P4
P5
Sun Mon Tue Wed Thu Fri Sat

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	4.66
P_5	28	24	2	22	12.0
Average		17		11.4	4.44

CPU ideal time = 0

Round Robin (Q-4) Analysis

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		18.6		13.0	3.84

CPU idle time : 0

SJF Analysis:

Process	Finish	Tr	Ts	Waiting	Tr/Ts
P_1	8	8	8	0	1.0
P_2	14	13	4	9	3.25
P_3	28	26	9	17	2.89
P_4	19	16	5	11	3.2
P_5	10	6	2	4	3.0
Avearage		13.8		8.2	2.67

CPU Idle time: 0

SRTF Analysis

Process	Finish	Tr	Ts	Waiting	Tr/Ts
P_1	19	19	8	11	2.38
P_2	5	4	4	0	1.0
P_3	28	26	9	17	2.89
P_4	12	9	5	4	1.8
P_5	7	3	2	1	1.5
Average		12.2		6.6	1.91

CPU Idle Time 0

Best Performing Algorithm

- SRTF is the best performing algorithm for this dataset
- Lowest Average turnaround and waiting time.
- OPtimal for minimizing average times precption allows short jobs to execute quickly