



NATIONAL TEXTILE
UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

Subject

Operating System

SUBMITTED BY:

Fatima Waseem:

23-NTU-CS1155

SECTION SE: 5th (A)

SUBMITTED TO:

Sir Nasir

Assignment no 1

Task 1 – Thread Information Display

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

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

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

Code:

```
C Task1.c
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <unistd.h>
4  #include <stdlib.h> // for rand() and srand()
5  #include <time.h> // for time()
6
7  #define NUM_THREADS 5
8
9  // Reg no: 23-NTU-CS-1155 Name: Fatima Waseem
10 void *thread_function(void *args){
11     int thread_num = *(int *)args; //get thread number (1,2,3, ...)
12     pthread_t tid = pthread_self(); // thread ID
13
14     printf("Thread %d started. Thread ID: %lu\n", thread_num, tid);
15
16     int sleep_time = (rand() % 3) + 1; // random number between 1 and 3
17     sleep(sleep_time);
18
19     printf("Thread %d (ID: %lu) completed after %d seconds.\n", thread_num, tid, sleep_time);
20
21     return NULL;
22 }
C Task1.c
23
24 int main(){
25     pthread_t threads[NUM_THREADS];
26     int thread_args[NUM_THREADS];
27
28     srand(time(NULL)); //seed random number generator
29
30     for(int i = 0; i < NUM_THREADS; i++){
31         thread_args[i] = i + 1; // thread number 1-5
32         pthread_create(&threads[i], NULL, thread_function, &thread_args[i]);
33     }
34
35     for (int i = 0; i < NUM_THREADS; i++){
36         pthread_join(threads[i], NULL);
37     }
38
39     printf("All threads have finished execution. \n");
40     return 0;
41 }
42 }
```

Output:

```
fatima@DESKTOP-3BA3T21:~/23-NTU-CS-1155 Assignment1$ ./task1.out
Thread 1 started. Thread ID: 138660936677056
Thread 2 started. Thread ID: 138660928284352
Thread 3 started. Thread ID: 138660919891648
Thread 4 started. Thread ID: 138660911498944
Thread 5 started. Thread ID: 138660903106240
Thread 1 (ID: 138660936677056) completed after 2 seconds.
Thread 5 (ID: 138660903106240) completed after 2 seconds.
Thread 3 (ID: 138660919891648) completed after 2 seconds.
Thread 4 (ID: 138660911498944) completed after 2 seconds.
Thread 2 (ID: 138660928284352) completed after 3 seconds.
All threads have finished execution.
```

Question 2:

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.

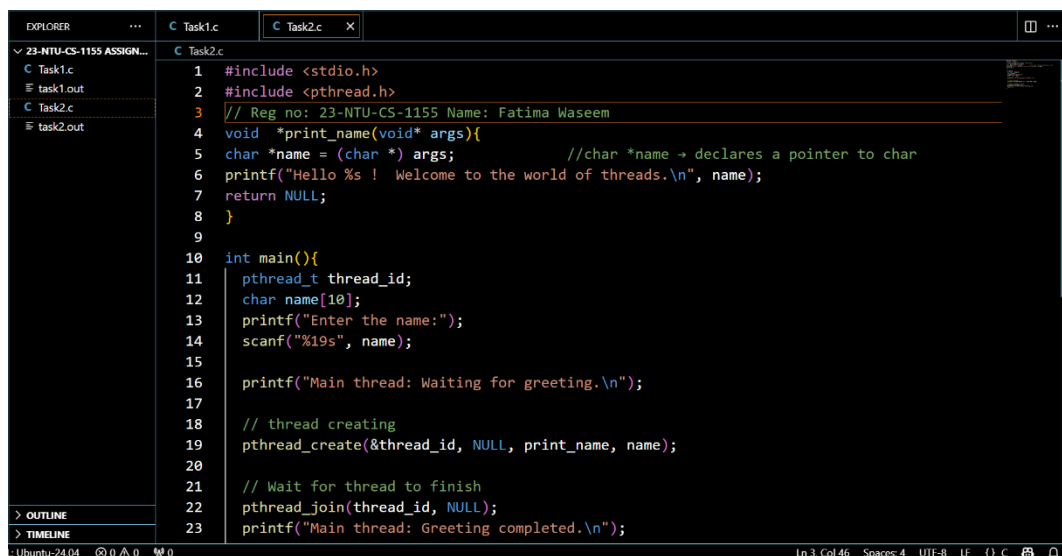
Example Output:

Main thread: Waiting for greeting...

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

Main thread: Greeting completed.

Code:



```
EXPLORER  ...  C Task1.c  C Task2.c  X
23-NTU-CS-1155 ASSIGN...
  C Task1.c
  task1.out
  C Task2.c
  task2.out

C Task2.c
1  #include <stdio.h>
2  #include <pthread.h>
3  // Reg no: 23-NTU-CS-1155 Name: Fatima Waseem
4  void *print_name(void* args){
5  char *name = (char *) args;           //char *name → declares a pointer to char
6  printf("Hello %s ! Welcome to the world of threads.\n", name);
7  return NULL;
8  }
9
10 int main(){
11 pthread_t thread_id;
12 char name[10];
13 printf("Enter the name:");
14 scanf("%19s", name);
15
16 printf("Main thread: Waiting for greeting.\n");
17
18 // thread creating
19 pthread_create(&thread_id, NULL, print_name, name);
20
21 // Wait for thread to finish
22 pthread_join(thread_id, NULL);
23 printf("Main thread: Greeting completed.\n");
```

Output:

```
fatima@DESKTOP-3BA3T21:~/23-NTU-CS-1155 Assignment1$ gcc Task2.c -o task2.out
fatima@DESKTOP-3BA3T21:~/23-NTU-CS-1155 Assignment1$ ./task2.out
Enter the name:Fatima
Main thread: Waiting for greeting.
Hello Fatima ! Welcome to the world of threads.
Main thread: Greeting completed.
fatima@DESKTOP-3BA3T21:~/23-NTU-CS-1155 Assignment1$
```

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 Task1.c  C Task2.c  C Task3.c  X
C Task3.c
1  #include <stdio.h>
2  #include <pthread.h>
3  // Reg no: 23-NTU-CS-1155 Name: Fatima Waseem
4  void* print_number(void* arg){
5      int num = *(int*)arg;
6      int square = num * num;
7      int cube = num * num * num;
8      printf("Thread received %d and its square is %d and its cube is %d.\n", num, square, cube);
9      return NULL;
10 }
11 int main(){
12     pthread_t thread_id;
13     int number;
14     printf("Enter the number:\n");
15     scanf("%d", &number);
16
17     //create thread
18     pthread_create(&thread_id, NULL, print_number, &number);
19
20     //wait for the thread finish
21     pthread_join(thread_id, NULL);
22     printf("Main thread done.\n");
23     return 0;
}
Ln 6, Col 28  Spaces: 4  UTF-8  LF  {}  C  [ ]  Q
```

Output:

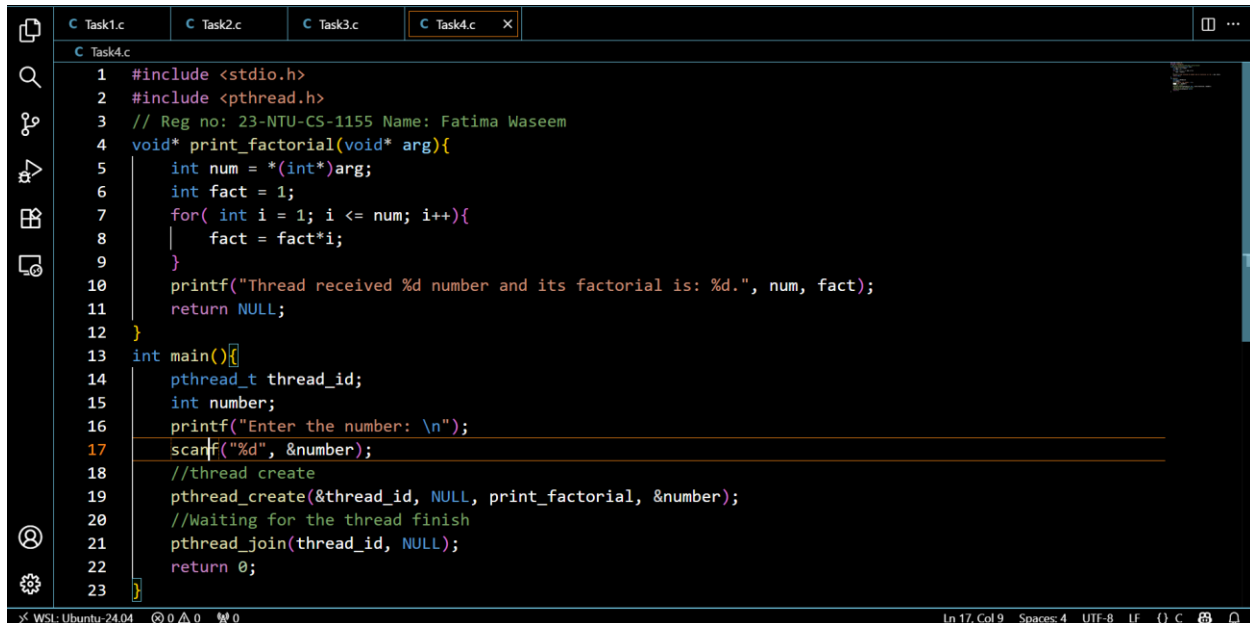
```
fatima@DESKTOP-3BA3T21:~/23-NTU-CS-1155 Assignment1$ gcc Task3.c -o task3.out
fatima@DESKTOP-3BA3T21:~/23-NTU-CS-1155 Assignment1$ ./task3.out
Enter the number:
2
Thread received 2 and its square is 4 and its cube is 8.
Main thread done.
fatima@DESKTOP-3BA3T21:~/23-NTU-CS-1155 Assignment1$
```

Task 4 – Thread Return Values

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

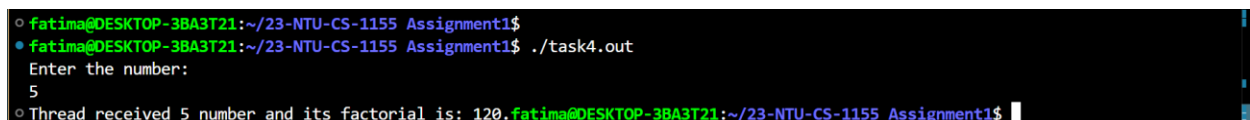
- The thread should return the result using a pointer.
- The main thread prints the result after joining.

Code:



```
1 #include <stdio.h>
2 #include <pthread.h>
3 // Reg no: 23-NTU-CS-1155 Name: Fatima Waseem
4 void* print_factorial(void* arg){
5     int num = *(int*)arg;
6     int fact = 1;
7     for( int i = 1; i <= num; i++){
8         fact = fact*i;
9     }
10    printf("Thread received %d number and its factorial is: %d.", num, fact);
11    return NULL;
12 }
13 int main(){
14     pthread_t thread_id;
15     int number;
16     printf("Enter the number: \n");
17     scanf("%d", &number);
18     //thread create
19     pthread_create(&thread_id, NULL, print_factorial, &number);
20     //Waiting for the thread finish
21     pthread_join(thread_id, NULL);
22     return 0;
23 }
```

Output:



```
fatima@DESKTOP-3BA3T21:~/23-NTU-CS-1155 Assignment1$
fatima@DESKTOP-3BA3T21:~/23-NTU-CS-1155 Assignment1$ ./task4.out
Enter the number:
5
Thread received 5 number and its factorial is: 120.fatima@DESKTOP-3BA3T21:~/23-NTU-CS-1155 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 ($GPA \geq 3.5$).
- The main thread counts how many students made the Dean's List.

Code:

```
C Task1.c C Task2.c C Task3.c C Task4.c C Task5.c X
C Task5.c
1  #include <stdio.h>
2  #include <pthread.h>
3  int deans_list = 0;
4  pthread_mutex_t lock; //mutex to protect shared variable
5  typedef struct{
6      int student_id;
7      char name[50];
8      float gpa;
9  } Student;
10
11 void* students_record(void* args){
12     Student *data = (Student*)args;
13
14     printf("Student ID: %d\n", data->student_id);
15     printf("Name: %s\n", data->name);
16     printf("GPA: %.2f\n", data->gpa);
17
18     //check Deans's List eligibility
19     if(data->gpa >= 3.5){
20         printf("%s on the Dean's list!\n", data->name);
21         pthread_mutex_lock(&lock);
22         deans_list++;
23         pthread_mutex_unlock(&lock);
24     }
25 }
```

Output:

```
fatima@DESKTOP-3BA3T21:~/23-NTU-CS-1155 Assignment1$ gcc Task5.c -o task5.out
fatima@DESKTOP-3BA3T21:~/23-NTU-CS-1155 Assignment1$ ./task5.out
Student ID: 101
Name: Fatima
GPA: 3.60
Fatima on the Dean's list!
Student ID: 102
Name: Farah
GPA: 3.70
Farah on the Dean's list!
Student ID: 103
Name: Eman
GPA: 3.70
Eman on the Dean's list!
fatima@DESKTOP-3BA3T21:~/23-NTU-CS-1155 Assignment1$
```

Section-B: Short Questions

1. Define an Operating System in a single line.

Operating System is a system software that acts as a **bridge** between the **user** and the **computer hardware**.

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

The CPU scheduler decides which **program (or thread)** executed next by the CPU from the **ready queue**.

3. List any three states of a process.

- **Ready State:**
The process is ready to run and waiting for the CPU to assigned.
- **Running State:**
The process is currently being executed by the CPU.
- **Waiting (or Blocked):**
The process is paused and waiting for some event like (input/output completion).

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

A **Process Control Block (PCB)** is a data structure used by **operating system** to store all important information about a process.

It contains details:

Process ID: Unique number for each process.

Process State: whether it's running, ready, or waiting.

Program counter: to resume execution correctly.

Memory management info: Which memory the process is using.

I/O and file info: what devices or files the process is using.

PCB acts as the identity card of a process that the OS used to manage and track it.

5. Differentiate between a process and a program.

Features	Program	Process
Definition	A program is a set of instructions written to perform a specific task.	A process is a program in execution (a running instance of a program).
State	Passive: It just exists as a file on disk.	Active: It is being executed by the CPU.
Lifespan	Permanent until deleted.	Temporary, it exists while the program runs.
Storage	Stored in secondary memory (like a hard drive).	Loaded into main memory (RAM) during execution.
Example	Chrome.exe Stored on computer.	When you open Chrome, each running window or tab becomes a process.

6. What do you understand by context switching?

Context Switching is the process by which the CPU switches form one process to another.

7. Define CPU utilization and throughput.

CPU utilization:

The percentage of time the CPU is actively working (not idle).

Example:

If CPU is busy 80% of the time and idle 20%, CPU utilization = 80%.

Throughput:

The number of processes completed per unit time.

Example:

If 50 processes finish in 10 seconds, throughput = **5 processes/seconds**.

8. What is the turnaround time of a process?

Turnaround time is the total time taken for a process to complete, from the **moment it enters the ready queue** to the moment it **finishes execution**.

It measures the overall waiting + execution time of a process.

9. How is waiting time calculated in process scheduling?

Waiting time in process scheduling is the total time a process spends in the ready queue waiting to get the CPU.

Formula:

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

Turnaround Time: Total time from arrival to completion.

Burst Time: Is the time process actually spends executing on the CPU.

Example:

Process arrives at 2s, completes at 10s then Turnaround Time = 8s

CPU execution (**Burst Time**) = 5s

Waiting Time = $8 - 5 = 3$ seconds.

10. Define response time in CPU scheduling.

Response Time in CPU scheduling is the first time when a process arrives in the ready queue until it gets the CPU for the first time.

Formula:

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

Example:

- Process arrives at 2s.
- CPU starts executing it at 5s.
- Response Time = $5 - 2 = 3$ s.

Response time measures **how quickly a process starts executing**, not how long it takes to finish.

11. What is preemptive scheduling?

Preemptive Scheduling is a CPU scheduling method where the OS can interrupt a running process to assign the CPU to another process with higher priority or based on the scheduling algorithm.

12. What is non-preemptive scheduling?

Non-Preemptive Scheduling is a CPU scheduling method where a running process cannot be interrupted.

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

Fairness:

- Every process gets an equal share of the CPU in a cyclic order.
- No process can starve (wait indefinitely).

Good Response Time:

- Since each process gets the CPU frequently, interactive processes (like typing or mouse input) respond quickly.

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

Starvation of long processes:

If short processes keep arriving, longer processes may wait indefinitely in the ready queue.

15. Define CPU idle time.

CPU idle Time is the time duration when the CPU is not doing any useful work because there are no processes in the ready queue to execute.

Example:

- If all processes are waiting for I/O operations, the CPU has nothing to do.
- During this time, CPU is idle.

16. State two common goals of CPU scheduling algorithms.

Maximize CPU Utilization:

- Keep CPU as busy as possible to improve system efficiency.

Minimizing waiting time and turnaround time:

- Ensure processes spend less time in waiting in the ready queue and finish quickly.

17. List two possible reasons for process termination.

Normal completion:

- The process finishes its task successfully and exits.

Abnormal Termination:

- The process is terminated due to an error or system problem.

Example:

- Division by zero, illegal memory access, or system crash.

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

Wait():

A parent process waits for its child process to finish execution.

Exit():

A process terminates its execution and inform the OS that it has finished.

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

Shared memory:

- Process communicate by reading and writing to a common memory area.
- Faster because data is directly accessed in memory.

Message-passing:

- Processes communicate by sending and receiving messages via the OS.

20. Differentiate between a thread and a process.

Features	Process	Thread
Definition	An independent program in execution.	A smaller unit of execution within a process.
Memory	Is has personal memory space (code, data, stack).	Shares the memory and resources of their parent process.
Communication	Inter-process communication (IPC) is required to communicate.	Threads can communicate easily via shared memory.
Example	Running Chrome and word simultaneously (two processes).	Multiple tabs in Chrome and multiple functions in Word running as threads.

21. Define multithreading.

Multithreading is the ability of the CPU or a single process to execute multiple threads at the same time.

- Each thread runs different part of the process but shares the same memory and resources.
- It helps in doing multiple tasks simultaneously within one process.

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

Type	CPU-Bound Process	I/O-Bound Process
Definition	A Process that spends most of its time using the CPU to perform calculations or processing.	A process that spends most of its time waiting for I/O operations (like reading from disk or keyboard).
CPU Usage	High usage of CPU.	Low usage of CPU.
Example	Video rendering, scientific calculations, data encryption.	Reading a document, web browsing.
Performance Goal	Needs a fast CPU to run efficiently	Needs fast I/O devices (like SSDs or network).

23. What are the main responsibilities of the dispatcher?

The dispatcher is a small OS program responsible for switching the processor from one process to another.

24. Define starvation and aging in process scheduling.

Starvation:

Starvation occurs when a process waits indefinitely because the gives preference to other processes.

Example:

A low priority process never gets CPU time because high-priority processes keep arriving.

Aging:

Aging is a technique to prevent starvation. It works by gradually increasing the priority of a waiting process as time passes, so it will eventually get the CPU.

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

A time quantum is the maximum time a process can run continuously before the CPU switches to another process.

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

Time Quantum too large:

- When time quantum is too large so the system becomes less responsive.
- Long processes may hog the CPU.

Time Quantum too Small:

- CPU switches processes very frequently.
- Less CPU time is available for actual execution, reducing efficiency.

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

The ratio of turnaround time to service time for a process.

Formula:

Turnaround Ratio = Turnaround Time (TR)/Service Time (TS)

(TR): Total time from processes arrival to completion.

(TS): Actual CPU time the process requires.

Example:

If **(TS) = 5ms** and **(TR) = 10ms** then **Ratio = 10/5 = 2**

A **higher** ratio means the process waited longer relative to its CPU time.

28. What is the purpose of a ready queue?

A queue that hold all the processes that are ready to execute and are waiting for the CPU allocation.

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

CPU burst:

A period when a process uses the CPU intensively to execute instructions.

I/O Burst:

A period when a process waits for or performs I/O operations (like reading/writing to disk or keyboard).

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

The FCFS and RR scheduling algorithms are starvation-free.

FCFS: Processes are executed in the order they arrive. Every process will eventually get the CPU.

RR: Each process gets a fixed time slice in a cyclic order. So even low- priority processes will get CPU time, so no process waits indefinitely.

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

- Parent Process calls fork ().
- Child Process gets a unique PID.
- Child may execute a new program with exec ().
- Parent may wait Using wait ().
- Process initialization.

32. Define zombie and orphan processes.

Zombie Process:

- A zombie process is a dead process that has finished execution but still has an entry in the process table.
- A child process finishes, but the parent hasn't called wait () child becomes zombie.

Orphan Process:

- An orphan process is a running process whose parent has terminated.
- Parent process ends while the child is still running child becomes an orphan.

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

CPU is allocated based on process **priority**.

Each process has a priority number, higher priority runs first.

Shortest Job First:

CPU is allocated to the process with the smallest burst time.

No priority numbers, scheduling depends on CPU burst length.

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

The time taken by the CPU to switch from one process to another.

Why it is considered overhead:

- Because during context switching, the CPU is not executing any user process instructions.
- It uses CPU time only to perform administrative work, which reduces overall efficiency.

35. List and briefly describe the three levels of schedulers in an Operating System.**Long-term scheduler:**

Runs infrequently, when a new process is created.

Medium-term scheduler:

Runs occasionally, can suspend or resume processes.

Short-term scheduler (CPU scheduler):

Runs frequently, usually every few milliseconds.

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

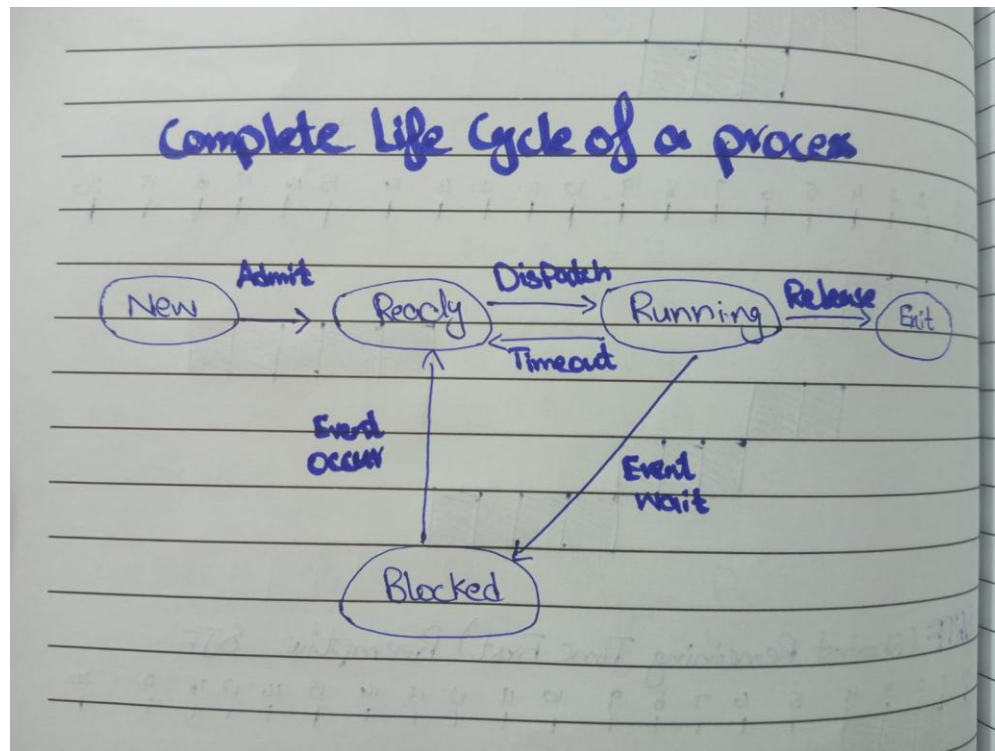
- Mode in which user application run.
- Limited access to system resources, cannot execute privileged instructions.
- Requires system calls to enter kernel mode.

Kernel mode:

- Mode in which OS core (kernel) runs.
- Full access to all hardware and CPU instructions.
- Executes privileged instructions returns to user mode after completion.

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

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



1. **Ready:** The program is waiting for its turn to use the processor (CPU).
2. **Running:** The program is actively using the CPU to do its work.
3. **Blocked:** The program is stuck waiting for something, like you waiting for a file to download. It can't continue until that thing happens.

The program constantly switches between these states:

- It goes from **Ready → Running** when the OS gives it a turn.
- It goes from **Running → Ready** if its turn is over.
- It goes from **Running → Blocked** when it needs to wait for something.
- It goes from **Blocked → Ready** when the thing it was waiting for finally happens.

This cycle keeps repeating until the program is finished and **Ends**.

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 changes from executing one process (or thread) to another. This is managed by the **Operating System (OS)** so that multiple processes can share a single CPU efficiently (multitasking).

Context Switch Overhead

- It refers to the **extra time and CPU effort** required to save the current process's state and load the next process's state.
- During this time, **no useful work** (like running user programs) is done only the OS performs internal management.
- The **higher the frequency** of context switches, the **greater the performance cost**.

Information Saved and Restored

When a context switch occurs, the OS must save the **current state** of the running process and load the **previously saved state** of the next process.
This information is stored in the **Process Control Block (PCB)** of each process.

Information Saved:

1. **CPU Registers** (e.g., program counter, stack pointer, general-purpose registers)
To resume execution from the same point later.
2. **Program Counter (PC)**
Holds the address of the next instruction to execute.
3. **Stack Pointer and Stack Contents**
Keeps track of function calls, parameters, and local variables.
4. **Memory Management Information**
Such as page tables or segment registers.
5. **Process State**
Ready, Running, or Waiting.
6. **I/O Status Information**
Open files, pending I/O requests, etc.

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

Process Control Block (PCB)

A **PCB** is a data structure used by the **Operating System (OS)** to store all important information about a **process**.

It helps the OS to **pause and resume** a process correctly and manage multiple processes efficiently.

Main Components of PCB

1. **Process ID (PID):**
A unique number that identifies each process.
2. **Process State:**
Shows the current state of the process, such as **New, Ready, Running, Waiting**, or **Terminated**.
3. **Program Counter (PC):**
Holds the **address of the next instruction** that the process will execute.
4. **CPU Registers:**
Stores the contents of CPU registers (like accumulator, stack pointer) when the process is not running, so it can continue later from the same point.
5. **CPU Scheduling Information:**
Includes details like **process priority** and **queue pointers** to help the CPU scheduler decide which process runs next.
6. **I/O Status Information:**
Lists all **open files, I/O devices**, and **pending I/O operations** the process is using.

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

1. Long-Term Scheduler (Job Scheduler)

Main Role:

- The **Long-Term Scheduler** decides **which processes should be brought into main memory (RAM)** from secondary storage (like a hard drive).
- It **controls the degree of multiprogramming**, meaning how many processes are in memory at a time.

Key Points:

- Selects processes from the **job pool** (on disk) and loads them into the **ready queue**.
- Maintains a **balance** between:
 - **CPU-bound processes:** Use more CPU time and less I/O.
 - **I/O-bound processes:** Spend more time waiting for I/O.
- Too many processes → memory overload
Too few processes → CPU underutilization

Frequency:

- It runs **less frequently** because job admission doesn't happen often.

Example:

- In a **batch processing system**, when many jobs are waiting on disk, the Long-Term Scheduler chooses which jobs to bring into memory for execution.

2. Medium-Term Scheduler (Swapper)

Main Role:

- The **Medium-Term Scheduler** is responsible for **suspending and resuming** processes.
- It helps manage **memory** more efficiently by **swapping** processes in and out of the main memory.

Key Points:

- When the system is overloaded or memory is full, it may **temporarily remove** a process from memory (swap it out) and save it on disk.
- Later, when memory is free, it **swaps the process back in**.
- Improves the performance by keeping only **active processes** in memory.

Frequency:

- It runs **occasionally**, depending on memory and system load.

Example:

- Suppose five processes are running, but memory is full. The OS may **suspend** (swap out) one process that's waiting for input, so another ready process can be **brought in** to use the CPU.

3. Short-Term Scheduler (CPU Scheduler)

Main Role:

- The **Short-Term Scheduler** decides **which of the ready processes gets to use the CPU next**.
- It performs **process switching** very frequently.

Key Points:

- Runs every time a process:
 - Starts or stops running
 - Is interrupted
 - Completes an I/O operation
- Uses **CPU scheduling algorithms**, such as:

- FCFS (First Come First Serve)
- SJF (Shortest Job First)
- Priority Scheduling
- Round Robin

Frequency:

- It runs **very frequently** every few milliseconds so it must be **fast and efficient**.

Example:

- Suppose 3 processes are in the ready queue.
The Short-Term Scheduler selects which one gets the CPU next based on the scheduling algorithm (e.g., Round Robin gives each process a time slice).

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

CPU Scheduling Criteria

CPU scheduling criteria are the **factors** used to **measure and compare** how efficiently a CPU scheduling algorithm performs.

They help determine **how well the operating system manages processes**.

The main criteria are:

1. **CPU Utilization**
2. **Throughput**
3. **Turnaround Time**
4. **Waiting Time**
5. **Response Time**

1. CPU Utilization

Meaning:

- It is the **percentage of time the CPU is actively working** (not idle).
- A high CPU utilization means the CPU is being used efficiently.

Goal:

- **Maximize** CPU utilization (ideally between **80% to 100%**).
- The CPU should stay **as busy as possible**, executing processes rather than waiting.

Example:

If the CPU was busy for 8 seconds in a 10-second interval:
Utilization = $(8/10) \times 100 = 80\%$

2. Throughput

Meaning:

- The **number of processes completed** per unit time.
- It shows how much **work** the CPU can complete.

Goal:

- **Maximize** throughput complete as many processes as possible in a given time.

Example:

If 10 processes are completed in 5 seconds
Throughput = $10 / 5 = 2$ processes per second

3. Turnaround Time

Meaning:

- The **total time** taken by a process from **submission to completion**.
- It includes all time spent in the ready queue, waiting, executing, and doing I/O.

Goal:

- **Minimize** turnaround time so that processes finish quickly.

Example:

If a process arrives at time 0 and finishes at time 10
Turnaround time = $10 - 0 = 10$ seconds

4. Waiting Time

Meaning:

- The **total time a process spends waiting** in the ready queue before getting the CPU.

Goal:

- **Minimize** waiting time fewer waiting means better performance and responsiveness.

Example:

If turnaround time = 15 sec and CPU burst = 5 sec
Waiting time = $15 - 5 = 10$ sec

5. Response Time

Meaning:

- The time between the submission of a request and the first response (or first CPU execution).
- Important for **interactive systems** where users expect quick feedback.

Goal:

- **Minimize** response time faster response improves user experience.

Example:

If a process arrives at time 0 and gets CPU for the first time at time 3
Response time = $3 - 0 = 3$ seconds

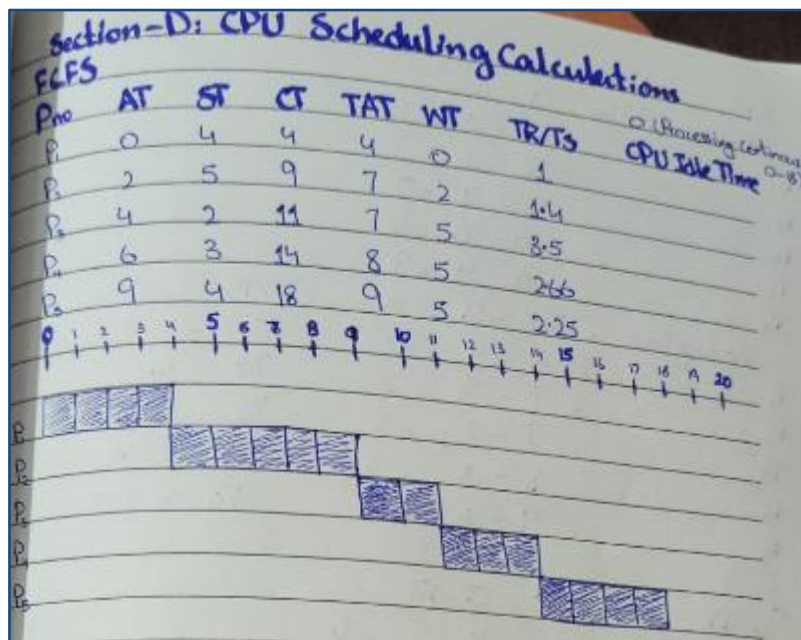
Section-D: CPU Scheduling Calculations

- 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

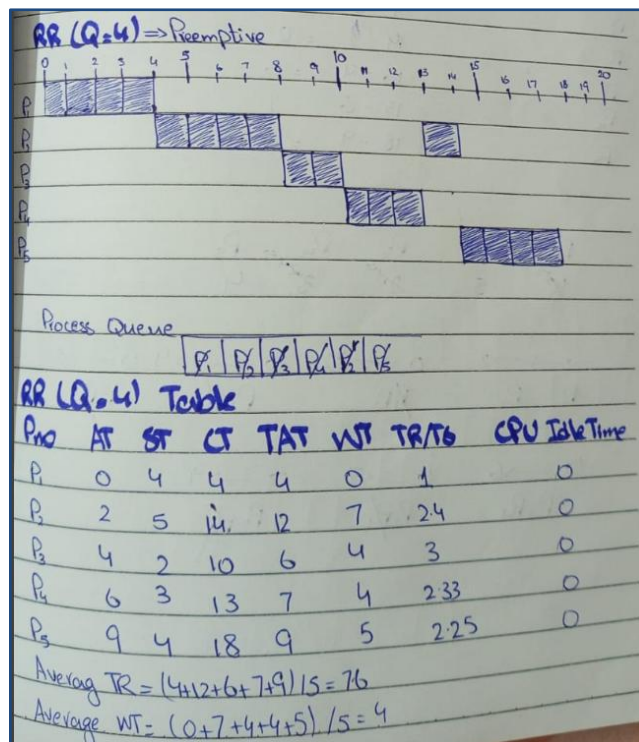
Part-A

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

FCFS:



RR (Q=4):

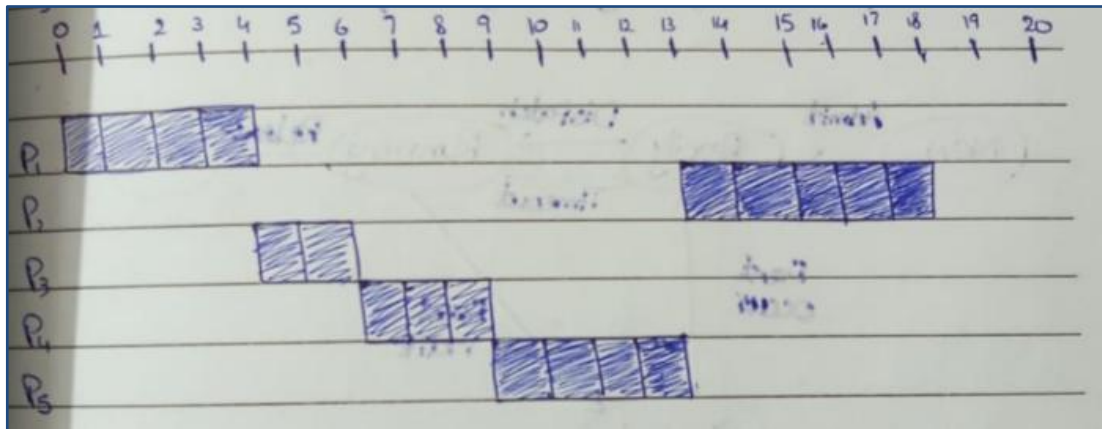


SJF:

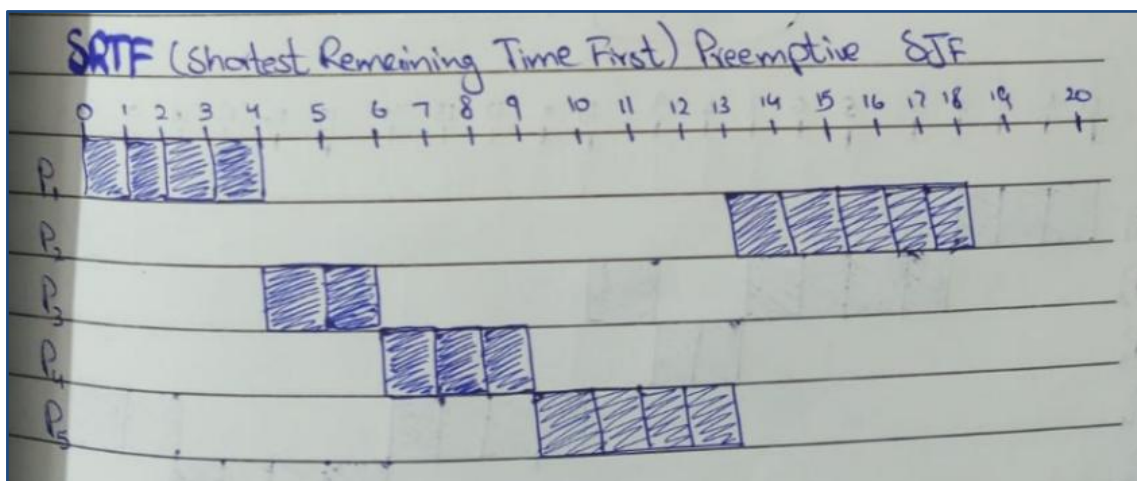
3) **SJF (non-preemptive)** Date _____

P _{no}	AT	ST	CT	TR	WT	TR/TS
P ₁	0	4	4	4	0	1
P ₂	2	5	18	16	11	3.2
P ₃	4	2	6	2	0	1
P ₄	6	3	9	3	0	1
P ₅	9	4	13	4	0	1

Average TR = 5.8
 Average WT = 2.2
 CPU Idle Time = 0



SRTF:



4) SRTF

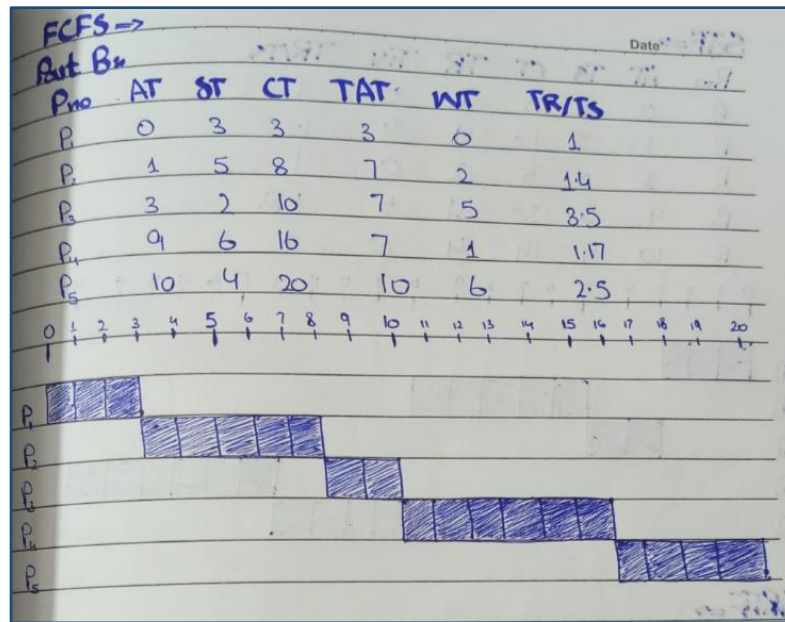
P _{no}	AT	BT	CT	TAT	WT	TR/TS
P ₁	0	4	4	4	0	1
P ₂	2	5	18	16	11	3.2
P ₃	4	2	6	2	0	1
P ₄	6	3	9	3	0	1
P ₅	9	4	13	4	0	1

Average TAT = 5.8
Average WT = 2.2
CPU Idle Time = 0

Part-B

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

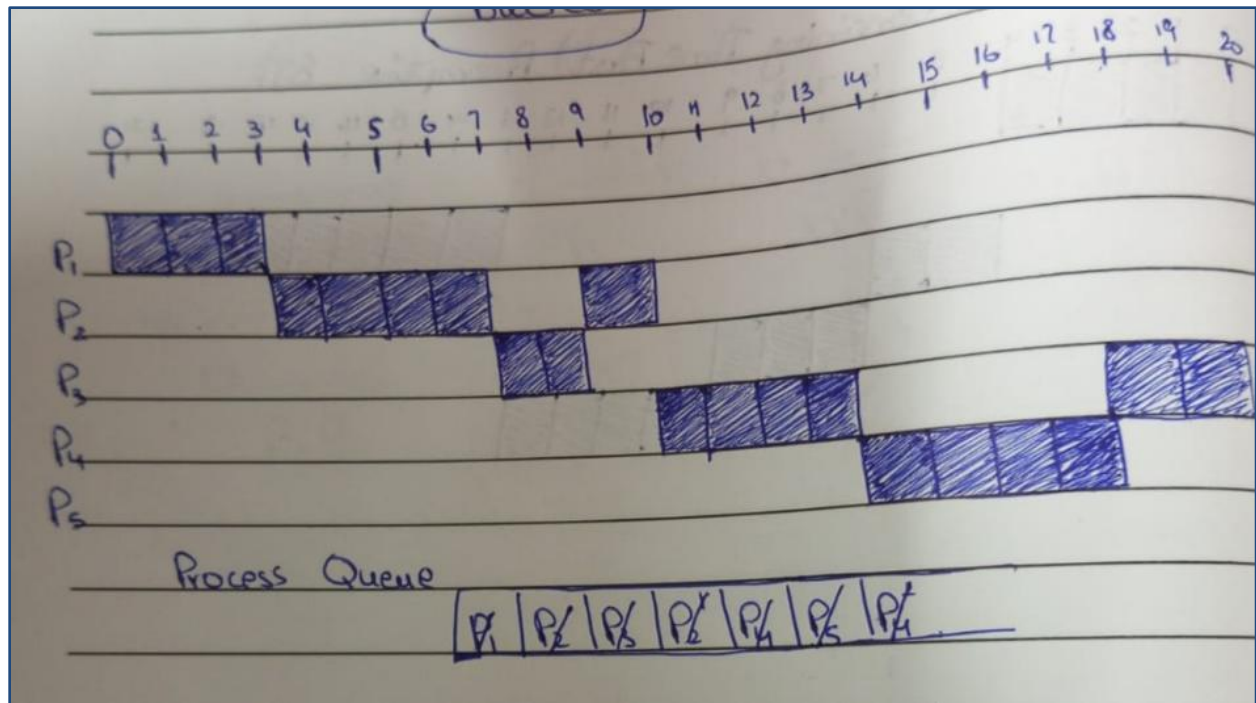
FCFS:



RR (Q=4):

RR \Rightarrow (Q=4)

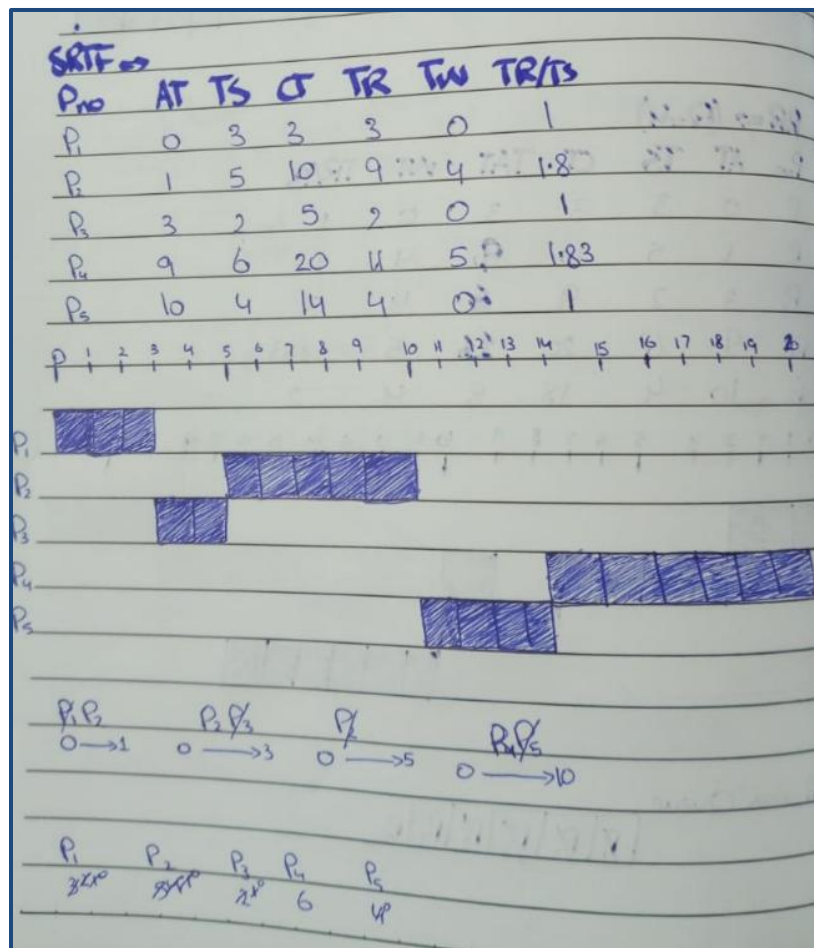
P _{no}	AT	TS	CT	TAT	WT	TR/TS
P ₁	0	3	3	3	0	1
P ₂	1	5	10	9 4	4	1.8
P ₃	3	2	9	6 4	4	3
P ₄	9	6	20	16 5	5	1.83
P ₅	10	4	18	8	4	2



SJF:



SRTF:



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

Process

Arrival Time

Service Time

P1

—

—

P2

—

—

P3



P4

P5

—

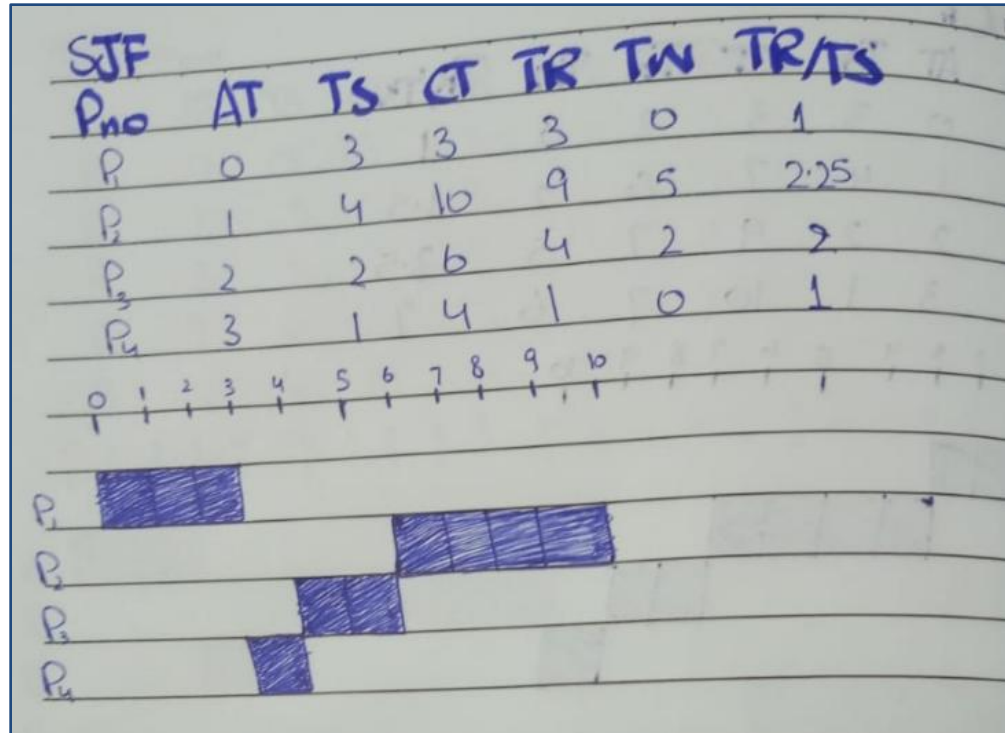
FCFS:



RR (Q=2):



SJF:



SRTF:

