



National Textile University

Department of Computer Science

Subject:

Operating System

Submitted to:

Sir Nasir Mehmood

Submitted by:

Akasha Fatima

Reg. number:

23-NTU-CS-FL-1132

Semester:

5th - A

Assignment-01

Section-A: Programming Tasks:

Task-01: 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.

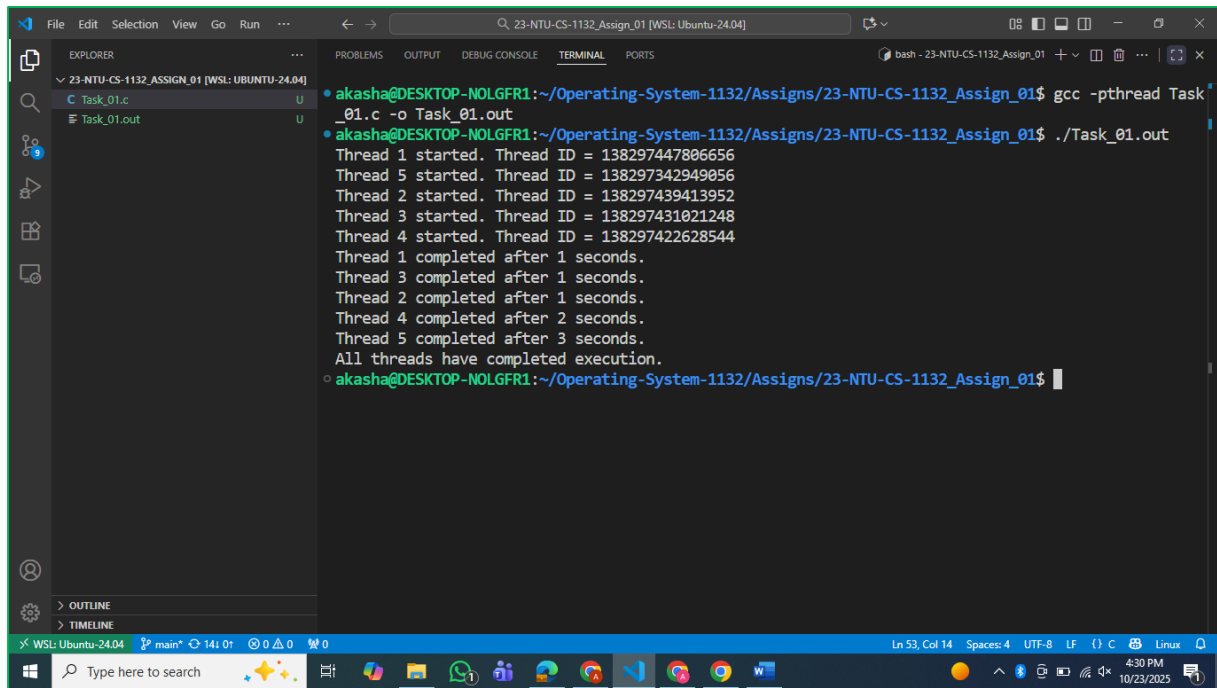
CODE:

```

1 // Name: Akasha Fatima
2 // Reg. No: 23-NTU-CS-1132
3 // Topic: Thread Information Display
4
5 // Write a program that creates 5 threads. Each thread should:
6 // Print its thread ID using `pthread_self()`.
7 // Display its thread number (1st, 2nd, etc.).
8 // Sleep for a random time between 1-3 seconds.
9 // Print a completion message before exiting.
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <pthread.h>
14 #include <unistd.h>
15 #include <time.h>
16
17 // Function to be executed by each thread
18 void* threadFunction(void* arg) {
19     int threadNum = *((int*)arg);
20     pthread_t threadID = pthread_self();
21     printf("Thread %d started. Thread ID = %lu\n", threadNum, (unsigned long)threadID);
22
23     // Sleep for a random time between 1 to 3 seconds
24     int sleepTime = (rand() % 3) + 1;
25     sleep(sleepTime);
26
27     printf("Thread %d completed after %d seconds.\n", threadNum, sleepTime);
28     return NULL;
29 }
30
31 int main() {
32     srand(time(NULL)); // Seed for random number generation
33
34     pthread_t t1, t2, t3, t4, t5;
35     int n1 = 1, n2 = 2, n3 = 3, n4 = 4, n5 = 5;
36
37     // Create 5 threads
38     pthread_create(&t1, NULL, threadFunction, (void*)&n1);
39     pthread_create(&t2, NULL, threadFunction, (void*)&n2);
40     pthread_create(&t3, NULL, threadFunction, (void*)&n3);
41     pthread_create(&t4, NULL, threadFunction, (void*)&n4);
42     pthread_create(&t5, NULL, threadFunction, (void*)&n5);
43
44     // Wait for all threads to complete
45     pthread_join(t1, NULL);
46     pthread_join(t2, NULL);
47     pthread_join(t3, NULL);
48     pthread_join(t4, NULL);
49     pthread_join(t5, NULL);
50
51     printf("All threads have completed execution.\n");
52     return 0;
53 }

```

Output:



```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Assigns/23-NTU-CS-1132_Assign_01$ gcc -pthread Task_01.c -o Task_01.out
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Assigns/23-NTU-CS-1132_Assign_01$ ./Task_01.out
Thread 1 started. Thread ID = 138297447806656
Thread 5 started. Thread ID = 138297342949056
Thread 2 started. Thread ID = 138297439413952
Thread 3 started. Thread ID = 138297431021248
Thread 4 started. Thread ID = 138297422628544
Thread 1 completed after 1 seconds.
Thread 3 completed after 1 seconds.
Thread 2 completed after 1 seconds.
Thread 4 completed after 2 seconds.
Thread 5 completed after 3 seconds.
All threads have completed execution.
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Assigns/23-NTU-CS-1132_Assign_01$
```

Task-02: Personalized Greeting Threads:

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.

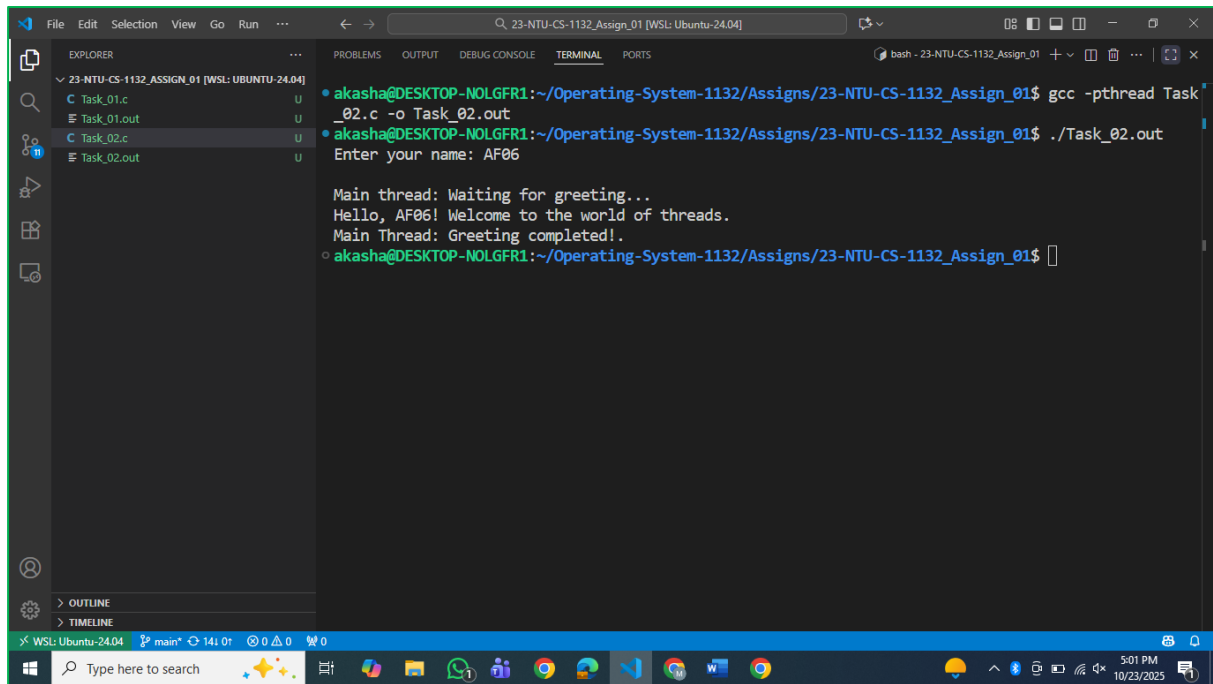
CODE:

```

1 // Name: Akasha Fatima
2 // Reg. No: 23-NTU-CS-1132
3 // Topic: Personalized Greeting Thread
4
5 // Write a C program that:
6 // Creates a thread that prints a personalized greeting message.
7 // The message includes the user's name passed as an argument to the thread.
8 // The main thread prints "Main thread: Waiting for greeting..." before joining the created thread.
9
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <pthread.h>
13 #include <string.h>
14 #include <unistd.h>
15
16 void* greetingFunction(void* arg) { // Function to be executed by the greeting thread
17     char* name = (char*)arg;
18     printf("Hello, %s! Welcome to the world of threads.\n", name);
19     return NULL;
20 }
21
22 int main() {
23     pthread_t greetingThread;
24     char name[50];
25
26     printf("Enter your name: "); // Get user's name
27     scanf(" %s", name);
28
29     pthread_create(&greetingThread, NULL, greetingFunction, (void*)name); // Create the greeting thread
30
31     printf("\nMain thread: Waiting for greeting...\n"); // Main thread message
32
33     pthread_join(greetingThread, NULL); // Wait for the greeting thread to complete
34
35     printf("Main Thread: Greeting completed!.\n");
36     return 0;
37 }

```

Output:



```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Assigns/23-NTU-CS-1132_Assign_01$ gcc -pthread Task_02.c -o Task_02.out
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Assigns/23-NTU-CS-1132_Assign_01$ ./Task_02.out
Enter your name: AF06

Main thread: Waiting for greeting...
Hello, AF06! Welcome to the world of threads.
Main Thread: Greeting completed!
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Assigns/23-NTU-CS-1132_Assign_01$
```

Task-03: 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 // Name: Akasha Fatima
2 // Reg. No: 23-NTU-CS-1132
3 // Topic: Number Info Thread
4
5 // Write a program that:
6 // Takes an integer input from the user.
7 // Creates a thread and passes this integer to it.
8 // The thread prints the number, its square, and cube.
9 // The main thread waits until completion and prints "Main thread: Work completed."
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <pthread.h>
14 #include <unistd.h>
15
16 void* numberInfoFunction(void* arg) { // Function to be executed by the number info thread
17     int number = *((int*)arg);
18     printf("Number: %d\n", number);
19     printf("Square: %d\n", number * number);
20     printf("Cube: %d\n", number * number * number);
21     return NULL;
22 }
23
24 int main() {
25     pthread_t numberInfoThread;
26     int number;
27
28     printf("Enter an integer: "); // Get integer input from the user
29     scanf("%d", &number);
30
31     pthread_create(&numberInfoThread, NULL, numberInfoFunction, (void*)&number); // Create the number info thread
32
33     pthread_join(numberInfoThread, NULL); // Wait for the number info thread to complete
34
35     printf("Main thread: Work completed.\n");
36     return 0;
37 }

```

Output:

```

akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Assigns/23-NTU-CS-1132_Assign_01$ gcc -pthread Task_03.c -o Task_03.out
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Assigns/23-NTU-CS-1132_Assign_01$ ./Task_03.out
Enter an integer: 6
Number: 6
Square: 36
Cube: 216
Main thread: Work completed.
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Assigns/23-NTU-CS-1132_Assign_01$

```

Task-04: Thread Returns Value:

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 // Name: Akasha Fatima
2 // Reg. No: 23-NTU-CS-1132
3 // Topic: Thread Return Value
4
5 // Write a program that creates a thread to compute the factorial of a number entered by the user.
6 // The thread should return the result using a pointer.
7 // The main thread prints the result after joining.
8
9 #include <stdio.h>
10 #include <stdlib.h>
11 #include <pthread.h>
12 #include <unistd.h>
13
14 // Function to compute factorial
15 void* factorialFunction(void* arg) {
16     int number = *((int*)arg);
17     long long* result = (long long*)malloc(sizeof(long long));
18     *result = 1;
19     for (int i = 1; i <= number; i++) {
20         *result *= i;
21     }
22     return (void*)result;
23 }
24
25 int main() {
26     pthread_t factorialThread;
27     int number;
28
29     // Get integer input from the user
30     printf("Enter a positive integer to compute its factorial: ");
31     scanf("%d", &number);
32
33     // Create the factorial thread
34     pthread_create(&factorialThread, NULL, factorialFunction, (void*)&number);
35
36     // Wait for the factorial thread to complete and get the result
37     long long* factorialResult;
38     pthread_join(factorialThread, (void**)&factorialResult);
39
40     // Print the result
41     printf("Factorial of %d is %lld\n", number, *factorialResult);
42
43     // Free allocated memory
44     free(factorialResult);
45
46     return 0;
47 }
```

Output:


```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Assigns/23-NTU-CS-1132_Assign_01$ gcc -pthread Task_04.c -o Task_04.out
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Assigns/23-NTU-CS-1132_Assign_01$ ./Task_04.out
Enter a positive integer to compute its factorial: 4
Factorial of 4 is 24
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Assigns/23-NTU-CS-1132_Assign_01$
```

Task-05: Structured 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.

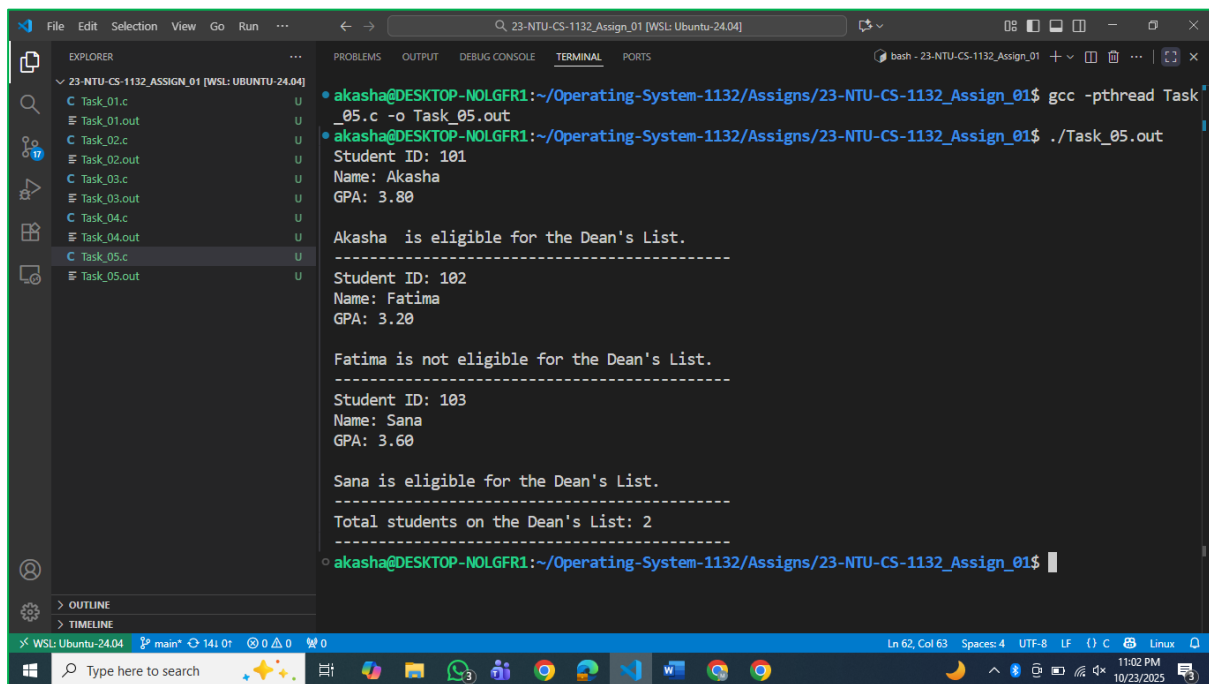
CODE:

```

1 // Name: Akasha Fatima
2 // Reg. No: 23-NTU-CS-1132
3 // Topic: Struct-Based Thread omunication
4
5 // Create a program that simulates a simple student database system.
6 // Define a struct: `typedef struct { int student_id; char name[50]; float gpa; } Student;`
7 // Create 3 threads, each receiving a different Student struct.
8 // Each thread prints student info and checks Dean's List eligibility (GPA ≥ 3.5).
9 // The main thread counts how many students made the Dean's List.
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <pthread.h>
14 #include <string.h>
15 #include <unistd.h>
16
17 typedef struct { // Define the Student struct
18     int student_id;
19     char name[50];
20     float gpa;
21 } Student;
22
23 // Function to be executed by each student thread
24 void* studentInfoFunction(void* arg) {
25     Student* student = (Student*)arg;
26     printf("Student ID: %d\n", student->student_id);
27     printf("Name: %s\n", student->name);
28     printf("GPA: %.2f\n", student->gpa);
29     if (student->gpa >= 3.5) {
30         printf("\n%s is eligible for the Dean's List.\n", student->name);
31         printf("-----\n");
32     } else {
33         printf("\n%s is not eligible for the Dean's List.\n", student->name);
34         printf("-----\n");
35     }
36     return NULL;
37 }
38
39 int main() {
40     pthread_t t1, t2, t3;
41     Student s1 = {101, "Akasha ", 3.8};
42     Student s2 = {102, "Fatima", 3.2};
43     Student s3 = {103, "Sana", 3.6};
44
45     // Create threads for each student
46     pthread_create(&t1, NULL, studentInfoFunction, (void*)&s1);
47     pthread_create(&t2, NULL, studentInfoFunction, (void*)&s2);
48     pthread_create(&t3, NULL, studentInfoFunction, (void*)&s3);
49
50     // Wait for all threads to complete
51     pthread_join(t1, NULL);
52     pthread_join(t2, NULL);
53     pthread_join(t3, NULL);
54
55     // Count Dean's List eligibility
56     int deanListCount = 0;
57     if (s1.gpa >= 3.5) deanListCount++;
58     if (s2.gpa >= 3.5) deanListCount++;
59     if (s3.gpa >= 3.5) deanListCount++;
60
61     printf("Total students on the Dean's List: %d\n", deanListCount);
62     printf("-----\n");
63     return 0;
64 }

```

Output:



```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Assigns/23-NTU-CS-1132_Assign_01$ gcc -pthread Task_05.c -o Task_05.out
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Assigns/23-NTU-CS-1132_Assign_01$ ./Task_05.out
Student ID: 101
Name: Akasha
GPA: 3.80
Akasha is eligible for the Dean's List.
-----
Student ID: 102
Name: Fatima
GPA: 3.20
Fatima is not eligible for the Dean's List.
-----
Student ID: 103
Name: Sana
GPA: 3.60
Sana is eligible for the Dean's List.
-----
Total students on the Dean's List: 2
-----
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/Assigns/23-NTU-CS-1132_Assign_01$
```

Section-B: Short Questions:

1. Answer all the questions briefly and clearly

2. Define an Operating System in one line.

Ans: An Operating System (OS) is a system software that acts as an intermediary link between the user and the hardware to manage the resources and the required services for a program execution.

3. What is the Primary function of the CPU scheduler?

Ans: The CPU scheduler selects one of the ready processes from the ready queue and allocates the CPU to it for execution to maximize the CPU use.

4. List any three states of a process.

Ans: The states of a process are as follows:

1. Ready
2. Running
3. Blocked
4. Waiting
5. Exiting

5. What is meant by a Process Control Box (PCB)?

Ans: A PCB is a metadata or also a data structure in the OS which is used to store the information about a process such as process ID, process parents ID, process state, CPU registers, memory details and scheduling information.

6. Differentiate between a process and a program.

Ans: The difference between a process and a program is as follows:

Aspect	Program	Process
Nature	The program is a passive set of instructions.	The “Process” is the active section of a program.
State	Not any specific run-time state.	Always have a specific state for each process at each level.

Identity	The program doesn't have a specific Id for identification.	Each single process in a program has a specific "Thread ID" .
Life Span	A program remained in execution until the file in which program is written is closed i.e., infinite time	The process has a finite time as the structure of process is as: Created(fork) → execute → join → terminate(exit)

7. What do you understand by "Context Switching"?

Ans: Context switch is a process or act of saving the state of the old process and to load the "saved state" of the new process to help the CPU to switch between the processes. It is represented by the Process Control Block of the respective process.

One major factor of the Context Switch is that system cannot be useful for any other work while switching the context of processes.

8. Define CPU utilization and throughput.

Ans: CPU Utilization:

- Percentage of time CPU stays busy i.e., keep the CPU busy as long as possible.
- In simple words, maximum time must be utilized rather than keeping the CPU in idle state.
- To achieve the high efficiency, CPU utilization must be **maximum**.

Throughput:

- Number of the processes that complete their execution per unit time.
- In simple words, maximum number of process executions = increase in throughput.
- For better performance, throughput time must be **maximum**.

9. What is the turnaround time of a process?

Ans: Turnaround time:

- It is the total time consumed for a process execution to perform any activity

i.e., waiting, ready, block, suspend, etc.

- It can be calculated as:

Turnaround Time = Original Time of Process Execution + Idle state of the process(waiting Time)

- To achieve the high efficiency, turnaround time of a process should be **minimum**.

10. How is waiting time calculated in process scheduling?

Ans: Waiting Time:

- The waiting time can be calculated as:

Waiting time = Turnaround Time - Original Time (total CPU execution time)

- It shows how long a process waits in the ready queue.

11. Define Response Time in CPU scheduling.

Ans: Response Time:

- The amount of time taken by a process from when the request is submitted until the first response it produced.
- For better performance, response time should be **minimum**.

12. What is the preemptive scheduling?

Ans: Preemptive Scheduling:

- Currently running process may be interrupted and CPU may move the next process to the ready state.
- Interruption may occur due to the arrival of new high-priority process or a blocked state in the ready queue.

13. What is non-preemptive scheduling?

Ans: Non-Preemptive Scheduling:

- In this scheduling type, a process remains in the working state continuously until it terminates or blocks itself for I/O operations.

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

Ans: The advantages of the Round Robin Scheduling algorithm are as follows:

- i. Each process gets an equal sharing of CPU time i.e., fair scheduling.
- ii. Provides good response time for interactive systems.
- iii. Simple and easy to implement using a queue.

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

Ans: Drawback of shortest job first algorithm is that:

- i. It may cause starvation(i.e., indefinite blocking) for long processes if short processes keep on arriving.
- ii. One more difficulty is to know or at least estimate the required processing time for each process.

16. Define CPU idle time.

Ans: CPU idles time is the time period in which the CPU remains unused (i.e., not performing any activity) because there are not any processes in the ready state.

17. State two common goals of CPU scheduling algorithms.

Ans: Two common goals of the CPU scheduling algorithms are:

- i. Maximizing the utilization of CPU and throughput as well.
- ii. Minimizing the turnaround time, waiting time and response time of the processes

18. List two possible reasons for process termination.

Ans: A process may be terminated because of:

- i. **Memory Available:** Process required more memory for execution than the provide one.
- ii. **Normal Completion:** Process executes an OS call for the completion of the service for which it is being run.
- iii. **Arithmetic Errors:** Process may try to perform invalid computation such

as dividing by zero.

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

Ans: The purpose of both system calls is as follows:

- ❖ **Exit():** Process execute last statement and then asks the OS to delete it by using exit() system call.
- ❖ **Wait():** This system call makes the Parent wait until the child process finishes.

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

Ans: The difference between shared memory and message-passing is as:

Aspect	Shared Memory	Message Passing
Communication	Each process shares a common memory space.	Each process has an individual memory space.
Speed	The process execution is fast due to same memory.	The process execution is slow due to the use of Kernel for message passing.
Best Suited For:	If the high speed communication needs to take place on the same system i.e., centralized machine.	If the communication is needed to make on distributed systems.

21. Differentiate between a Thread and a Process.

Ans: The difference between Thread and Process is as:

Aspect	Process	Thread
Definition	An independent program in execution with its own memory space.	A lightweight unit of execution within a process

		sharing the same memory space.
Memory	Each process has its own separate memory space.	Threads share the memory space of the parent process.
Creation Overhead	High overhead due to separate memory allocation.	Low overhead since threads share the resources.
Communication	IPC mechanisms like pipes, sockets or shared memory are needed.	Threads can communicate directly via shared variables but need synchronization.

22. Define multi-threading.

Ans: Multi-threading is the process of running more than one thread in a single process at a single time to perform multiple tasks simultaneously by the efficient utilization of CPU.

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

Ans: The difference between CPU-bound process and an I/O-bound process is as follows:

Aspect	CPU-bound process	I/O-bound process
Definition	Process that spends maximum time by doing CPU computations.	Process that spends maximum time by waiting for the I/O operations.
CPU Usage	Process uses CPU heavily for the calculations.	Process use CPU briefly, waits for the I/O services.
System Loads	Keeps CPU busy and the I/O idle.	Keeps I/O busy and the CPU idle to some extent.
Examples	Scientific calculations, data encryption	File transfer, database access, web browsing

24. What are the main responsibilities of the dispatcher?

Ans: The dispatcher gives control of the CPU to the new or selected process by performing context switching, setting the correct mode and then executing the process.

25. Define starvation and aging in the process scheduling.

Ans: Starvation:

Starvation occurs when a process waits indefinitely in the ready queue because other high priority processes keep on execution by the CPU. e.g., a process with low priority remains in the waiting state while all the other processes with the high priority keep on execution.

Aging:

Aging is a technique of gradually increasing the priority of a waiting process to prevent it from starvation. e.g., The OS may boost the priority of a long waiting process after a few seconds.

26. What is Time Quantum (or Time Slice)?

Ans: A time quantum is the specific i.e., fixed time given to each process in the Round Robin algorithm before switching to the next process.

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

Ans: In the Round Robin algorithm time slice or quantum time decides how long the process will run before switching to the next. If

- ❖ **Quantum time is too large:** System behaves like First Comes First Serves, scheduling algorithms. Responsive time increases for the interactive systems as the. e.g., If a process gets 10 seconds and other processes are at delay state, the system may experience delays.
- ❖ **Quantum time is too Small:** Causes frequent context switching and higher CPU overhead. CPU waste time between switching the processes rather than doing the real work.

28. Define the Turnaround Ratio (TR/TS).

Ans: The Turnaround Ratio is the ratio of the **Turnaround Time (TR)** to its **Service Time (TS)** i.e.,

$$\text{Turnaround Time} = \frac{\text{Turnaround Time (TR)}}{\text{Service Time (TS)}}$$

It shows how efficiently a process complete as compared to its original time of execution. The effectiveness is measured on the scale of 1.0.

29. What is the purpose of the ready queue?

Ans: The purpose of the ready queue is to manage all the processes that have been loaded in the memory and wait for the CPU time for execution.

30. Differentiate between CPU burst and the I/O burst.

Ans: CPU Burst: The total time is spent by a process in the running state, actively using the CPU for the computation.

I/O burst: The total time spent by a process waiting for an I/O operation i.e., waiting for the user's input.

31. Which scheduling algorithm is starvation free and why?

Ans: The **Round Robin** algorithm can be considered starvation free because every process regardless of the execution time, is assured by the quantum time that prevents the single process from the indefinite delayed.

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

Ans: The main steps involved in the process creation in the UNIX are as follows:

- i. **Fork():** Parent process used fork() system call to create a child process which has a unique identifier.
- ii. **Exec():** The child process uses exec() system call to load new process.
- iii. **Wait():** The parent process used wait() system call to wait until the child process terminate or complete.

33. Define zombie and orphan processes.

Ans: Zombie Process: A process that has its execution complete but still has an entry in the process table to allow the parent process to read the status.

Orphan Process: The process whose Parent process terminated before the child completes execution.

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

Ans: The difference between Priority scheduling and shortest job first is as follows:

Aspect	Priority Scheduling	Shortest Job First
Scheduling Basis	CPU is assigned to the process which has high priority.	CPU is assigned to the process which has minimum burst time
Focus	Mainly focus on the process's importance rather than execution time.	Focus on the execution time to reduce the waiting time.
Starvation	Lower priority process may starve due to higher process execution.	Larger processes may starve due to short processes frequent execution.
Uses	Used in real-time systems	Used to perform background activities such as print jobs.

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

Ans: Context switch time is the time taken by the CPU to switch the context i.e., saving the state of old process and loading the new process state. It is considered to be overhead because CPU does not perform any useful activity during the context switching.

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

Ans: The three levels of schedulers in an OS are:

- i. **Long-Term Scheduler (Job Selector):** Controls which process enters the main memory for the execution from the secondary storage.
- ii. **Medium-Term Scheduler (Swapper):** Manage swapping, suspensions of processes and main memory
- iii. **Short-Term Scheduler (CPU Scheduler):** Selects one of the ready processes for the CPU execution.

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

Ans: The difference between user mode and kernel mode is as follows:

Aspect	User Mode	Kernel Mode
Definitions	In this mode user-level applications are executed with a limited privilege.	In this mode OS kernel execute with full access to the hardware and system resources.
System Calls	User programs must make the system calls like write(), read(), execute() to request OS services.	Kernel executes system calls routine to provide requested services.
Crash Impact	If a program crashes in this mode, it does not affect the whole system.	If a process crash in this mode it may affect the whole Operating system.
Examples	Running applications such as MS Word, browser, or a C program etc.	Managing processes, system calls routine etc.

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.

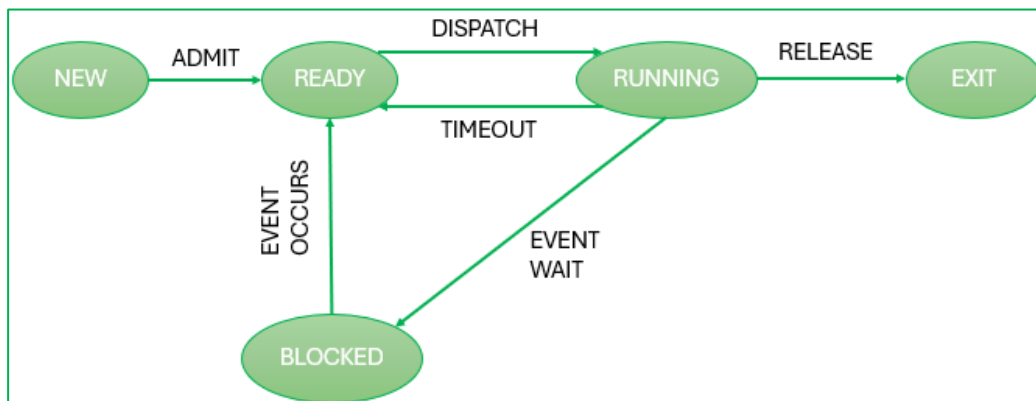
Ans: A process moves through various states during its life cycle. The five major states are:

1. **New:** The process is created and waiting to be admitted into main memory.
2. **Ready:** The process is loaded into memory and is waiting for CPU allocation.
3. **Running:** The process is currently executing on the CPU.
4. **Waiting:** The process is waiting for an I/O operation or an event to be completed.
5. **Terminated:** The process has finished execution and its resources are released.

State Transitions:

- **New → Ready:** The process is admitted into the ready queue.
- **Ready → Running:** The scheduler assigns the CPU.
- **Running → Waiting:** The process requests I/O or waits for an event.
- **Waiting → Ready:** The process finishes waiting and becomes ready again.
- **Running → Terminated:** Execution completes successfully.

Diagram:



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

Ans: Context Switch Overhead:

Context switch is a process or act of saving the state of the old process and to load the “saved state” of the new process to help the CPU to switch between the processes. It is represented by the Process Control Block of the respective process. It is considered to be overhead because CPU does not perform any useful activity during the context switching. Thus, it can be said that:

Higher the frequency of switching = greater the overhead and reduce the system’s efficiency.

Information Saved/Restored:

When a context switch takes place, it will save the state the of the current process and load the next process’s state. This information is primarily stored in the Process Control Box of the process which may include:

- Program counters
- CPU registers
- Stack pointers
- Process state and scheduling information

Example:

If a process A was executed in the memory and then paused for the execution of the process B, the CPU will save the Process As detail i.e., CPU registers, states etc, and then load the state of the process B.

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

Ans: The Process Control Block is the section which comprises on all the related information of every single process i.e., its state, CPU registers, pointers etc. Various components of PCB are as follows:

- i. Process Identifier:** Unique identifier for each process.

- ii. **Process State:** The current state of the process i.e., waiting, starting, exiting, etc.
- iii. **Program Counter:** The address of next instruction to be executed after the current process.
- iv. **CPU Registers:** Details of all the registers that must be used during the context switch of the processes i.e., index registers, general purpose registers, etc.
- v. **CPU Scheduling Information:** Provides information on the priority processes, CPU time usage and other scheduling parameters.
- vi. **Memory Management Information:** Provides details about the memory allocated to the respective process.
- vii. **Accounting Information:** Tracks CPU usage, execution time of the process and the ownership as well.
- viii. **I/O Status Information:** A list of I/O devices allocated to the current task and the opened files.

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

Ans: The difference between the all three schedulers is as follows:

Aspect	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
Function	Selects processes from secondary storage (job pool) and loads them into memory.	Selects one of the ready processes for CPU execution.	Temporarily removes and later reintroduces processes to control load.
Name	JOB Scheduler	CPU Scheduler	Process Swapping Scheduler

Speed	Slowest of all the schedulers.	Fastest, runs more frequently	Intermediate speed between Long and short term schedulers.
Multi-programming Concepts	Controls the degree of multiprogramming.	Gives less control over the degree of multiprogramming.	Reduces the degree of multiprogramming when memory is overloaded.
Decisions	Decide who enters the system.	Decide who runs next.	Decide who stays in the memory.
Examples	Admitting new user jobs into main memory.	Choosing the next ready process using algorithms like RR or SJF.	Swapping a process out to disk when memory is full.

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

Ans: The CPU scheduling criteria and their optimization goals are explained in the table below:

Criteria	Meaning	Example
Utilization	Maximum capacity of CPU must be utilized rather than keeping the CPU in idle state.	For efficient CPU working, Maximum utilization should be done.
Throughput	Maximum number of process causes increase in the throughput time.	Maximum throughput time is necessary for efficient execution.

Turnaround	Total time consumed by a process for its execution including each state i.e., ready, exit, etc.	Minimum turnaround time for the perfect execution.
Waiting	Idle time of the CPU i.e., remaining time other than the execution in the ready queue.	Minimum time is required for the efficient process execution.
Response	Time between process submissions and first response produced by the process.	For efficient CPU working, minimum response time is needed.

Section-D: CPU Scheduling Calculations:

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

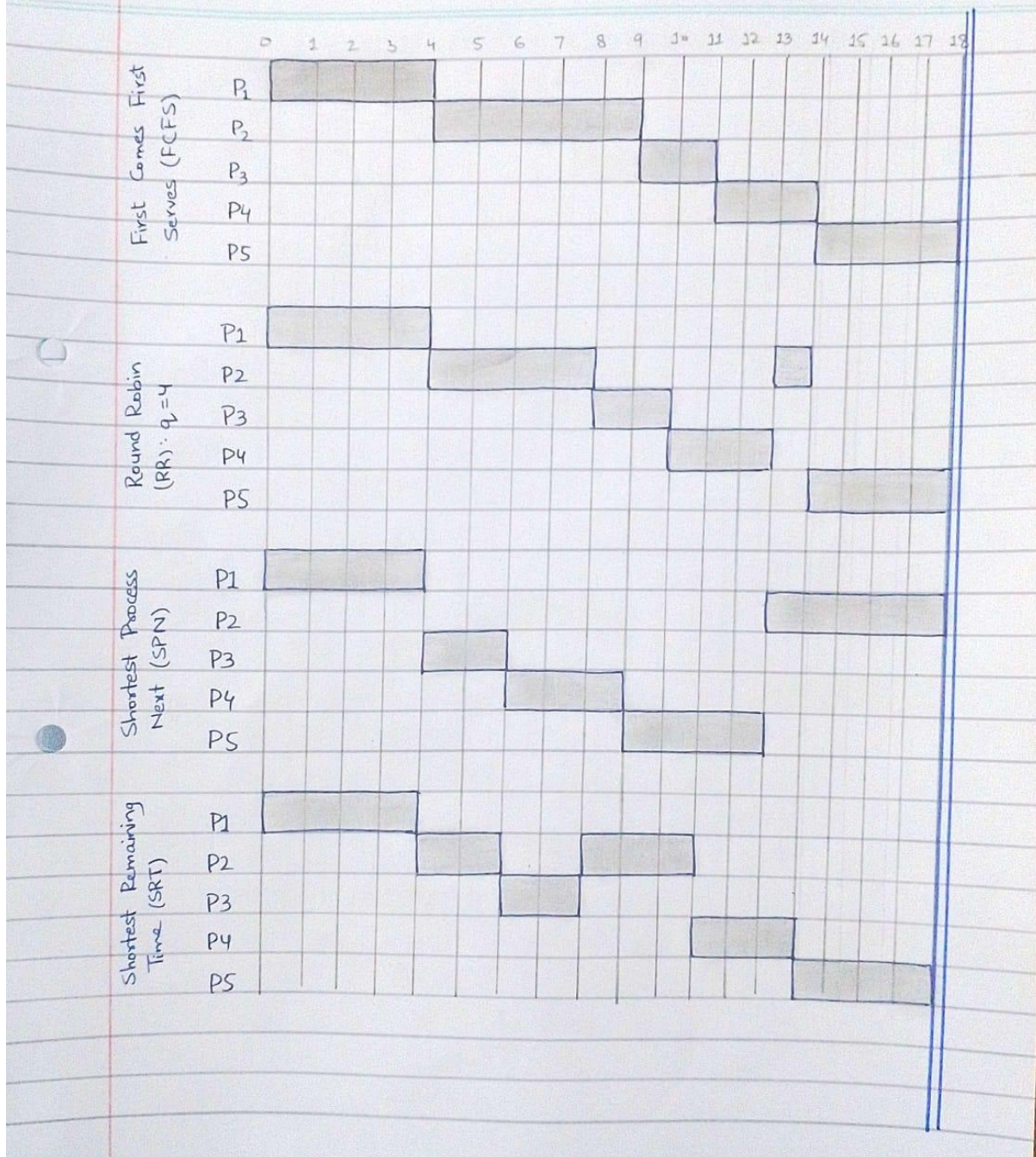
- Draw Gantt charts for FCFS, RR (Q=4), SJF, and SRTF.
- Compute Waiting Time, Turnaround Time, TR/TS ratio, and CPU Idle Time.
- Compare average values and identify which algorithm performs best.

Part A:

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

Solution:

CPU Idle Time = 0



First Comes First Serves(FCFS)						
Finish Time	4	9	11	14	18	Average
Arrival Time	0	2	4	6	9	
Service Time (TS)	4	5	2	3	4	3.6

Turnaround Time (TR) (Finish Time - Arrival Time)	4	7	7	8	9	7.0
Waiting Time (TR – TS)	0	2	5	5	5	3.4
Turnaround Ratio (TR/TS)	1	1.4	3.5	2.7	2.3	1.94

Round Robin (RR): q = 4						
Finish Time	4	14	10	13	18	Average
Arrival Time	0	2	4	6	9	
Service Time (TS)	4	5	2	3	4	3.6
Turnaround Time (TR) (Finish Time - Arrival Time)	4	12	6	7	9	7.60
Waiting Time (TR – TS)	0	7	4	4	5	4
Turnaround Ratio (TR/TS)	1	2.4	3	2.3	2.3	2.2

Shortest Job First (STF)						
Finish Time	4	18	6	9	13	Average
Arrival Time	0	2	4	6	9	
Service Time (TS)	4	5	2	3	4	3.6
Turnaround Time (TR) (Finish Time - Arrival Time)	4	16	2	3	4	7.60
Waiting Time (TR – TS)	0	11	0	0	0	2.2
Turnaround Ratio (TR/TS)	1	3.2	1	1	1	1.44

Shortest Remaining Job First (SRTF)						
Finish Time	4	11	8	14	18	Average
Arrival Time	0	2	4	6	9	
Service Time (TS)	4	5	2	3	4	3.6
Turnaround Time (TR)	4	9	4	8	9	7.60

(Finish Time - Arrival Time)						
Waiting Time (TR – TS)	0	4	2	5	5	3.2
Turnaround Ratio (TR/TS)	1	1.8	2	2.7	2.3	1.96

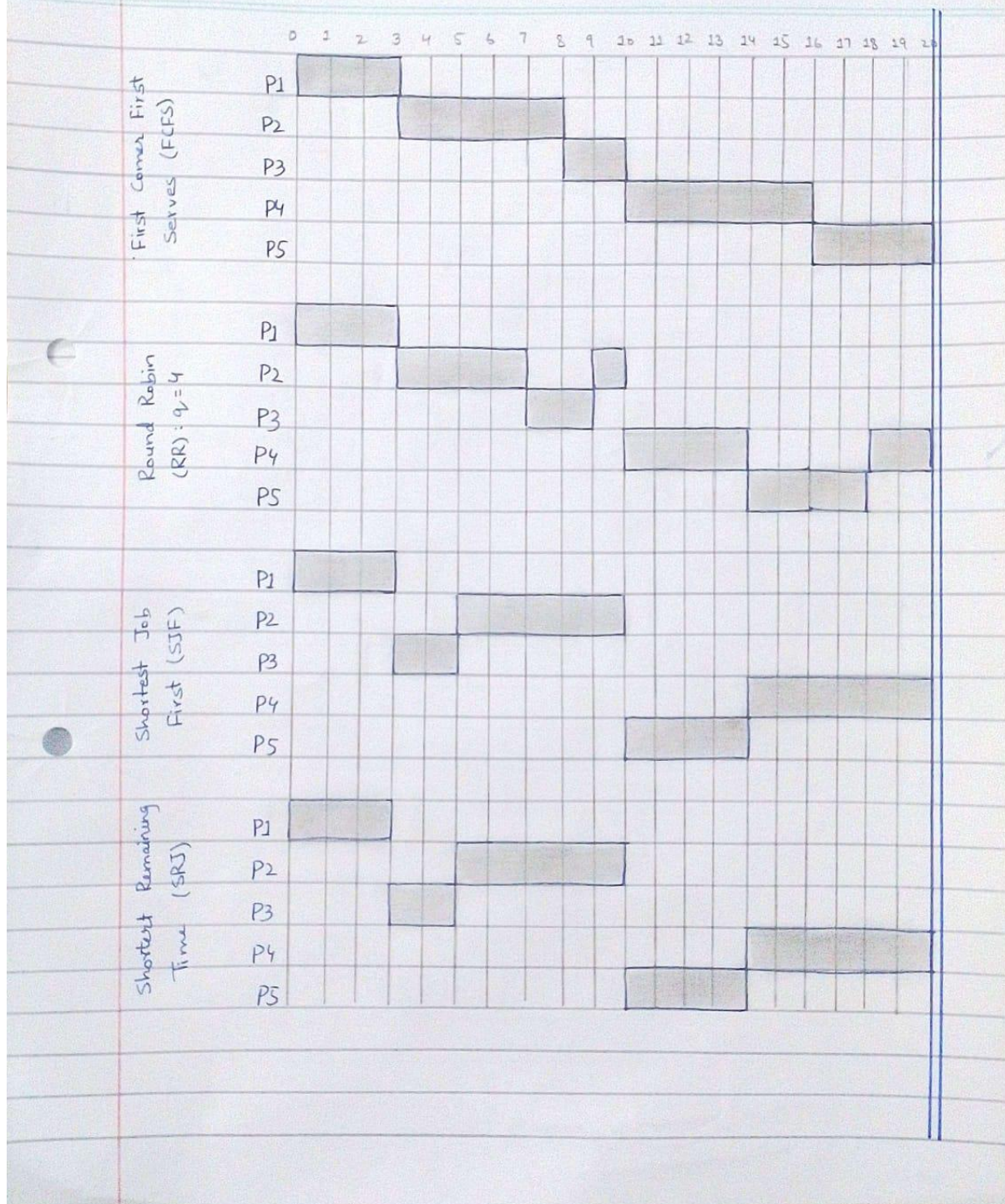
Best Algorithm: By the above average calculations of all the above algorithms we come to know that **Shortest Job First (SJF)** is the best algorithm for this process as it allocates the CPU to the processes having shortest jobs. **RR** gives higher average times while **FCFS** is also not so optimal.

Part B:

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

Solution:

CPU Idle Time = 0



First Comes First Serves(FCFS)						
Finish Time	3	8	10	16	20	Average
Arrival Time	0	1	3	9	10	

Service Time (TS)	3	5	2	6	4	4.0
Turnaround Time (TR) (Finish Time - Arrival Time)	3	7	7	7	10	6.8
Waiting Time (TR – TS)	0	2	5	1	6	2.8
Turnaround Ratio (TR/TS)	1	1.4	3.5	1.2	2.5	1.92

Round Robin (RR): q =4						
Finish Time	3	10	9	20	18	Average
Arrival Time	0	1	3	9	10	
Service Time (TS)	3	5	2	6	4	4.0
Turnaround Time (TR) (Finish Time - Arrival Time)	3	9	6	11	8	7.4
Waiting Time (TR – TS)	0	4	4	5	4	3.4
Turnaround Ratio (TR/TS)	1	1.8	3	1.83	2	1.93

Shortest Job First (SJF)						
Finish Time	3	10	5	20	14	Average
Arrival Time	0	1	3	9	10	
Service Time (TS)	3	5	2	6	4	4.0
Turnaround Time (TR) (Finish Time - Arrival Time)	3	9	2	11	4	5.8
Waiting Time (TR – TS)	0	4	0	5	0	1.8
Turnaround Ratio (TR/TS)	1	1.8	1	1.83	1	1.33

Shortest Remaining Job First (SRJF)						
Finish Time	3	10	5	20	14	Average
Arrival Time	0	1	3	9	10	
Service Time (TS)	3	5	2	6	4	4.0

Turnaround Time (TR) (Finish Time - Arrival Time)	3	9	2	11	4	5.8
Waiting Time (TR – TS)	0	4	0	5	0	1.8
Turnaround Ratio (TR/TS)	1	1.8	1	1.83	1	1.33

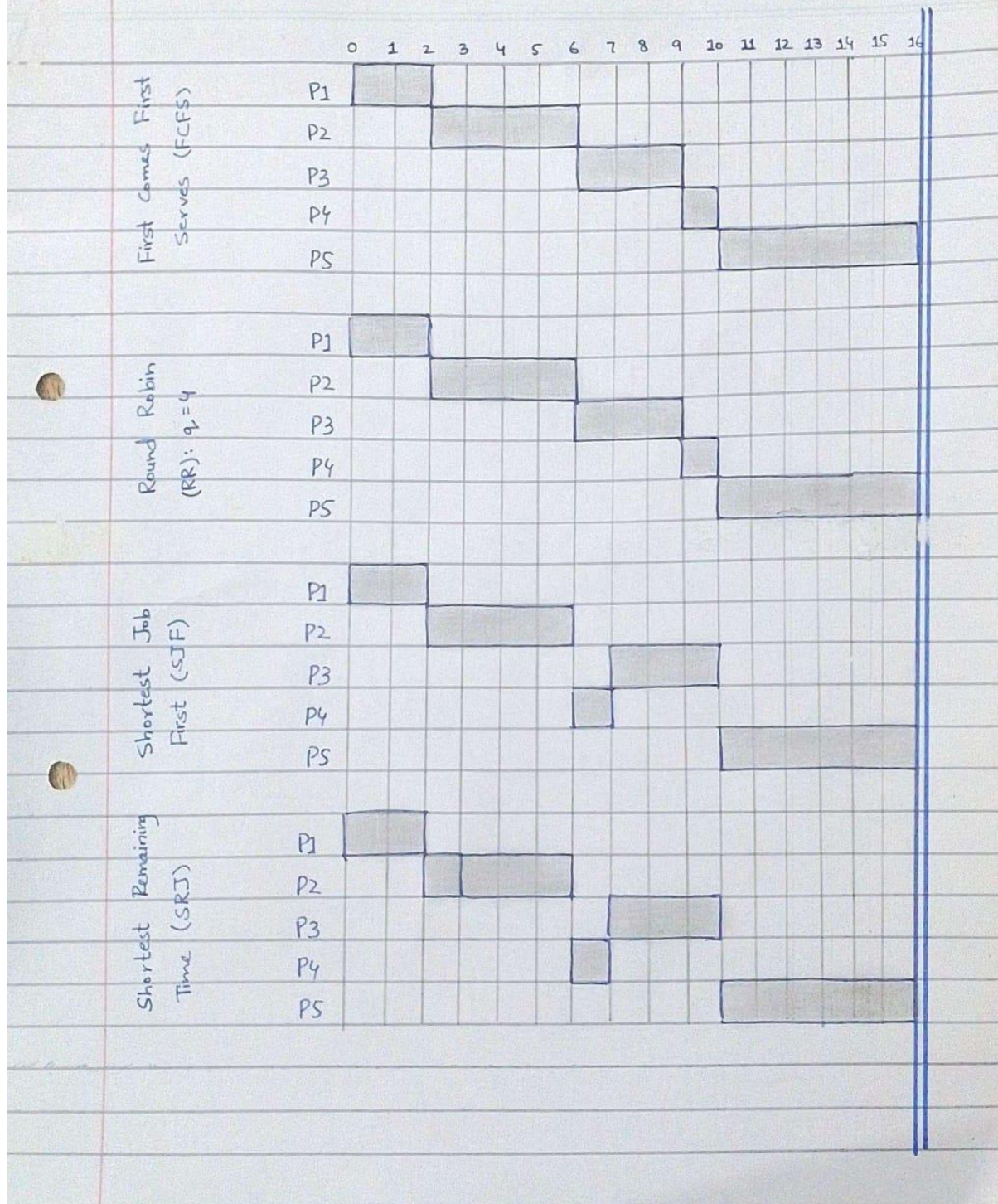
Best Algorithm: By the above average calculations of all the above algorithms we come to know that **Shortest Job First (SJF)** and **Shortest Remaining Job First (SRJF)** are the best algorithms for this process as it gives the lowest average turnaround and waiting times. **FCFS** is simple but least efficient and **Round Robin** increases the waiting slightly.

Part C:

Process	Arrival Time	Service Time
P1	0	2
P2	1	4
P3	5	3
P4	6	1
P5	8	6

Solution:

CPU Idle Time = 0



CS Scanned with CamScanner

First Comes First Serves(FCFS)						
Finish Time	2	6	9	10	16	Average

Arrival Time	0	1	5	6	8	
Service Time (TS)	2	4	3	1	6	3.2
Turnaround Time (TR) (Finish Time - Arrival Time)	2	5	4	4	8	4.6
Waiting Time (TR – TS)	0	1	1	3	2	1.4
Turnaround Ratio (TR/TS)	1	1.25	1.3	4	1.22	1.78

Round Robin (RR) : q = 4						
Finish Time	2	6	9	10	16	Average
Arrival Time	0	1	5	6	8	
Service Time (TS)	2	4	3	1	6	3.2
Turnaround Time (TR) (Finish Time - Arrival Time)	2	5	4	4	8	4.6
Waiting Time (TR – TS)	0	1	1	3	2	1.4
Turnaround Ratio (TR/TS)	1	1.25	1.3	4	1.22	1.78

Shortest Job First (SJF)						
Finish Time	2	6	10	7	16	Average
Arrival Time	0	1	5	6	8	
Service Time (TS)	2	4	3	1	6	3.2
Turnaround Time (TR) (Finish Time - Arrival Time)	2	5	5	1	8	4.2
Waiting Time (TR – TS)	0	1	2	0	2	1
Turnaround Ratio (TR/TS)	1	1.25	1.67	1	1.33	1.25

Shortest Remaining Job First (SRJF)						
Finish Time	2	6	10	7	16	Average

Arrival Time	0	1	5	6	8	
Service Time (TS)	2	4	3	1	6	3.2
Turnaround Time (TR) (Finish Time - Arrival Time)	2	5	5	1	8	4.2
Waiting Time (TR – TS)	0	1	2	0	2	1
Turnaround Ratio (TR/TS)	1	1.25	1.67	1	1.33	1.25

Best Algorithm: By the above average calculations of all the above algorithms we come to know that **Shortest Job First (SJF)** and **Shortest Remaining Job First (SRJF)** are the best algorithms for this process as it gives the lowest average turnaround and waiting times. **FCFS** is simple but least efficient and **Round Robin** increases the waiting slightly.