



National Textile University

Department of Computer Science

Subject:

Operating Systems

Submitted To:

Sir Nasir Mehmood

Submitted By:

Alishba Riasat

Registration No:

23-NTU-CS-1135

Assignment No:

1

Semester:

5th

Section-A: Programming Tasks

▪ Task 1 – Thread Information Display

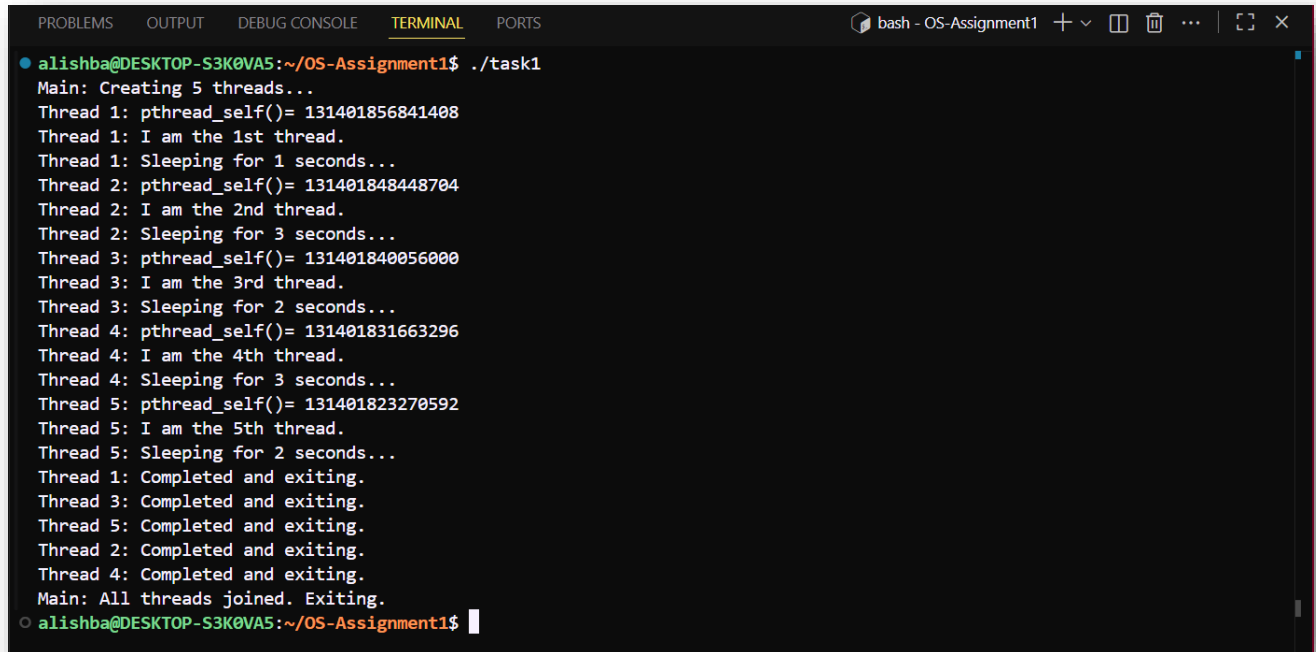
Write a program that creates 5 threads. Each thread should:

- 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.

Code:

```
1  /*
2  Name: Alishba Riasat
3  Reg No: 23-NTU-CS-1135
4  Task 1: Thread Information Display
5  */
6
7  #include <stdio.h>
8  #include <unistd.h>
9  #include <pthread.h>
10 #include <stdlib.h>
11
12 void* thread_work(void* arg) {
13     int thread_no = *(int*) arg;
14     printf("Thread %d: pthread_self()= %lu\n", thread_no, pthread_self());
15     if (thread_no == 1) {
16         printf("Thread %d: I am the 1st thread.\n", thread_no);
17     }
18     else if(thread_no == 2) {
19         printf("Thread %d: I am the 2nd thread.\n", thread_no);
20     }
21     else if(thread_no == 3) {
22         printf("Thread %d: I am the 3rd thread.\n", thread_no);
23     }
24     else{
25         printf("Thread %d: I am the %dth thread.\n", thread_no, thread_no);
26     }
27
28     int sleeptime = (rand()%3) + 1;
29     printf("Thread %d: Sleeping for %d seconds...\n", thread_no, sleeptime);
30     sleep(sleeptime);
31
32     printf("Thread %d: Completed and exiting.\n", thread_no);
33     return NULL;
34 }
35
36 int main() {
37     srand(getpid()); //Seed rand with PID
38     pthread_t threads[5];
39     int ids[5];
40
41     printf("Main: Creating 5 threads...\n");
42
43     for(int i=0; i<5; i++){
44         ids[i]= i+1;
45         pthread_create(&threads[i], NULL, thread_work, &ids[i]);
46         usleep(10000); // To avoid the race condition
47     }
48     for(int i=0; i<5; i++){
49         pthread_join(threads[i], NULL);
50     }
51
52     printf("Main: All threads joined. Exiting.\n");
53     return 0;
54 }
```

Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash - OS-Assignment1 + v [ ] [ ] ... [ ] [ ] x


alishba@DESKTOP-S3K0VA5:~/OS-Assignment1$ ./task1
Main: Creating 5 threads...
Thread 1: pthread_self()= 131401856841408
Thread 1: I am the 1st thread.
Thread 1: Sleeping for 1 seconds...
Thread 2: pthread_self()= 131401848448704
Thread 2: I am the 2nd thread.
Thread 2: Sleeping for 3 seconds...
Thread 3: pthread_self()= 131401840056000
Thread 3: I am the 3rd thread.
Thread 3: Sleeping for 2 seconds...
Thread 4: pthread_self()= 131401831663296
Thread 4: I am the 4th thread.
Thread 4: Sleeping for 3 seconds...
Thread 5: pthread_self()= 131401823270592
Thread 5: I am the 5th thread.
Thread 5: Sleeping for 2 seconds...
Thread 1: Completed and exiting.
Thread 3: Completed and exiting.
Thread 5: Completed and exiting.
Thread 2: Completed and exiting.
Thread 4: Completed and exiting.
Main: All threads joined. Exiting.
alishba@DESKTOP-S3K0VA5:~/OS-Assignment1$
```

▪ 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.

Code:



```
1  /*
2  Name: Alishba Riasat
3  Reg No: 23-NTU-CS-1135
4  Task 2: Personalized Greeting Thread
5  */
6
7  #include <stdio.h>
8  #include <unistd.h>
9  #include <pthread.h>
10
11 void* greet(void* arg) {
12     char* name = (char*)arg;
13     printf("Thread says: Hello, %s! Welcome to the world of threads.\n", name);
14     return NULL;
15 }
16
17 int main(){
18     pthread_t tid;
19     char name[50];
20
21     printf("Enter your name: ");
22     if(scanf("Is",name) !=1){
23         printf("Input Error.\n");
24         return 1;
25     }
26
27     printf("Main thread: Waiting for greeting...\n");
28     pthread_create(&tid, NULL, greet, name);
29     pthread_join(tid, NULL);
30     printf("Main thread: Greeting completed.\n");
31     return 0;
32 }
```

Output:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

bash - OS-Assignment1 + ▾ 🗑 ⋮ | 🪄 ✕

```
● alishba@DESKTOP-S3K0VA5:~/OS-Assignment1$ gcc Assignment1-task2.c -o task2 -lpthread
● alishba@DESKTOP-S3K0VA5:~/OS-Assignment1$ ./task2
Enter your name: Alishba
Main thread: Waiting for greeting...
Thread says: Hello, Alishba! Welcome to the world of threads.
Main thread: Greeting completed.
○ alishba@DESKTOP-S3K0VA5:~/OS-Assignment1$
```

Ln 32, Col 2 (673 selected) Spaces: 4 UTF-8 LF {} C 🐞 🪄 1h 21m 🔔

- **Task 3 – Number Info Thread**

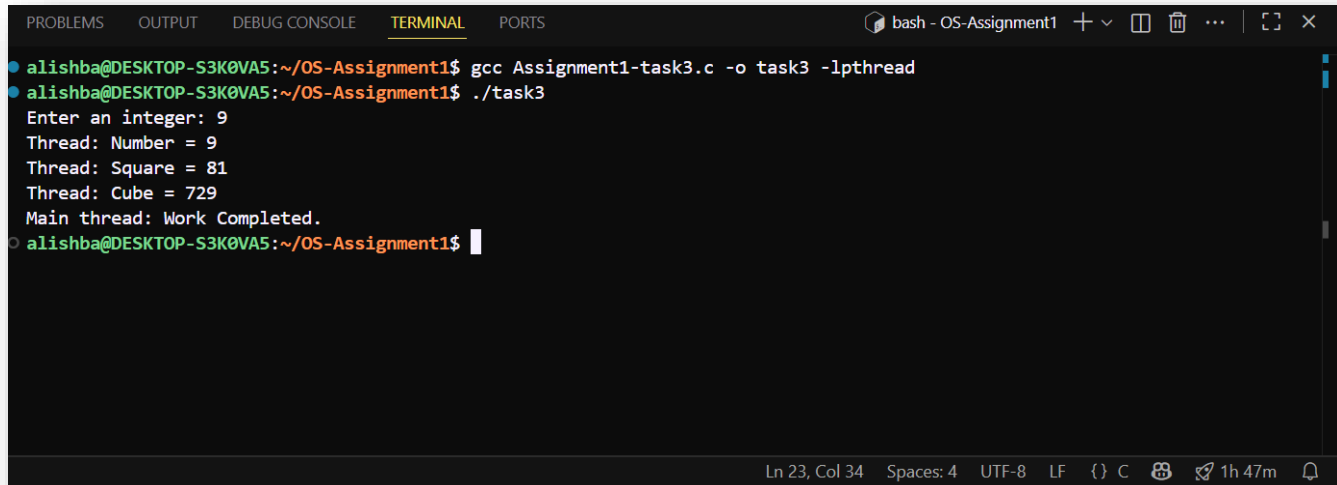
Write a program that:

- 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:

```
1  /*
2   Name: Alishba Riasat
3   Reg No: 23-NTU-CS-1135
4   Task 3: Number Info Thread
5   */
6
7  #include <stdio.h>
8  #include <unistd.h>
9  #include <pthread.h>
10
11 void* number_info(void* arg) {
12     int n = *(int*) arg;
13     printf("Thread: Number = %d\n", n);
14     printf("Thread: Square = %d\n", n*n);
15     printf("Thread: Cube = %d\n", n*n*n);
16     return NULL;
17 }
18
19 int main() {
20     pthread_t tid;
21     int number;
22
23     printf("Enter an integer: ");
24     if (scanf("%d", &number) != 1) {
25         printf("Invalid Input. \n");
26         return 1;
27     }
28     pthread_create(&tid, NULL, number_info, &number);
29     pthread_join(tid, NULL);
30     printf("Main thread: Work Completed.\n");
31     return 0;
32 }
```

Output:



```
bash - OS-Assignment1 + - [ ] ... [ ] x
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
alishba@DESKTOP-S3K0VA5:~/OS-Assignment1$ gcc Assignment1-task3.c -o task3 -lpthread
alishba@DESKTOP-S3K0VA5:~/OS-Assignment1$ ./task3
Enter an integer: 9
Thread: Number = 9
Thread: Square = 81
Thread: Cube = 729
Main thread: Work Completed.
alishba@DESKTOP-S3K0VA5:~/OS-Assignment1$
```

Ln 23, Col 34 Spaces: 4 UTF-8 LF {} C 1h 47m

Task 4 – Thread Return Values

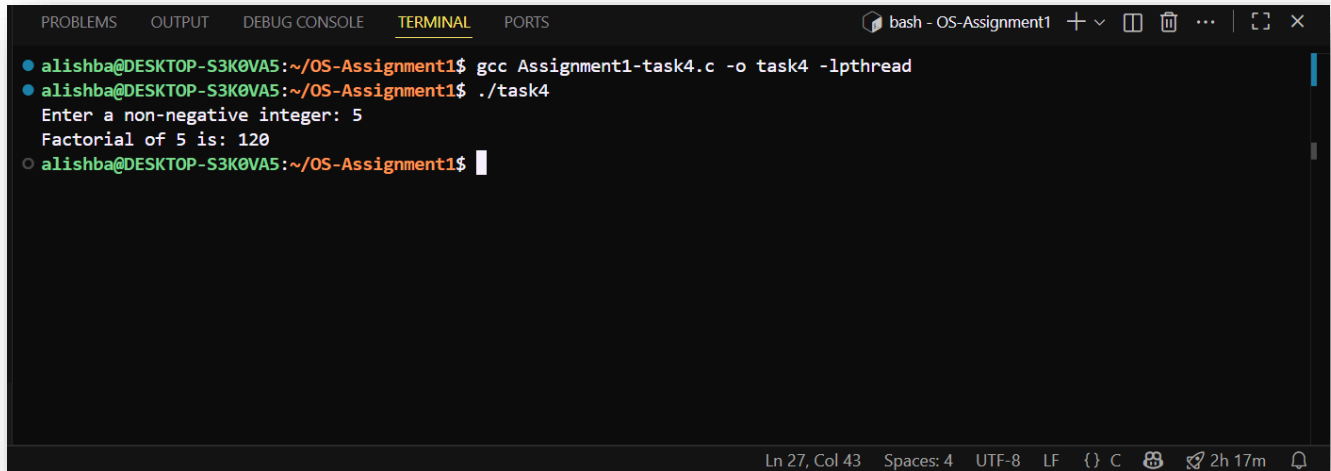
Write a program that creates a thread to compute the factorial of a number entered by the user.

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

Code:

```
1  /*
2  Name: Alishba Riasat
3  Reg No: 23-NTU-CS-1135
4  Task 4: Thread Return Values
5  */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <pthread.h>
10
11 void* compute_factorial (void* arg) {
12     int n = *(int*)arg;
13     long long*result = malloc(sizeof(long long));
14     if (!result) pthread_exit(NULL);
15     *result = 1;
16     for (int i=2; i<=n; i++) {
17         *result= (*result) * i;
18     }
19     // return pointer to heap memory
20     return (void*)result;
21 }
22
23 int main(){
24     pthread_t tid;
25     int number;
26
27     printf("Enter a non-negative integer: ");
28     if (scanf("%d", &number) !=1 || number <0) {
29         printf("Invalid Input. \n");
30         return 1;
31     }
32
33     pthread_create(&tid, NULL, compute_factorial, &number);
34     void* ret;
35     pthread_join(tid, &ret);
36     if (ret != NULL) {
37         long long* factorial = (long long*)ret;
38         printf("Factorial of %d is: %lld\n", number, *factorial);
39         free (factorial);
40     }
41     else {
42         printf("Thread returned NULL.\n");
43     }
44     return 0;
45 }
```

Output:



```
bash - OS-Assignment1
alishba@DESKTOP-S3K0VA5:~/OS-Assignment1$ gcc Assignment1-task4.c -o task4 -lpthread
alishba@DESKTOP-S3K0VA5:~/OS-Assignment1$ ./task4
Enter a non-negative integer: 5
Factorial of 5 is: 120
alishba@DESKTOP-S3K0VA5:~/OS-Assignment1$
```

▪ 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 ($\text{GPA} \geq 3.5$).
- The main thread counts how many students made the Dean's List.

Code:

```
1  /*
2  Name: Alishba Riasat
3  Reg No: 23-NTU-CS-1135
4  Task 5: Struct-Based Thread Communication
5  */
6
7  #include <stdio.h>
8  #include <pthread.h>
9  #include <string.h>
10
11  typedef struct {
12      int student_id;
13      char name[50];
14      float gpa;
15  } Student;
16
17  typedef struct {
18      Student s;
19      int index;
20  } ThreadArg;
21
22  void* student_worker(void* arg) {
23      ThreadArg* ta = (ThreadArg*)arg;
24      Student st = ta->s;
25      printf("Thread %d: Student ID: %d, Name: %s, GPA: %.2f\n", ta->index, st.student_id, st.name, st.gpa);
26      if (st.gpa >= 3.5f) {
27          printf("Thread %d: %s made the Dean's List.\n", ta->index, st.name);
28      } else {
29          printf("Thread %d: %s did NOT make the Dean's List.\n", ta->index, st.name);
30      }
31      return NULL;
32  }
33
34  int main() {
35      pthread_t tid[3];
36      ThreadArg args[3];
37
38      args[0].s.student_id= 101; strcpy(args[0].s.name, "Alishba");
39      args[0].s.gpa= 3.8f; args[0].index= 1;
40
41      args[1].s.student_id= 102; strcpy(args[1].s.name, "Zara");
42      args[1].s.gpa= 3.3f; args[1].index= 2;
43
44      args[2].s.student_id= 103; strcpy(args[2].s.name, "Fatima");
45      args[2].s.gpa= 3.6f; args[2].index= 3;
46
47      // Create threads
48      for (int i = 0; i < 3; i++) {
49          pthread_create(&tid[i], NULL, student_worker, &args[i]);
50      }
51
52      // Wait for threads
53      for (int i = 0; i < 3; i++) {
54          pthread_join(tid[i], NULL);
55      }
56
57      // Count Dean's List
58      int count = 0;
59      for (int i = 0; i < 3; i++) if (args[i].s.gpa >= 3.5f) count++;
60
61      printf("Main: Total students on Dean's List = %d out of 3\n", count);
62
63      return 0;
64  }
65
```

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

```
bash - OS-Assignment1 + ▢ ▢ ... | [ ] x
```

```
● alishba@DESKTOP-S3K0VA5:~/OS-Assignment1$ gcc Assignment1-task5.c -o task5 -lpthread
● alishba@DESKTOP-S3K0VA5:~/OS-Assignment1$ ./task5
Thread 1: Student ID: 101, Name: Alishba, GPA: 3.80
Thread 1: Alishba made the Dean's List.
Thread 2: Student ID: 102, Name: Zara, GPA: 3.30
Thread 2: Zara did NOT make the Dean's List.
Thread 3: Student ID: 103, Name: Fatima, GPA: 3.60
Thread 3: Fatima made the Dean's List.
Main: Total students on Dean's List = 2 out of 3
○ alishba@DESKTOP-S3K0VA5:~/OS-Assignment1$
```

```
Ln 65, Col 1 (1555 selected) Spaces: 4 UTF-8 LF {} C 2h 31m
```

Section-B: Short Questions

1. Define an Operating System in a single line.

- An operating system is software that manages a computer's hardware and software resources, acting as an intermediary between the user and hardware, while ensuring that all applications run efficiently.
- Examples include Windows, Linux, macOS, and Android.

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

The CPU scheduler decides which process in the ready queue should receive CPU time next, aiming to maximize CPU utilization, ensure fairness, and improve the efficiency of process execution.

3. List any three states of a process.

A process can exist in several states, such as:

- **New** (created but not ready)
- **Ready** (waiting for CPU)
- **Running** (currently executing)
- **Waiting/Blocked** (waiting for I/O)
- **Terminated** (completed execution).

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

- A PCB is a data structure maintained by the operating system that stores all relevant information about a process, including its:
 - Process ID
 - Current state
 - CPU registers
 - Memory allocation
 - Scheduling information.
- It allows efficient process management and context switching.

5. Differentiate between a process and a program.

<i>Feature</i>	<i>Process</i>	<i>Program</i>
<i>Definition</i>	A dynamic instance of a program in execution with its own memory and resources.	A static set of instructions stored on disk; not executing.
<i>State</i>	Has a current state (New, Ready, Running, Waiting, Terminated).	Does not have a state; inactive until executed.
<i>Resources</i>	Allocated system resources like CPU, memory, I/O devices.	No resources allocated; just code stored on storage.
<i>Execution</i>	Active; runs and performs tasks.	Inactive; passive until loaded into memory and executed.
<i>Instance</i>	Multiple processes can be created from the same program.	Only one program exists; multiple processes can run from it.

6. What do you understand by context switching?

Context switching is the process by which the operating system **saves the state of a running process** and loads the state of another process, allowing multiple processes to share the CPU effectively. It is essential for multitasking and ensures that no process monopolizes the CPU.

7. Define CPU utilization and throughput.

- **CPU utilization** is the percentage of time the CPU is actively processing instructions rather than being idle.
 - **Formula:** $\text{CPU Utilization (\%)} = (\text{CPU Busy Time} / \text{Total Time}) \times 100$
- **Throughput** refers to the number of processes completed per unit of time, which is an indicator of the system's overall efficiency.
 - **Formula:** $\text{Throughput} = \text{Number of processes completed} / \text{Total time}$

8. What is the turnaround time of a process?

- Turnaround time is the total duration from the moment a process is submitted to the system until it is fully completed.
- This includes **waiting time, execution time, and I/O time**, reflecting the overall responsiveness of the system.
- **Formula:**

$$\text{Turnaround Time (TR)} = \text{Completion Time} - \text{Arrival Time}$$

9. How is waiting time calculated in process scheduling?

Waiting time is the total time a process spends in the **ready queue waiting for CPU access**, excluding the time spent executing on the CPU or performing I/O operations. It is used to evaluate the efficiency of scheduling algorithms.

Formula:

$$\text{Waiting Time (W)} = \text{Turnaround Time (TR)} - \text{CPU Burst Time (TS)}$$

10. Define response time in CPU scheduling.

Response time is the time interval between submitting a request or process and the first time the CPU responds to it, crucial for interactive systems.

Formula:

$$\text{Response Time (R)} = \text{Time of first CPU allocation} - \text{Arrival Time}$$

11. What is preemptive scheduling?

Preemptive scheduling allows the OS to interrupt a currently running process and assign the CPU to another process, usually based on priority or time-slicing, ensuring fairness and responsiveness.

12. What is non-preemptive scheduling?

Non-preemptive scheduling means once a process starts executing, it runs to completion or until it voluntarily waits for I/O, with no forced interruptions from the OS.

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

- The Round Robin (RR) scheduling algorithm provides **fair and cyclic CPU allocation**, ensuring that no process is starved and every process gets a chance to execute.
- It also offers **better response time** for interactive processes, making the system more responsive to user input.

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

The main drawback of SJF is **starvation**, where longer processes may wait indefinitely if short processes keep arriving, as shorter CPU bursts are always prioritized.

15. Define CPU idle time.

CPU idle time is the period when the CPU is **not executing any process**, often occurring when the ready queue is empty or all processes are waiting for I/O operations to complete.

16. State two common goals of CPU scheduling algorithms.

The primary goals of CPU scheduling algorithms are:

1. **Maximize CPU utilization and throughput** while minimizing waiting and turnaround times.
2. **Ensure fairness**, so that no process is unfairly delayed or starved.

17. List two possible reasons for process termination.

A process can terminate due to the following reasons:

1. **Normal Termination:** The process completes its execution successfully.
2. **Abnormal Termination:** The process is terminated due to errors, resource limitations, or external signals from the operating system or user.

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

- The **exit()** system call terminates a process and passes its completion status to the parent.
- The **wait()** system call allows the parent process to **pause execution until its child process finishes**, ensuring proper collection of exit status and preventing zombie processes.

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

<i>Feature</i>	<i>Shared Memory</i>	<i>Message Passing</i>
<i>Communication</i>	Processes communicate via a common memory area.	Processes communicate by sending and receiving messages.
<i>Speed</i>	Faster, as no system calls are needed for data transfer.	Slower due to copying messages between processes.
<i>Synchronization</i>	Requires explicit synchronization (e.g., semaphores).	Synchronization handled implicitly by the message system.
<i>Complexity</i>	More complex to implement due to concurrency control.	Easier to implement; safer for inter-process communication.

20. Differentiate between a thread and a process.

Feature	Process	Thread
Definition	Independent execution unit with its own memory and resources.	Lightweight execution unit within a process sharing memory and resources.
Memory	Has separate memory space.	Shares memory space with other threads of the same process.
Resource Usage	Higher overhead for creation and context switching.	Lower overhead; faster to create and switch.
Execution	Executes independently.	Executes concurrently with other threads in the same process.
Communication	Requires inter-process communication to share data.	Can communicate directly through shared memory of the process.

21. Define multithreading.

Multithreading is the capability of a process to run **multiple threads simultaneously**. It allows a single process to perform multiple tasks concurrently, improving CPU utilization, responsiveness, and overall efficiency, especially in interactive and high-performance applications.

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

Feature	CPU-bound Process	I/O-bound Process
Definition	A process that spends most of its time performing computations and requires extensive CPU time.	A process that spends most of its time waiting for input/output operations rather than using the CPU.
CPU Usage	High, as the process frequently utilizes the CPU for computations.	Low, as the process often waits for I/O operations to complete.
I/O Usage	Minimal, rarely waits for input/output devices.	High, frequently performs I/O operations and spends more time waiting.
Scheduling Impact	Prioritized in CPU scheduling to optimize CPU utilization.	May be scheduled to maximize I/O device utilization and minimize waiting.

23. What are the main responsibilities of the dispatcher?

- The dispatcher is a part of the operating system that **loads the selected process onto the CPU**, performs context switching, and starts execution.
- It handles process transitions, manages CPU control, and ensures that switching between processes is fast and efficient to minimize overhead.

24. Define starvation and aging in process scheduling.

- **Starvation** occurs when a low-priority process waits indefinitely because higher-priority processes keep taking the CPU.
- **Aging** is a technique to gradually increase the priority of waiting processes, ensuring that all processes eventually get CPU time and preventing starvation.

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

A time quantum is a fixed amount of CPU time allocated to each process in **preemptive or Round Robin scheduling**. When the quantum expires, the CPU may switch to the next process, ensuring fairness and responsiveness in a multitasking environment.

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

- If the **time quantum is too large**, scheduling behaves like First-Come-First-Serve (FCFS), reducing responsiveness for interactive processes.
- If the **time quantum is too small**, context switching occurs too frequently, increasing overhead and reducing CPU efficiency.

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

The **turnaround ratio** is a measure of the efficiency of process execution, defined as the ratio of a process's **turnaround time (TR)** to its **CPU burst time (TS)**.

Formula:

Turnaround Ratio (TR/TS) = Turnaround Time (TR) / CPU Burst Time (TS)

It indicates how much longer a process takes to complete compared to its actual CPU execution time. A **higher ratio** may suggest delays due to waiting or scheduling, while a **lower ratio** indicates efficient execution.

28. What is the purpose of a ready queue?

The ready queue is a data structure maintained by the operating system to store all **processes that are ready to execute** but waiting for CPU allocation. It ensures orderly scheduling and helps the CPU scheduler select the next process efficiently.

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

<i>Feature</i>	<i>CPU Burst</i>	<i>I/O Burst</i>
Definition	A period during which a process executes instructions on the CPU performing computations.	A period during which a process waits for or performs input/output operations.
Resource Usage	Uses CPU extensively during execution.	Uses I/O devices; CPU remains idle or executes other processes.
Duration	Typically, shorter than I/O bursts in interactive systems.	Can be longer, depending on device speed and data transfer.
Scheduling Impact	Important for CPU scheduling decisions; affects turnaround and waiting time.	Important for I/O scheduling; affects CPU idle time and throughput.

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

The **Round Robin (RR) scheduling algorithm** is starvation-free because it allocates CPU time cyclically to all processes using fixed time slices. This ensures that every process eventually receives CPU access, regardless of priority or arrival order.

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

- The parent calls `fork()` to create a child process, which is a duplicate of the parent.
- The child can optionally use `exec()` to run a new program.
- The parent can call `wait()` to pause and collect the child's exit status.
- Both processes get resources (some shared, some private) and may run concurrently.

32. Define zombie and orphan processes.

- A **zombie process** is one that has completed execution but still has an entry in the process table because its parent has not collected its exit status.
- An **orphan process** is a child whose parent terminates before it does; the **init process** typically adopts it to ensure proper cleanup.

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

<i>Feature</i>	<i>Priority Scheduling</i>	<i>Shortest Job First (SJF)</i>
<i>Basis of Selection</i>	Selects the process with the highest priority for execution.	Selects the process with the shortest CPU burst time.
<i>Starvation Potential</i>	High, as low-priority processes may wait indefinitely.	High, as longer processes may be continually postponed.
<i>Preemption</i>	Can be preemptive or non-preemptive.	Can be preemptive (Shortest Remaining Time First) or non-preemptive.
<i>Scheduling Focus</i>	Ensures important tasks get CPU quickly based on priority.	Minimizes average waiting and turnaround times by prioritizing short jobs.

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

- **Context switch time** is the duration required to **save the state of the current process** and **load the state of the next process**.
- It is considered overhead because the CPU is not performing productive computation during this time; it only manages process transitions.

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

- 1) **Long-term scheduler:** Controls the admission of processes into the system from the job pool, balancing CPU and I/O usage.
- 2) **Medium-term scheduler:** Temporarily suspends (swaps out) or resumes processes to optimize performance and resource utilization.
- 3) **Short-term scheduler:** Selects the next process from the ready queue to execute on the CPU, managing fast context switches for multitasking.

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

Feature	User Mode	Kernel Mode
Definition	The CPU runs application code with restricted access to hardware and system resources.	The CPU executes operating system code with full access to all hardware and system resources.
Access Level	Limited; cannot execute privileged instructions or directly access hardware.	Full; can execute privileged instructions and manage system resources.
Purpose	Ensures safety and stability by isolating user applications from the OS.	Handles critical tasks such as process scheduling, memory management, and I/O operations.
Error Impact	Errors affect only the application; the OS remains safe.	Errors can affect the entire system, potentially causing crashes.

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:

A **process** is an instance of a program in execution. It goes through several states from creation to termination, and these transitions are managed by the operating system.

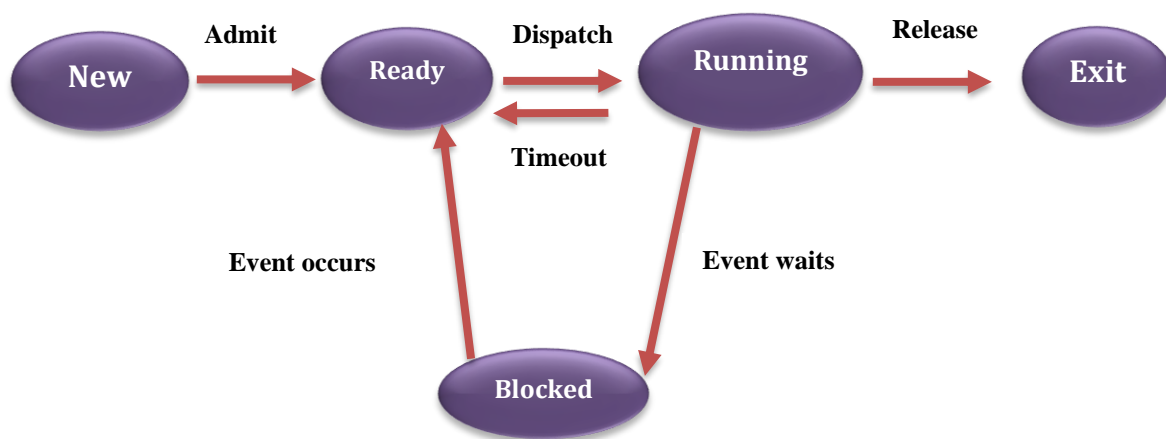
States:

- **New:** Process is being created; resources are being allocated.
- **Ready:** Process is loaded in main memory and waiting for CPU scheduling.
- **Running:** CPU is executing the process instructions.
- **Waiting (Blocked):** Process cannot proceed until some I/O or event occurs.
- **Terminated:** Process has finished execution and is removed from memory.

Transitions:

- **New → Ready:** Process has been admitted to memory.
- **Ready → Running:** CPU scheduler selects the process.
- **Running → Waiting:** Process requires I/O or waits for an event.
- **Running → Ready:** Preemption occurs due to time-slice expiry or higher-priority process.
- **Waiting → Ready:** I/O or event is complete.
- **Running → Terminated:** Process completes execution.

Diagram:



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 executing one process to another. While necessary for multitasking, context switching is not free and introduces overhead, which is:

"The extra time taken to save the state of the current process and load the state of the next process."

This overhead reduces CPU efficiency because the CPU spends some time switching rather than executing actual process instructions.

Information Saved and Restored:

During a context switch, the operating system must save the state of the currently running process and restore the state of the next process.

The information typically includes:

- **CPU registers:** Program counter (PC), stack pointer (SP), general-purpose registers, and status registers.
- **Process control block (PCB):** Contains process ID, process state, priority, and scheduling information.
- **Memory management information:** Page tables or segment tables.
- **I/O status information:** Open files, pending I/O operations.

After saving this information, the OS loads the saved state of the next process so it can resume execution **exactly where it left off**.

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

Process Control Block (PCB)

A Process Control Block (PCB) is a data structure maintained by the operating system to store all the information about a process. It is used for process management, scheduling, and context switching.

Components of a PCB

1. Process Identification Information

- **Process ID (PID):** Unique identifier for the process.
- **Parent Process ID (PPID):** ID of the process that created this process.
- **User/Group ID:** Identifies the owner of the process.

2. Process State

- Current state of the process: **New, Ready, Running, Waiting, or Terminated.**

3. CPU Registers

- Stores the contents of CPU registers when the process is not running.
- Includes **Program Counter (PC), Stack Pointer (SP), general-purpose registers, and status registers.**

4. CPU Scheduling Information

- **Priority**, pointers to scheduling queues, and other parameters used by the CPU scheduler.

5. Memory Management Information

- Base and limit registers, **page tables**, segment tables, or other memory-related data required for accessing the process's address space.

6. Accounting Information

- CPU usage, execution time, time limits, job numbers, and other accounting details for monitoring or billing.

7. I/O Status Information

- List of **open files**, allocated I/O devices, and **pending I/O operations.**

Importance of PCB

- Enables the operating system to **manage multiple processes efficiently.**
 - Supports **context switching** by saving and restoring process states.
 - Allows the OS to **schedule, track, and control processes systematically.**
-

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

<i>Scheduler</i>	<i>Function / Role</i>	<i>Frequency of Execution</i>	<i>Decision Basis</i>	<i>Example</i>
<i>Long-Term Scheduler</i>	Selects processes from the job pool (secondary memory) and loads them into main memory for execution.	Infrequent (seconds to minutes)	Decides which jobs are admitted to memory based on priority, type, and resources.	Job scheduler in batch systems
<i>Medium-Term Scheduler</i>	Temporarily suspends (swaps out) and later resumes (swaps in) processes to improve CPU utilization and balance CPU/I/O.	Occasional (minutes)	Balances CPU-bound and I/O-bound processes.	Swapping scheduler in multiprogramming OS
<i>Short-Term Scheduler</i>	Selects which process in the ready queue gets the CPU next for execution.	Very frequent (milliseconds)	Based on CPU scheduling algorithms (FCFS, SJF, Round Robin, Priority).	CPU scheduler in time-sharing systems

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

CPU scheduling is the process by which the operating system decides **which process in the ready queue gets the CPU next**. The goal of CPU scheduling is to **improve system performance** by making efficient use of the CPU and minimizing delays for processes. The performance of scheduling algorithms is measured using the following criteria:

1. CPU Utilization

- **Definition:** The percentage of time the CPU is actively executing processes rather than being idle.
- **Importance:** A higher CPU utilization indicates better use of resources.
- **Goal:** Maximize CPU utilization so that the CPU remains busy most of the time.

2. Throughput

- **Definition:** The number of processes completed per unit of time.
- **Importance:** Measures how efficiently the system is processing work.
- **Goal:** Maximize throughput, meaning more processes are completed in a given period.
- **Example:** If 10 processes finish in 5 seconds, throughput = 2 processes/sec.

3. Turnaround Time

- **Definition:** The total time taken by a process from **submission to completion**.
- **Importance:** Indicates how quickly a process completes execution.
- **Goal:** Minimize average turnaround time so that processes complete faster.
- **Example:** A process submitted at 0 sec and finished at 10 sec has a turnaround time of 10 sec.

4. Waiting Time

- **Definition:** The total time a process spends **waiting in the ready queue** before it gets the CPU.
- **Importance:** Lower waiting time improves responsiveness and fairness among processes.
- **Goal:** Minimize average waiting time to reduce unnecessary delays.
- **Example:** If a process waits 4 seconds in the ready queue before execution, waiting time = 4 sec.

5. Response Time

- **Definition:** The time from **submission of a request** until the **first response is produced**, not necessarily the completion of the process.
- **Importance:** Critical in **interactive or time-sharing systems**, as users expect quick feedback.

- **Goal:** Minimize response time to improve system interactivity and user experience.
- **Example:** In a text editor, the time between pressing a key and the character appearing on the screen is response time.

Summary of Optimization Goals:

- Maximize CPU utilization and throughput for better overall system performance.
- Minimize turnaround time, waiting time, and response time to make the system faster, fair, and responsive to user needs.

Section-D: CPU Scheduling Calculations

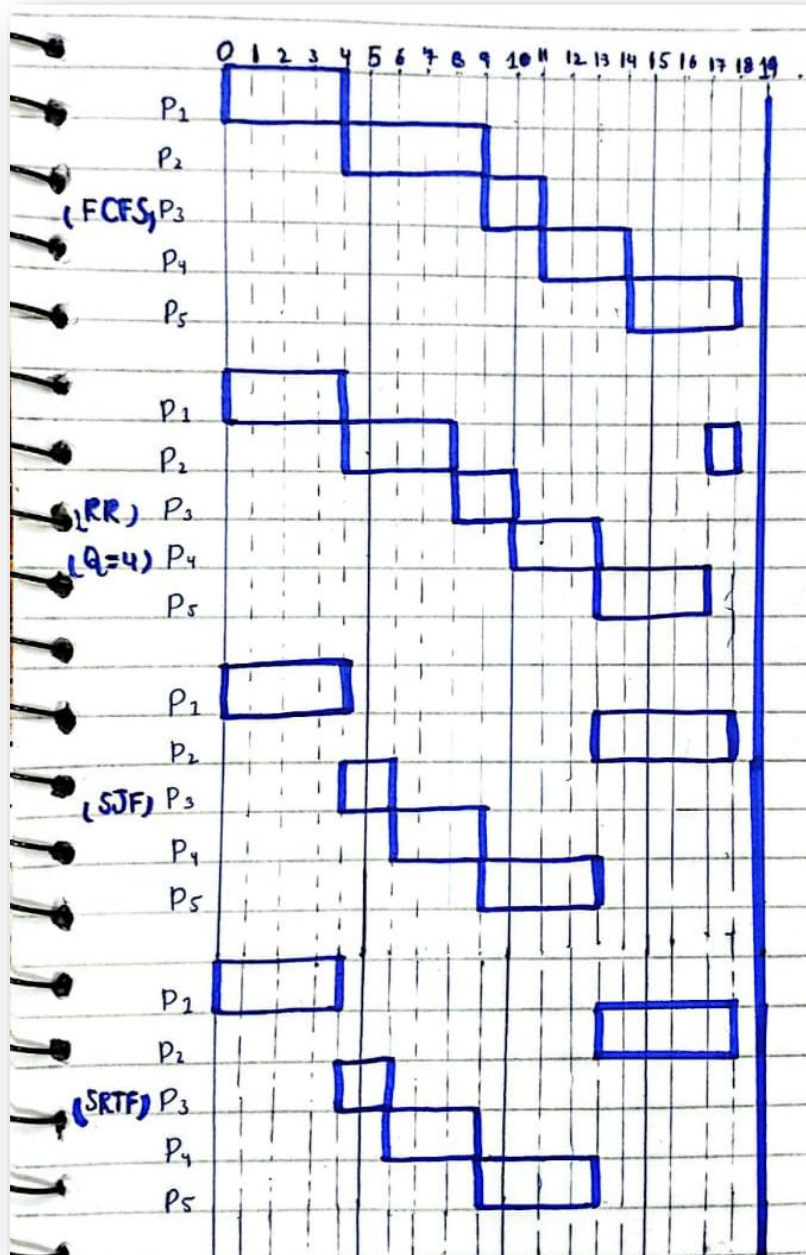
Perform the following calculations for each part (A–C).

- 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

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

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



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

1. FCFS

<i>Process</i>	<i>Arrival Time</i>	<i>Waiting Time</i>	<i>Service Time (Ts)</i>	<i>Finish Time</i>	<i>Turnaround Time (Tr)</i>	<i>TR/TS ratio</i>
P1	0	0	4	4	4	1
P2	2	2	5	9	7	1.4
P3	4	5	2	11	7	3.5
P4	6	5	3	14	8	2.6667
P5	9	5	4	18	9	2.25

- CPU idle time is 0.

2. Round Robin (q=4)

<i>Process</i>	<i>Arrival Time</i>	<i>Waiting Time</i>	<i>Service Time (Ts)</i>	<i>Finish Time</i>	<i>Turnaround Time (Tr)</i>	<i>TR/TS ratio</i>
P1	0	0	4	4	4	1
P2	2	11	5	18	16	3.2
P3	4	4	2	10	6	3
P4	6	4	3	13	7	2.333
P5	9	4	4	17	8	4

- CPU idle time is 0.

3. SJF

<i>Process</i>	<i>Arrival Time</i>	<i>Waiting Time</i>	<i>Service Time (Ts)</i>	<i>Finish Time</i>	<i>Turnaround Time (Tr)</i>	<i>TR/TS ratio</i>
P1	0	0	4	4	4	1
P2	2	11	5	18	16	3.2
P3	4	0	2	6	2	1
P4	6	0	3	9	3	1
P5	9	0	4	13	4	1

- CPU idle time is 0.

4. SRTF

<i>Process</i>	<i>Arrival Time</i>	<i>Waiting Time</i>	<i>Service Time (Ts)</i>	<i>Finish Time</i>	<i>Turnaround Time (Tr)</i>	<i>TR/TS ratio</i>
P1	0	0	4	4	4	1
P2	2	11	5	18	16	3.2
P3	4	0	2	6	2	1
P4	6	0	3	9	3	1
P5	9	0	4	13	4	1

- CPU idle time is 0.

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

<i>Algorithm</i>	<i>Average waiting time</i>	<i>Average Turnaround time</i>	<i>Average Tr/Ts ratio</i>
FCFS	3.4	7	2.1633
Round Robin (q=4)	4.6	8.2	2.706
SJF	2.2	5.8	1.44
SRTF	2.2	5.8	1.44

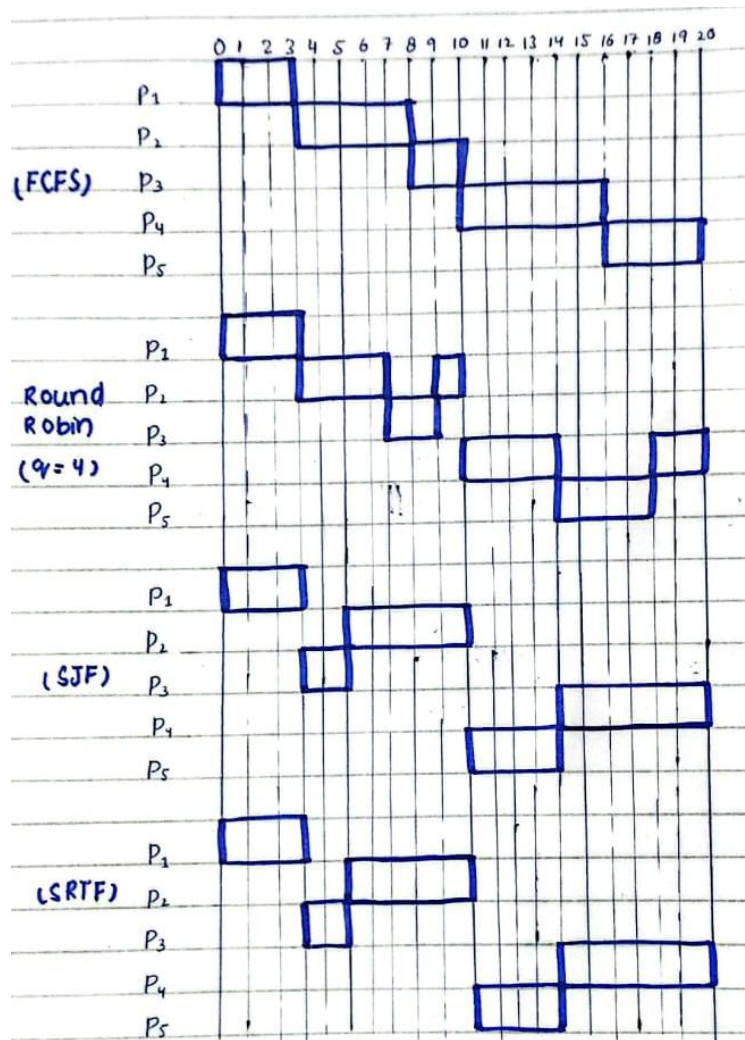
Best Algorithms:

- SJF and SRTF perform best for this process set as they give the lowest average waiting time (2.2), lowest average turnaround time (5.8) and lowest average Tr/Ts ratio (1.44).
- RR (q=4) shows higher averages here because of time slicing and FCFS is in the middle.

Part-B

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

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



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

1. FCFS

<i>Process</i>	<i>Arrival Time</i>	<i>Waiting Time</i>	<i>Service Time (Ts)</i>	<i>Finish Time</i>	<i>Turnaround Time (Tr)</i>	<i>TR/TS ratio</i>
P1	0	0	3	3	3	1
P2	1	2	5	8	7	1.4
P3	3	5	2	10	7	3.5
P4	9	1	6	16	7	1.667
P5	10	6	4	20	10	2.5

- CPU idle time is 0.

2. Round Robin (q=4)

<i>Process</i>	<i>Arrival Time</i>	<i>Waiting Time</i>	<i>Service Time (Ts)</i>	<i>Finish Time</i>	<i>Turnaround Time (Tr)</i>	<i>TR/TS ratio</i>
P1	0	0	3	3	3	1
P2	1	4	5	10	9	1.8
P3	3	4	2	9	6	3
P4	9	5	6	20	11	1.8333
P5	10	4	4	18	8	2

- CPU idle time is 0.

3. SJF

<i>Process</i>	<i>Arrival Time</i>	<i>Waiting Time</i>	<i>Service Time (Ts)</i>	<i>Finish Time</i>	<i>Turnaround Time (Tr)</i>	<i>TR/TS ratio</i>
P1	0	0	3	3	3	1
P2	1	4	5	10	9	1.8
P3	3	0	2	5	2	1
P4	9	5	6	20	11	1.8333
P5	10	0	4	14	4	1

- CPU idle time is 0.

4. SRTF

<i>Process</i>	<i>Arrival Time</i>	<i>Waiting Time</i>	<i>Service Time (Ts)</i>	<i>Finish Time</i>	<i>Turnaround Time (Tr)</i>	<i>TR/TS ratio</i>
P1	0	0	3	3	3	1
P2	1	4	5	10	9	1.8
P3	3	0	2	5	2	1
P4	9	5	6	20	11	1.8333
P5	10	0	4	14	4	1

- CPU idle time is 0.

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

<i>Algorithm</i>	<i>Average waiting time</i>	<i>Average Turnaround time</i>	<i>Average Tr/Ts ratio</i>
FCFS	2.8	6.8	2.0134
Round Robin (q=4)	3.4	7.4	1.92666
SJF	1.8	5.8	1.32666
SRTF	1.8	5.8	1.32666

Best Algorithms:

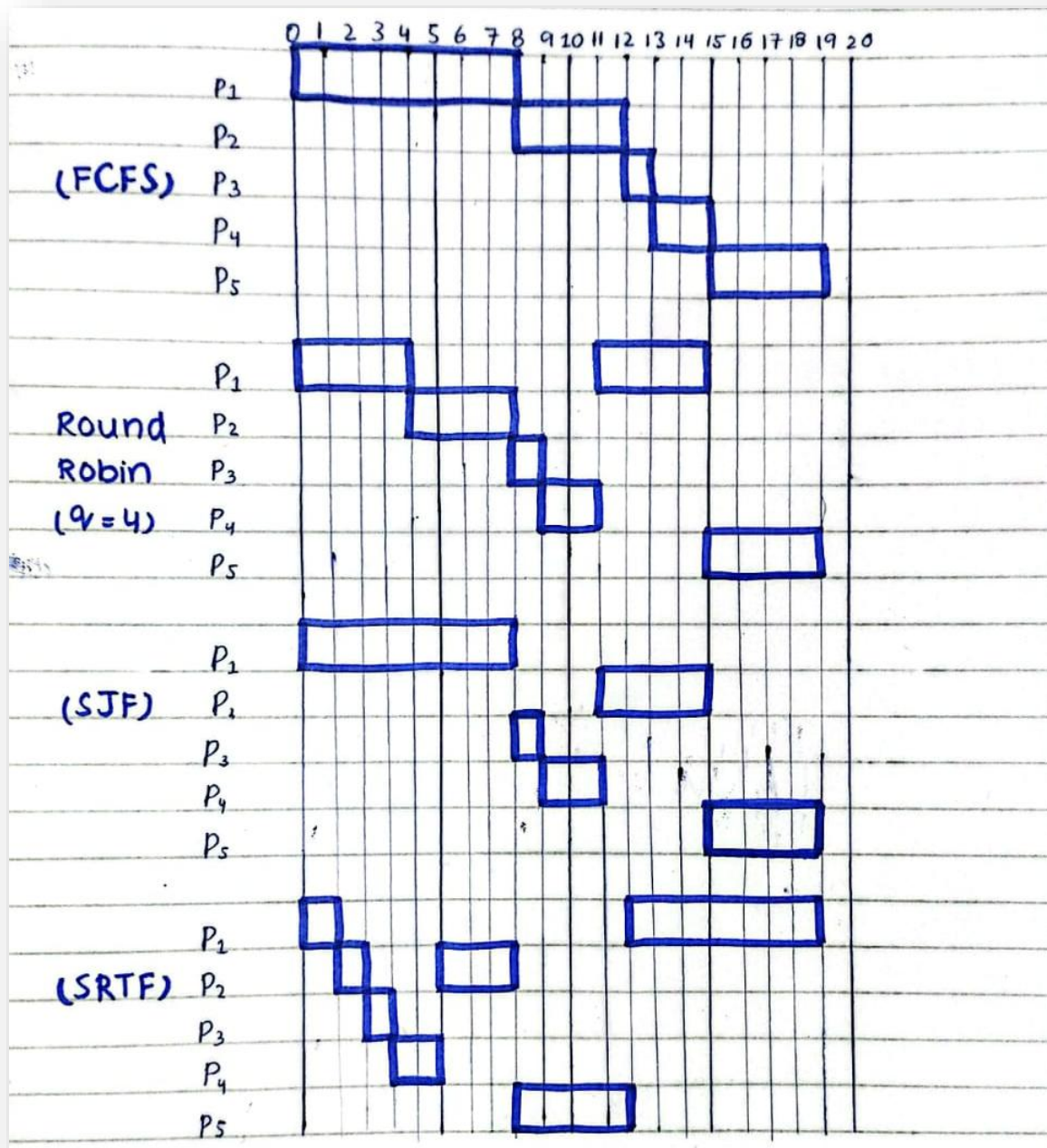
- SJF and SRTF perform best for this process set as they give the lowest average waiting time (1.8), lowest average turnaround time (5.8) and lowest average Tr/Ts ratio (1.32666).
- RR (q=4) shows higher averages here because of time slicing and FCFS is in the middle.

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

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

PROCESS	ARRIVAL TIME	SERVICE TIME
P1	0	8
P2	1	4
P3	2	1
P4	3	2
P5	6	4

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



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

1. FCFS

<i>Process</i>	<i>Arrival Time</i>	<i>Waiting Time</i>	<i>Service Time (Ts)</i>	<i>Finish Time</i>	<i>Turnaround Time (Tr)</i>	<i>TR/TS ratio</i>
P1	0	0	8	8	8	1
P2	1	7	4	12	11	2.75
P3	2	10	1	13	11	11
P4	3	10	2	15	12	6
P5	6	9	4	19	13	3.25

- CPU idle time is 0.

2. Round Robin (q=4)

<i>Process</i>	<i>Arrival Time</i>	<i>Waiting Time</i>	<i>Service Time (Ts)</i>	<i>Finish Time</i>	<i>Turnaround Time (Tr)</i>	<i>TR/TS ratio</i>
P1	0	7	8	15	15	1.875
P2	1	3	4	8	7	1.75
P3	2	6	1	9	7	7
P4	3	6	2	11	8	4
P5	6	9	4	19	13	3.25

- CPU idle time is 0.

3. SJF

<i>Process</i>	<i>Arrival Time</i>	<i>Waiting Time</i>	<i>Service Time (Ts)</i>	<i>Finish Time</i>	<i>Turnaround Time (Tr)</i>	<i>TR/TS ratio</i>
P1	0	0	8	8	8	1
P2	1	10	4	15	14	3.5
P3	2	6	1	9	7	7
P4	3	6	2	11	8	4
P5	6	9	4	19	13	3.25

- CPU idle time is 0.

4. SRTF

<i>Process</i>	<i>Arrival Time</i>	<i>Waiting Time</i>	<i>Service Time (Ts)</i>	<i>Finish Time</i>	<i>Turnaround Time (Tr)</i>	<i>TR/TS ratio</i>
P1	0	11	8	19	19	2.375
P2	1	3	4	8	7	1.75
P3	2	0	1	3	1	1
P4	3	0	2	5	2	1
P5	6	2	4	12	6	1.50

- CPU idle time is 0.

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

<i>Algorithm</i>	<i>Average waiting time</i>	<i>Average Turnaround time</i>	<i>Average Tr/Ts ratio</i>
FCFS	7.2	11	4.8
Round Robin (q=4)	6.2	10	3.575
SJF	6.2	10	3.575
SRTF	3.2	7	1.525

Best Algorithms:

- SRTF perform best for this chosen process set as it gives the lowest average waiting time (3.2), lowest average turnaround time (7) and lowest average Tr/Ts ratio (1.525) because it always runs the process with shortest remaining time, which reduces waiting for short jobs that arrive while long jobs are running.
- SJF and RR tie here for the middle position while FCFS performs worst for this process set.