



National Textile University
Department of Computer Science

Subject
Operating System
Submitted to:

Sir Nasir Mehmood

Submitted by:

Kashmir Jamshaid

Registration Number
23-NTU-CS-1167

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:

```

1 // Kashmir Jmashaid
2 // 23-NTU-CS-1167
3 // TASK-1
4 // Write a program that creates 5 threads. Each thread should:
5 // - Print its thread ID using `pthread_self()`.
6 // - Display its thread number (1st, 2nd, etc.).
7 // - Sleep for a random time between 1-3 seconds.
8 // - Print a completion message before exiting.
9
10 #include <stdio.h>
11 #include <pthread.h>
12 #include <unistd.h>
13 #include <stdlib.h> // For srand(),time() and also rand()
14 // Thread function
15 void* thread_function(void* arg) {
16     int thread_num = *(int*)arg; // Convert the passed argument to an integer (thread number)
17
18     printf("Thread No: %d\n", thread_num);
19     printf("Thread %d ID: %lu\n", thread_num, pthread_self());
20
21     int sleep_time = (rand() % 3) + 1; // Sleep for random time between 1-3 seconds
22     printf("Thread %d sleeping for %d seconds...\n", thread_num, sleep_time);
23     sleep(sleep_time);
24
25     printf("Thread %d completed!\n\n", thread_num);
26     return NULL;
27 }
28
29 int main() {
30     pthread_t threads[5];
31     int thread_nums[5];
32
33     printf("Main thread starting...\n");
34     printf("Main Thread ID: %lu\n\n", pthread_self());
35
36     srand(time(NULL)); // Seed for random sleep times
37
38     // Create 5 threads
39     for (int i = 0; i < 5; i++) {
40         thread_nums[i] = i + 1;
41         pthread_create(&threads[i], NULL, thread_function, &thread_nums[i]);
42     }
43
44     // Wait for all threads to finish
45     for (int i = 0; i < 5; i++) {
46         pthread_join(threads[i], NULL);
47     }
48
49     printf("All threads completed. Main thread exiting...\n");
50     return 0;
51 }
52

```

Output:

```
OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
bash - Assignment#1

kashmirkj@Kashmirpc:~/Operating_System$ ls
kashmirkj@Kashmirpc:~/Operating_System$ cd Assignment#1
kashmirkj@Kashmirpc:~/Operating_System/Assignment#1$ gcc Task1.c -o Task1 -lpthread
kashmirkj@Kashmirpc:~/Operating_System/Assignment#1$ ./Task1
Main thread starting...
Main Thread ID: 134461694027584

Thread No: 1
Thread 1 ID: 134461691786944
Thread 1 sleeping for 3 seconds...
Thread No: 2
Thread 2 ID: 134461683394240
Thread 2 sleeping for 2 seconds...
Thread No: 3
Thread 3 ID: 134461675001536
Thread 3 sleeping for 1 seconds...
Thread No: 4
Thread 4 ID: 134461666608832
Thread 4 sleeping for 2 seconds...
Thread No: 5
Thread 5 ID: 134461658216128
Thread 5 sleeping for 2 seconds...
Thread 3 completed!

Thread 5 completed!

Thread 2 completed!

Thread 4 completed!

Thread 1 completed!

All threads completed. Main thread exiting...
kashmirkj@Kashmirpc:~/Operating_System/Assignment#1$
```

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...

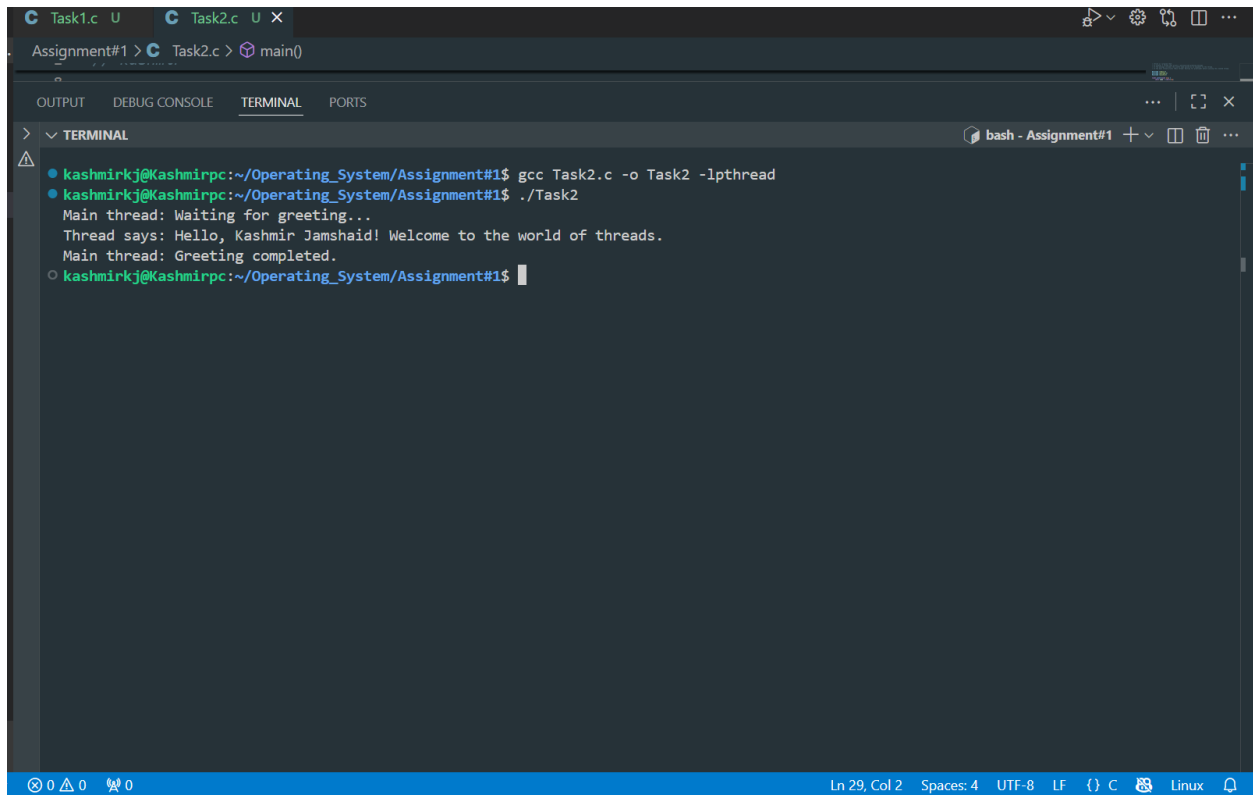
Thread says: Hello, Ali! Welcome to the world of threads.

Main thread: Greeting completed.

Code :

```
1 // Kashmir
2 // 23-NTU-CS-1167
3 // TASK-2
4 // Write a C program that:
5 // * Creates a thread that prints a personalized greeting message.
6 // * The message includes the user's name passed as an argument to the thread.
7 // * The main thread prints "Main thread: Waiting for greeting..." before joining the created thread.
8
9 #include <stdio.h>
10 #include <pthread.h>
11 #include <unistd.h>
12
13 void* greet(void* arg) {
14     char* name = (char*)arg;
15     printf("Thread says: Hello, %s! Welcome to the world of threads.\n", name);
16     pthread_exit(NULL);
17 }
18
19 int main() {
20     pthread_t thread;
21     char* userName = "Kashmir Jamshaid";
22
23     printf("Main thread: Waiting for greeting...\n");
24     pthread_create(&thread, NULL, greet, (void*)userName);
25     pthread_join(thread, NULL);
26     printf("Main thread: Greeting completed.\n");
27
28     return 0;
29 }
30
```

Output:



```
Task1.c U Task2.c U X
Assignment#1 > Task2.c > main()
OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash - Assignment#1
kashmirkj@Kashmirpc:~/Operating_System/Assignment#1$ gcc Task2.c -o Task2 -lpthread
kashmirkj@Kashmirpc:~/Operating_System/Assignment#1$ ./Task2
Main thread: Waiting for greeting...
Thread says: Hello, Kashmir Jamshaid! Welcome to the world of threads.
Main thread: Greeting completed.
kashmirkj@Kashmirpc:~/Operating_System/Assignment#1$
```

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 :

```
1 // Kashmir Jamshaid
2 // 23-NTU-CS-1167
3 // TASK-3
4 // Write a program that:
5 // • Takes an integer input from the user.
6 // • Creates a thread and passes this integer to it.
7 // • The thread prints the number, its square, and cube.
8 // • The main thread waits until completion and prints "Main thread: Work completed."
9
10 #include <stdio.h>
11 #include <pthread.h>
12 #include <unistd.h>
13
14 void* calculate(void* arg) {
15     int num = *(int*)arg;
16     printf("Thread: Number = %d\n", num);
17     printf("Thread: Square = %d\n", num * num);
18     printf("Thread: Cube = %d\n", num * num * num);
19     return NULL;
20 }
21
22 int main() {
23     pthread_t thread_id;
24     int number;
25
26     printf("Enter an integer: ");
27     scanf("%d", &number);
28
29     pthread_create(&thread_id, NULL, calculate, &number);
30     pthread_join(thread_id, NULL);
31
32     printf("Main thread: Work completed.\n");
33     return 0;
34 }
35
```

Output:

```
Assignment#1 > Task3.c > ...
14 void* calculate(void* arg) {
    OUTPUT DEBUG CONSOLE TERMINAL PORTS
    bash - Assignment#1
    kashmirkj@Kashmirpc:~/Operating_System/Assignment#1$ gcc Task3.c -o Task3 -lpthread
    kashmirkj@Kashmirpc:~/Operating_System/Assignment#1$ ./Task3
    Enter an integer: 6
    Thread: Number = 6
    Thread: Square = 36
    Thread: Cube = 216
    Main thread: Work completed.
    kashmirkj@Kashmirpc:~/Operating_System/Assignment#1$
```

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 :


```

1 // Kashmir Jamshaid
2 // 23-NTU-CS-1167
3 // TASK-4
4 // Write a program that creates a thread to compute the factorial of a number entered by the user.
5 // * The thread should return the result using a pointer.
6 // * The main thread prints the result after joining.
7
8 #include <stdio.h>
9 #include <pthread.h>
10 #include <stdlib.h>
11
12 void* factorial(void* arg) {
13     int n = *(int*)arg;
14     long long* result = malloc(sizeof(long long)); // Allocate (give ) memory to store result
15     *result = 1;
16
17     for (int i = 1; i <= n; i++) {
18         *result *= i;
19     }
20
21     return result; // Return the pointer to main thread
22 }
23
24 int main() {
25     pthread_t thread_id;
26     int number;
27     long long* fact_result;
28
29     printf("Enter a number: ");
30     scanf("%d", &number);
31
32     pthread_create(&thread_id, NULL, factorial, &number);
33     pthread_join(thread_id, (void**)&fact_result);
34
35     printf("Factorial of %d = %lld\n", number, *fact_result);
36     printf("Main thread: Work completed.\n");
37
38     free(fact_result); // Free allocated memory
39     return 0;
40 }
41
42

```

Output:

```
24 int main() {  
OUTPUT DEBUG CONSOLE TERMINAL PORTS  
> TERMINAL bash - Assignment#1  
kashmirkj@Kashmirpc:~/Operating_System/Assignment#1$ gcc Task4.c -o Task4 -lpthread  
kashmirkj@Kashmirpc:~/Operating_System/Assignment#1$ ./Task4  
Enter a number: 5  
Factorial of 5 = 120  
Main thread: Work completed.  
kashmirkj@Kashmirpc:~/Operating_System/Assignment#1$
```

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

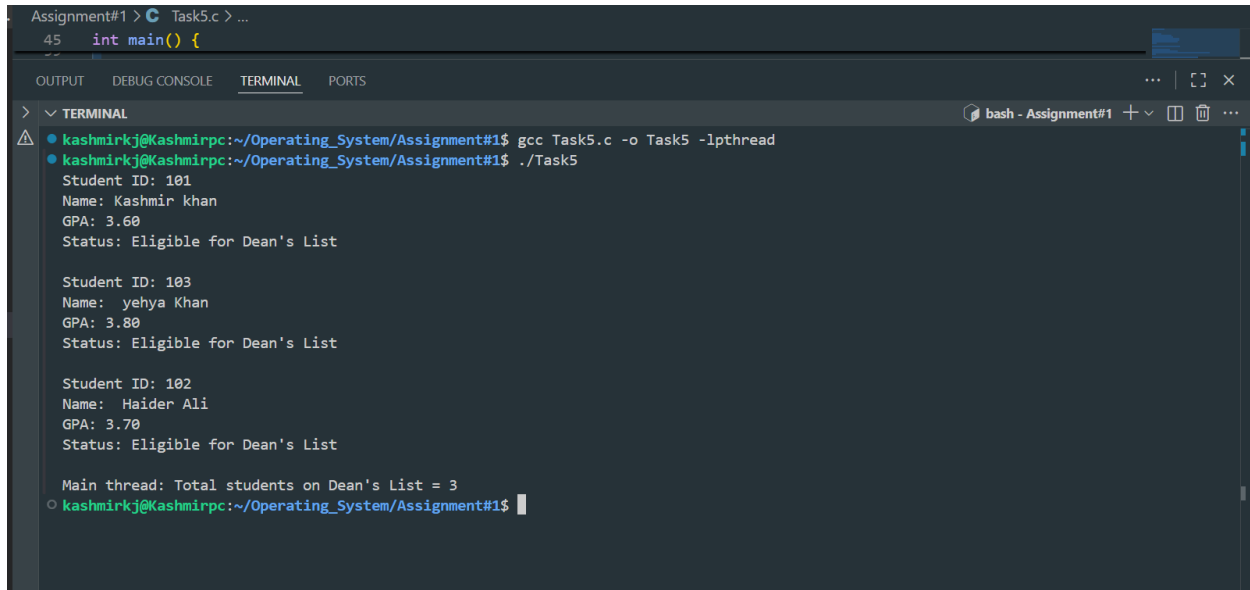
Code:

```

1 // Kashmir Jamshaid
2 // 23-NTU-CS-1167
3 // TASK-5
4 // Create a program that simulates a simple student database system.
5 // Define a struct: typedef struct { int student_id; char name[50]; float gpa; } Student;
6 // Create 3 threads, each receiving a different Student struct.
7 // Each thread prints student info and checks Dean's List eligibility (GPA ≥ 3.5).
8 // The main thread counts how many students made the Dean's List.
9
10 #include <stdio.h>
11 #include <pthread.h>
12 #include <string.h>
13
14 // Define Student structure
15 typedef struct {
16     int student_id;
17     char name[50];
18     float gpa;
19 } Student;
20
21 int dean_count = 0;
22 pthread_mutex_t lock; // Mutex for safe counter update find Lock
23
24 // Thread function
25 void* check_deans_list(void* arg) {
26     Student* s = (Student*)arg;
27
28     printf("Student ID: %d\n", s->student_id);
29     printf("Name: %s\n", s->name);
30     printf("GPA: %.2f\n", s->gpa);
31
32     // Check eligibility for Dean's List
33     if (s->gpa >= 3.5) {
34         printf("Status: Eligible for Dean's List\n\n");
35         pthread_mutex_lock(&lock);
36         dean_count++;
37         pthread_mutex_unlock(&lock);
38     } else {
39         printf("Status: Not eligible for Dean's List\n\n");
40     }
41
42     return NULL;
43 }
44
45 int main() {
46     pthread_t threads[3];
47     Student students[3] = {
48         {101, "Kashmir khan", 3.6},
49         {102, "Haider Ali ", 3.7},
50         {103, "yehya Khan", 3.8}
51     };
52
53     pthread_mutex_init(&lock, NULL);
54
55     // Create 3 threads
56     for (int i = 0; i < 3; i++) {
57         pthread_create(&threads[i], NULL, check_deans_list, &students[i]);
58     }
59
60     // Wait for all threads to finish or complete the task
61     for (int i = 0; i < 3; i++) {
62         pthread_join(threads[i], NULL);
63     }
64
65     printf("Main thread: Total students on Dean's List = %d\n", dean_count);
66
67     pthread_mutex_destroy(&lock);
68     return 0;
69 }
70

```

Output:



```
Assignment#1 > Task5.c > ...
45  int main() {
    kashmirkj@Kashmirpc:~/Operating_System/Assignment#1$ gcc Task5.c -o Task5 -lpthread
    kashmirkj@Kashmirpc:~/Operating_System/Assignment#1$ ./Task5
    Student ID: 101
    Name: Kashmir Khan
    GPA: 3.60
    Status: Eligible for Dean's List

    Student ID: 103
    Name: yehya Khan
    GPA: 3.80
    Status: Eligible for Dean's List

    Student ID: 102
    Name: Haider Ali
    GPA: 3.70
    Status: Eligible for Dean's List

    Main thread: Total students on Dean's List = 3
    kashmirkj@Kashmirpc:~/Operating_System/Assignment#1$
```

Section :B Short Questions :

1-Define an Operating System:

An operating system is software that manages a computer's hardware and software and helps users and programs work together smoothly.

2-Primary function of the CPU scheduler:

To select which ready process should be executed next by the CPU.

3-Three states of a process:

- Ready
- Running
- Waiting (blocked)

4-Process Control Block (PCB):

A PCB is a data structure that holds (stores) important details about a process, like its ID, current state, registers, and memory information.

5-Difference between a process and a program:

A program is a set of instructions (passive) stored on disk, while a process is that program running in the computer's memory.

6-Context switching:

It is the act of stopping one process, saving its state, and then loading and running another process.

7-CPU utilization and throughput:

CPU Utilization: The percentage of time the CPU is actively working on tasks.

Throughput: The number of processes the system finishes in a given period.

8-Turnaround time:

The total time from when a process is submitted until it finishes execution.

9-Waiting time calculation:

Waiting Time = Turnaround Time – Burst Time

10-Response Time:

The time between when a process is submitted and when it produces its first response or output.

11-Preemptive scheduling:

A scheduling method in which the CPU can interrupt and switch from a running process to another before the first one completes.

12-Non-preemptive scheduling:

A scheduling method in which a process keeps control of the CPU until it finishes or willingly gives it up

13-Two advantages of Round Robin scheduling:

- Ensures that all processes get a fair share of CPU time.

- Decreases the waiting time for shorter processes.

14-Major drawback of SJF algorithm:

It can lead to starvation, where long processes may never get enough CPU time to finish.

15-CPU idle time:

The period when the CPU is idle(unused) and not executing any tasks (waiting for task) .

16-Two common goals of CPU scheduling algorithms:

- Maximize CPU utilization.
- Minimize waiting and turnaround time.

17-Two reasons for process termination:

- Process completes execution.
- Process encounters an error or is killed by another process.

18-Purpose of wait() and exit() system calls:

- wait (): Makes the parent process pause until its child process finishes.
- exit (): Ends a process and sends its exit status back to the parent process.

19-Shared memory vs Message passing:

Processes share a common memory space to exchange data.	Processes communicate by sending and receiving messages.
Communication is faster but needs synchronization (e.g., semaphores).	Communication is slower but simpler and safer.
Suitable for closely coupled systems	Suitable for distributed systems.

20-Thread vs Process:

A thread is a lightweight task (unit of CPU) within a process	The process is a heavyweight, independent program in execution.
Threads share the same memory and resources.	Processes have separate memory and resources.
Context switching between threads is faster	Context switching between processes is slower.

21-Multithreading:

The ability of a process to execute multiple threads at the same time within a shared memory space.

22-CPU-bound vs I/O-bound process:

Spends most of its time performing computations (CPU work).	Spends most of its time waiting for I/O operations (like disk, keyboard, or network).
Needs more CPU time and less I/O time.	Needs more I/O time and less CPU time.
Example: Mathematical calculations, simulations, encryption.	Example: File reading, web requests, printing tasks.

23-What are the main responsibilities of the dispatcher?

The dispatcher gives the CPU to the process selected by the scheduler. It saves the current process state, loads the next one, and starts its execution managing the actual context switch.

24-Define starvation and aging in process scheduling.

Starvation: When a process waits too long because others always get the CPU first.

Aging: Fixes starvation by slowly raising the priority of waiting processes.

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

A time quantum is a short, fixed time each process gets to use the CPU in Round Robin scheduling before it's stopped and sent to the end of the queue.

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

- If the time quantum is too long, it acts like FCFS and gives slow responses.
- If it's too short, too many context switches happen, causing extra overhead.

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

The turnaround ratio shows how much total time a process took compared to the CPU time it needed.

Formula: Turnaround Ratio = Turnaround Time / Service Time

28-What is the purpose of a ready queue?

The ready queue stores all processes that are ready to run but waiting for the CPU. When the CPU is free, the scheduler selects one to execute.

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

- **A CPU burst** happens when a process is using the CPU to perform calculations or tasks.
- **An I/O burst** occurs when the process is waiting for input or output, like reading from a file or printing.

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

Round Robin scheduling avoids starvation since each process gets an equal chance to run for a fixed time slice, ensuring none are left waiting forever.

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

- The parent process uses **fork()** to create a new child process.
- The child process can call **exec()** to run a new program.
- The parent process uses **wait()** to pause until the child finishes.

- The child ends with **exit()**, sending its status back to the parent.

32-Define zombie and orphan processes.

- **Zombie process:** A process that has finished execution but remains in the process table because the parent hasn't collected its exit status.
- **Orphan process:** A process whose parent has ended before it; it gets adopted by the **init** process.

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

Priority Scheduling	Shortest Job First (SJF)
CPU is assigned based on priority value (higher priority runs first).	CPU is assigned to the process with the shortest CPU burst time.
Can cause starvation for low-priority processes.	Can cause starvation for long processes.
Priority can be based on importance or aging technique.	Priority is based only on burst time (shortest job first).

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

Context switch time is the period the CPU uses to save one process's state and load another's. It's overhead because no real user work happens during this time, the system is just managing processes.

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

- **Long-term scheduler:** Chooses which processes to bring into the ready queue.

- **Medium-term scheduler:** Manages pausing and resuming of processes.
- **Short-term scheduler:** Picks the next ready process to run on the CPU

36-Differentiate between User Mode and Kernel Mode.

Runs user applications like browsers or editors.	Runs operating system code and kernel-level tasks.
Has limited access to hardware and system resources.	Has full access to hardware and memory.
System calls are required to switch to kernel mode for privileged operations.	Executes privileged instructions directly without restrictions.

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 shows the steps a process goes through from when it starts to when it finishes in an operating system. Here is complete explanation of process states::

Process States

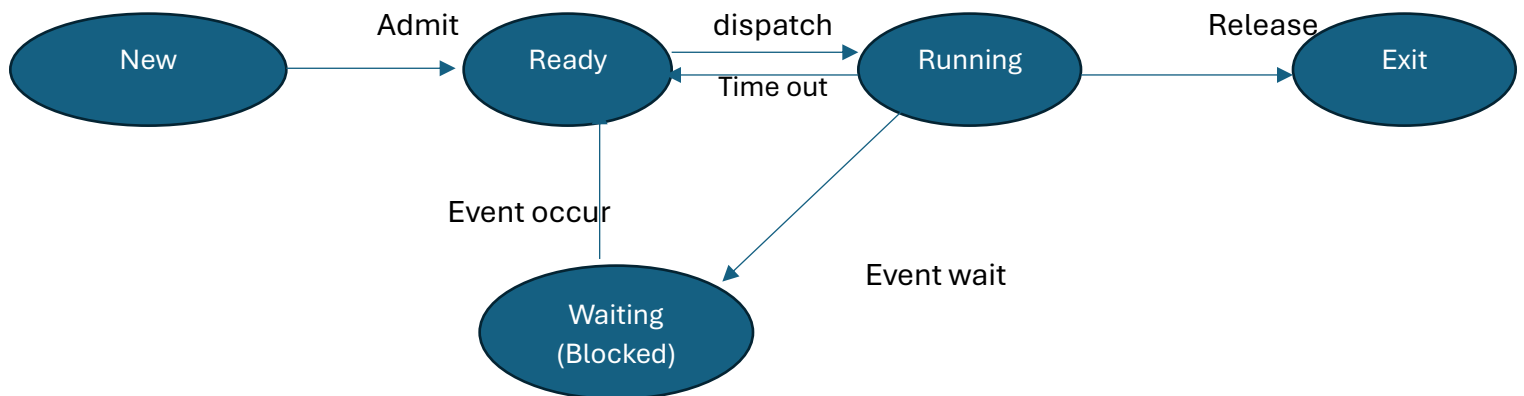
1. **New:** The process is being created, and resources are allocated (e.g., memory, process control block).
2. **Ready:** The process is ready to execute and is waiting for CPU allocation by the scheduler.
3. **Running:** The process is actively executing on the CPU.
4. **Waiting (Blocked):** The process is paused, waiting for an external event (e.g., I/O operation completion).

5. **Terminated (Exit):** The process has completed execution, and its resources are released

State Transitions

The process moves between states based on system events and scheduling decisions:

- **New → Ready:** The process is admitted by the scheduler and moves to the ready queue.
- **Ready → Running:** The scheduler dispatches the process to the CPU for execution.
- **Running → Waiting:** The process is interrupted (e.g., for I/O or event wait) and moves to the waiting state.
- **Waiting(Blocked) → Ready:** The event (e.g., I/O completion) occurs, and the process returns to the ready queue.
- **Running → Terminated (Exit):** The process completes execution or is terminated, and its resources are freed.



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

Answer:

What is a Context Switch:

A context switch occurs when the CPU switches from executing one process (or thread) to another. This allows multiple processes to share the CPU efficiently in a multitasking operating system. The operating system saves the state of the currently running process and loads the state of the next process, ensuring each process can resume execution seamlessly later.

Context: The state of a process, including all the information needed to pause and later resume its execution.

When it happens:

- Context switches occur during events like:
- A process moves from Running to Waiting (e.g., waiting for I/O).
- A process is preempted (e.g., its time slice ends in a preemptive scheduler).
- A higher-priority process becomes Ready.
- A process moves from Running to Terminated (exit).

Context Switch Overhead

The time taken to perform a context switch is called context switch overhead because it represents system work rather than progress on the process's actual task. During this time:

- No application-level computation occurs.
- The CPU is used to save and restore process states, which consumes cycles.
- Overhead varies depending on hardware, operating system, and the number of processes. Typically, it takes a few microseconds to milliseconds.

Information Saved and Restored:

During a context switch, the operating system saves the state of the current process and loads the state of the next process. The key information involved includes:

1. CPU Registers:

- General-purpose registers (e.g., data and address registers).
- Special-purpose registers (e.g., flags, condition codes).
- These store temporary data and computation states.

2. Program Counter (PC):

- Stores the memory address of the next instruction to execute.
- Ensures the process resumes at the correct point.

3. Stack Pointer:

- Points to the current position in the process's stack, which stores function calls, local variables, and return addresses.
- Critical for maintaining the process's call stack.

4. Process State:

- The current state in the process life cycle (e.g., Running, Ready, Waiting).
- Stored in the Process Control Block (PCB), which tracks all process metadata.

5. Memory and I/O Information:

- Memory Management Information: Page tables, memory mappings, or segment registers to track the process's memory allocation.
- I/O Status: Information about open files, pending I/O operations, or device states.

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

A PCB is a data structure the operating system uses to keep all the important details about a process. It helps the OS manage, switch, and keep track of processes easily. Each process has its own PCB.

Role:

The PCB helps the OS control processes, schedule them, and switch between them when needed.

Storage:

PCBs are stored safely in the system's kernel memory inside a process table.

Main Components of a PCB

- **Process ID (PID):**
A unique number that identifies each process.
- **Process State:**
Tells whether the process is new, ready, running, waiting, or finished.
- **Program Counter:**
Stores the address of the next instruction to run so the process can continue correctly.
- **CPU Registers:**
Hold the current working values and are saved when switching between processes.
- **Memory Information:**
Keeps details about where the process's data and instructions are stored in memory.
- **Accounting Information:**
Shows things like process priority, CPU usage time, and other stats.
- **I/O Status:**
Lists files and devices the process is using or waiting for.

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

Scheduler in OS:

A **scheduler** is a part of the operating system that decides which process should run at a given time.

It manages how processes move between different states (New, Ready, Running, Waiting) to make efficient use of the CPU and memory.

There are **three main types of schedulers**:

1. Long Term schedulers
2. Medium Term schedulers
3. Short Term schedulers

Type of Scheduler	Function	When It Works	Example
Long-Term Scheduler	Chooses which new processes to bring from the hard drive into main memory.	When new processes are created.	Adding new jobs to the ready queue in a batch system.
Medium-Term Scheduler	Pauses and later resumes processes to manage memory use.	When the system is overloaded or memory is full.	Swapping out background tasks and loading them again.
Short-Term Scheduler	Picks which ready process should use the CPU next.	Frequently, usually every few milliseconds.	Selecting the next process to run using Round Robin, etc

- The Long-Term Scheduler controls which processes enter memory.
- The Medium-Term Scheduler pauses and resumes processes.
- The Short-Term Scheduler picks the next process for the CPU.

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

Answer:

1-CPU Utilization

The percentage of time the CPU is performing useful work (executing processes in the **Running** state) versus being idle or handling overhead (e.g., context switching).

Relation to Schedulers:

- **Long-Term Scheduler:**
Balances CPU-bound and I/O-bound processes to keep the CPU busy.
 - **For example:** Admitting too many CPU-bound processes can overload the CPU, while too many I/O-bound processes may leave it idle.
- **Short-Term Scheduler:**
Ensures frequent context switches to keep the CPU allocated to **Ready** processes, minimizing idle time.
 - **Example:** In a web server handling multiple requests, high CPU utilization (e.g., 85%) indicates the CPU is efficiently processing requests rather than idling.

2-Throughput:

The number of processes completed per unit of time (e.g., processes per second or hour), reflecting the system's overall productivity.

Relation to Schedulers:

- **Long-Term Scheduler:**

Controls throughput by admitting an optimal number of processes (degree of multiprogramming). Too many processes can lead to resource contention, reducing throughput.

➤ **Short-Term Scheduler:**

Affects throughput by selecting processes that complete quickly (e.g., in shortest job first scheduling).

- **Example:** In a batch processing system (e.g., a payroll system), high throughput means processing 100 payroll jobs per hour instead of 50.

3-Turnaround Time :

The total time from when a process is submitted (enters the **New** state) to when it completes (reaches the **Terminated** state). It includes waiting time, execution time, and I/O time.

Relation to Schedulers:

➤ **Long-Term Scheduler:**

Affects turnaround time by deciding when processes enter the **Ready** queue. Delaying admission increases turnaround time.

➤ **Short-Term Scheduler:**

Minimizes turnaround time by prioritizing processes with shorter execution times (e.g., shortest job first algorithm).

➤ **Medium-Term Scheduler:**

Impacts turnaround time by swapping processes out, which can delay completion if memory is constrained.

- **Example:** In a university grading system, a process to compute student grades has a turnaround time of 5 seconds (submission to completion), which the scheduler aims to minimize.

4- Waiting Time

The total time a process spends in the **Ready** queue waiting for CPU allocation, excluding execution or I/O time.

Relation to Schedulers:

➤ Short-Term Scheduler:

Directly impacts waiting time by determining how quickly a process moves from **Ready** to **Running**. Algorithms like round-robin reduce waiting time for fairness.

➤ Long-Term Scheduler:

Indirectly affects waiting time by controlling the number of processes in the **Ready** queue.

- **Example:** In a time-sharing system, a text editor process waiting 200 milliseconds in the **Ready** queue before getting CPU time has a low waiting time, improving user experience.

5- Response Time

The time from when a process is submitted (enters the **New** state) to when it produces its first output or response (e.g., first CPU execution in the **Running** state).

Relation to Schedulers:

➤ Short-Term Scheduler:

Critical for minimizing response time by quickly dispatching interactive processes (e.g., in round-robin or priority scheduling for user-facing tasks).

➤ Long-Term Scheduler:

Affects response time by controlling how soon a process enters the **Ready** queue.

- **Example:** In a web browser, the response time is the delay (e.g., 100 milliseconds) between clicking a link and seeing the page start to load, which the scheduler aims to minimize.

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 :

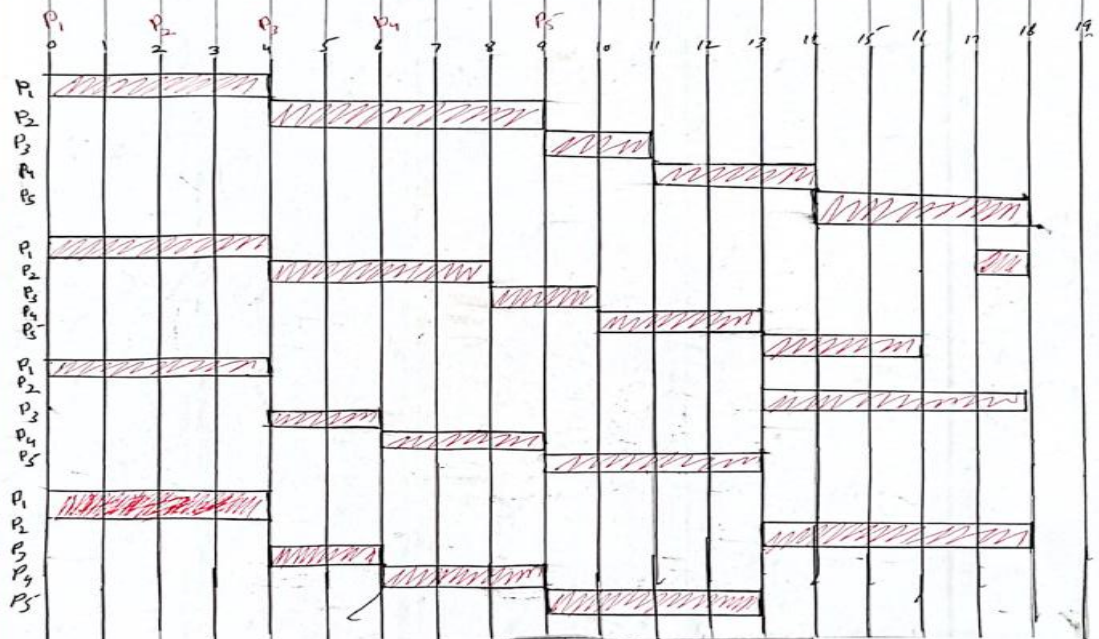
Part - A.

FCFS

RR
(0-4)

SJF
OR
SPN

SRTF



FCFS:

Process	Finish	Turnaround	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

- Avg Waiting time: $(0+2+5+5+5)/5 = 3.4$
- Avg turn around time: $(4+7+7+8+9)/5 = 7.0$
- Avg TR/TS Ratio: $(1+1.4+3.5+2.67+2.25)/5 = 2.16$
- CPU Idle time: 0

Round Robin (Q=4) Analysis:

Process	Finish	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

- 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: $(1+3.2+3+2.33+2)/5 = 2.31$
- CPU idle time: 0.

SJF Analysis:

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

- Avg Waiting time: $(0+11+0+0+0)/5 = 2.2$
- Avg Turnaround: $(4+16+2+3+4)/5 = 5.8$
- Avg Tr/T_s 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.

Best Algorithms:

- 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 metrics.

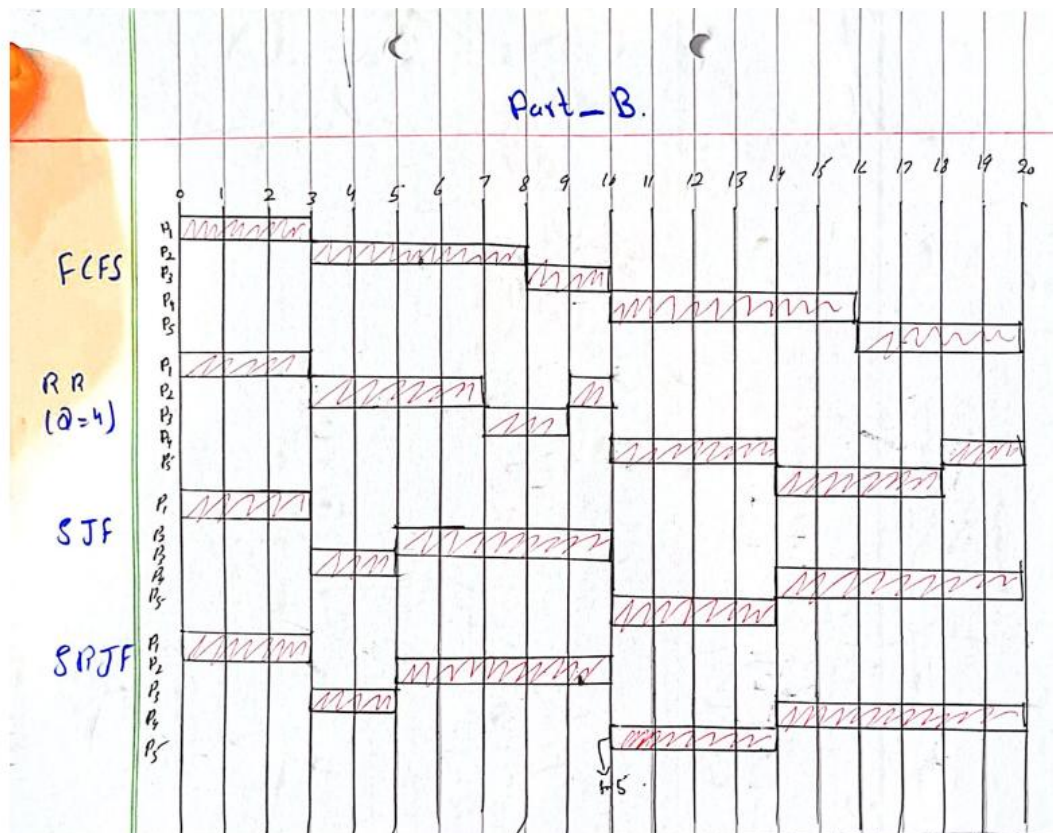
Part_B:

Part-B

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

Solution :

Chat :



FCFS Analysis

Process	finish	T_r	T_s	waiting	T_r/T_s
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 T_r/T_s : $(1+1.4+3.5+1.17+2.5)/5 = 1.91$
- CPU idle time: 0.

Round Robin ($Q=4$) Analysis

Process	finish	T_r	T_s	waiting	T_r/T_s
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 : $(1 + 1.8 + 3 + 1.83 + 2) / 5 = 1.93$
- 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 : $(1 + 1.8 + 1 + 1.83 + 1) / 5 = 1.32$
- CPU idle time : 0

SRTF 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	4	4	0	1.0

- Avg waiting: $(0 + 4 + 0 + 5 + 0) / 5 = 1.8$
- Avg turnaround: $(3 + 4 + 2 + 11 + 4) / 5 = 5.8$
- Avg Tr/Ts: $(1 + 1.8 + 1 + 1.83 + 1) / 5 = 1.33$
- CPU idle time: 0

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 metrics

Part C :

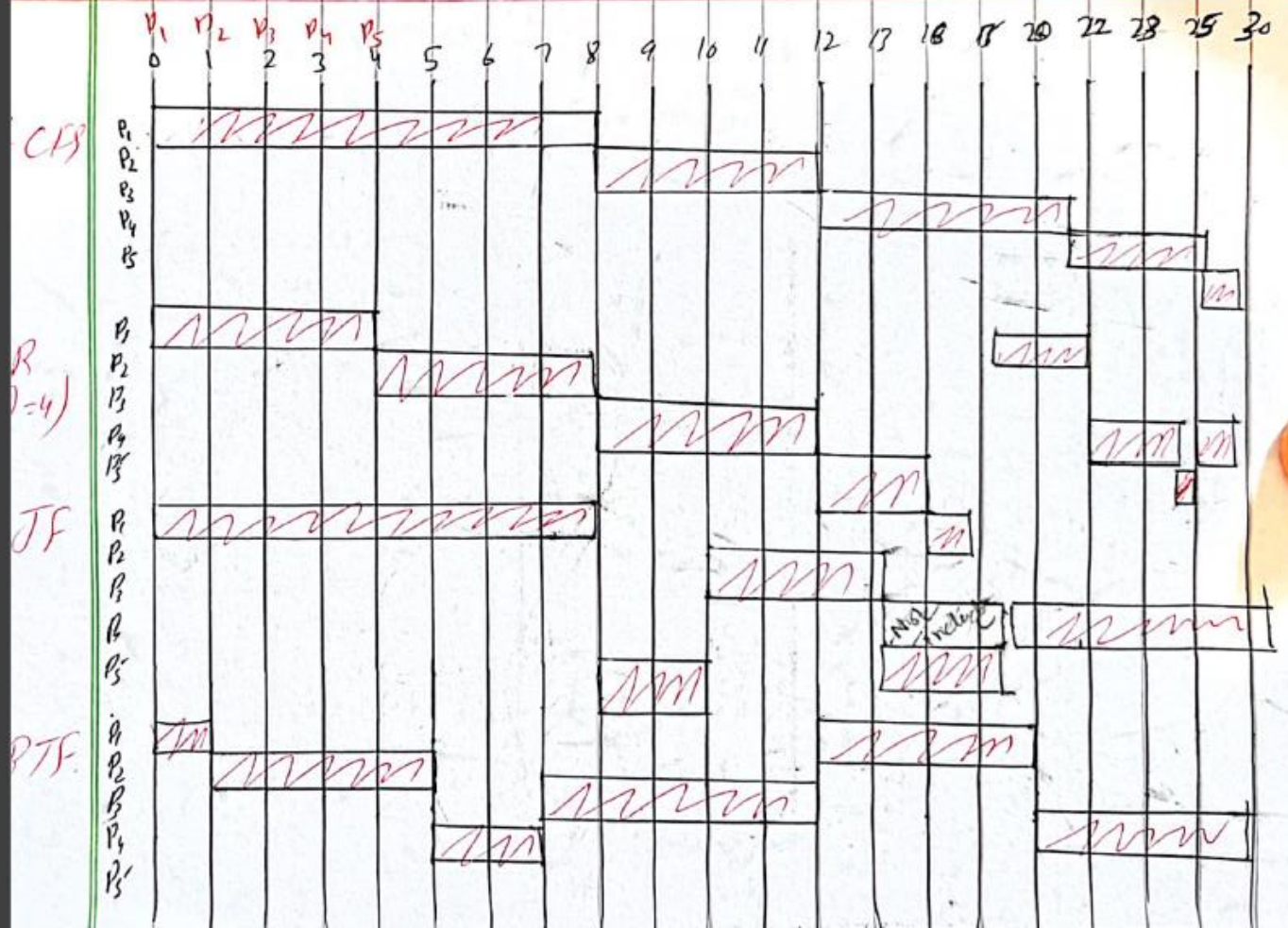
Part-C (Select Your own individual arrivals time and service time)

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

Solution :

Chat :

Part - C



FCFS Analysis:

Process	Finish	T_r	T_s	Waiting	T_r/T_s
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		17		11.4	4.49

CPU Idle time = 0

Round Robin (Q=4) Analysis:

Process	Finish	T_r	T_s	Waiting	T_r/T_s
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
Averages		18.6		13.6	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
Average		13.8	2.2	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 best performing algorithm for this datasets.
- Lowest Average turn around and waiting time.
- Optimal for minimizing average times, preemption allows short jobs to execute quickly.