National Textile University
Department of Computer Science

Name:

Gulam Rasool

Reg-No:

23-NTU-CS-FL-1159

SECTION:

BSSE-(A)

SEMESTER:

5TH

Subject:

Operating System

Assignment No:

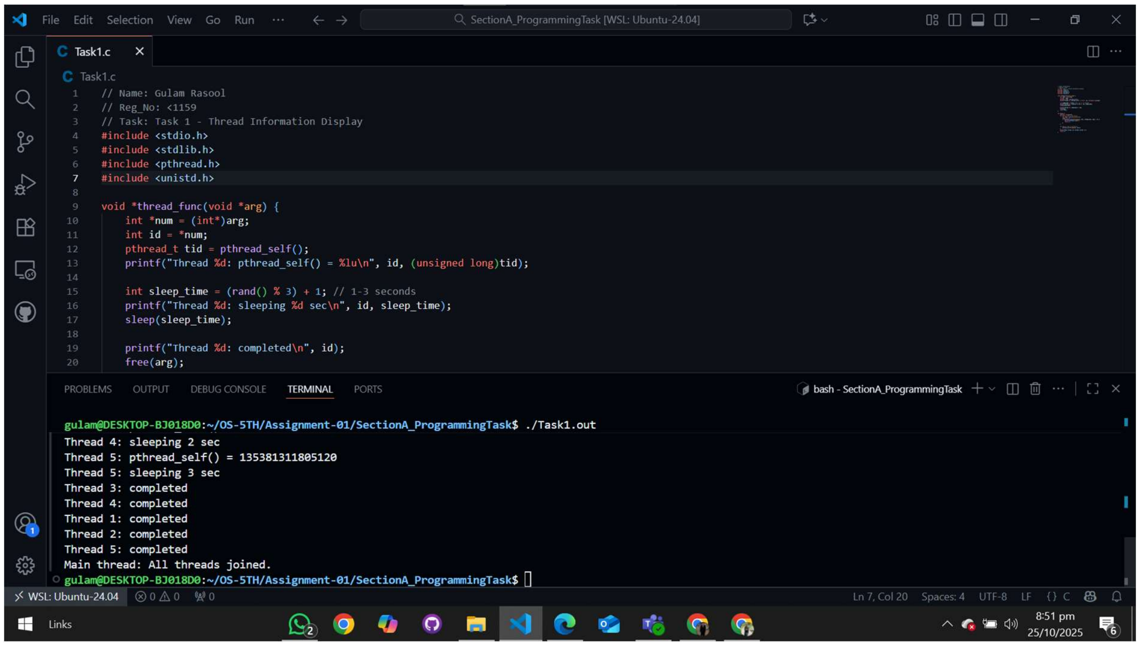
01

Submitted to:

Sir Nasir Mehmood

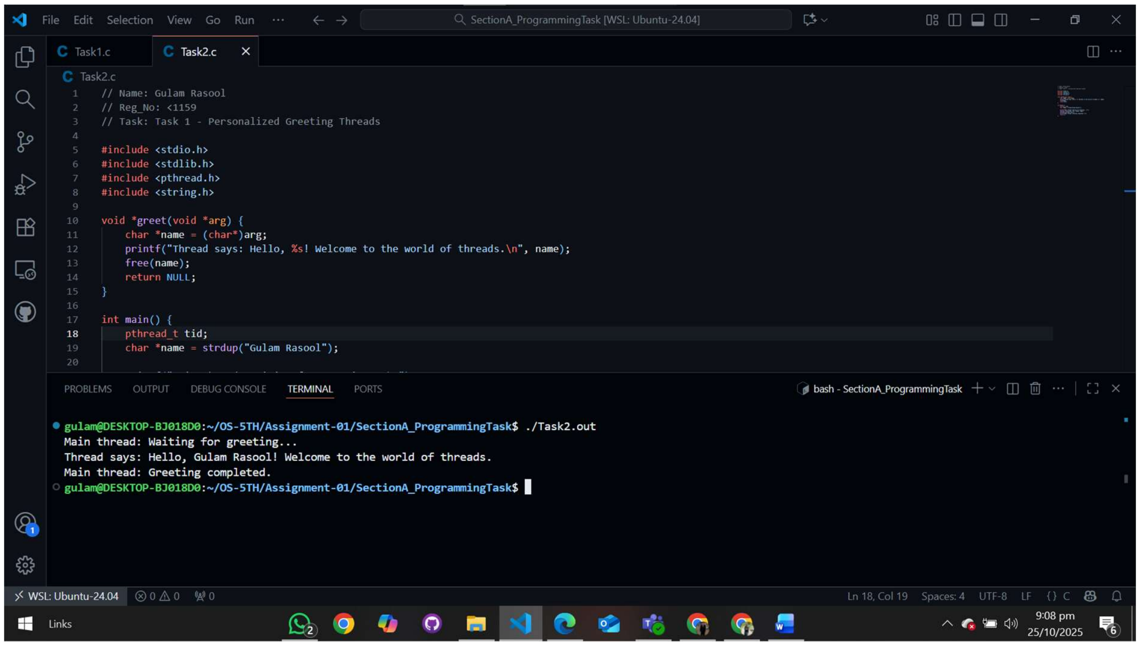
Section-A: Programming Tasks

Task 1 – Thread Information Display


```c
// Name: Gulam Rasool
// Reg_No: <1159
// Task: Task 1 - Thread Information Display
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

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

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

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

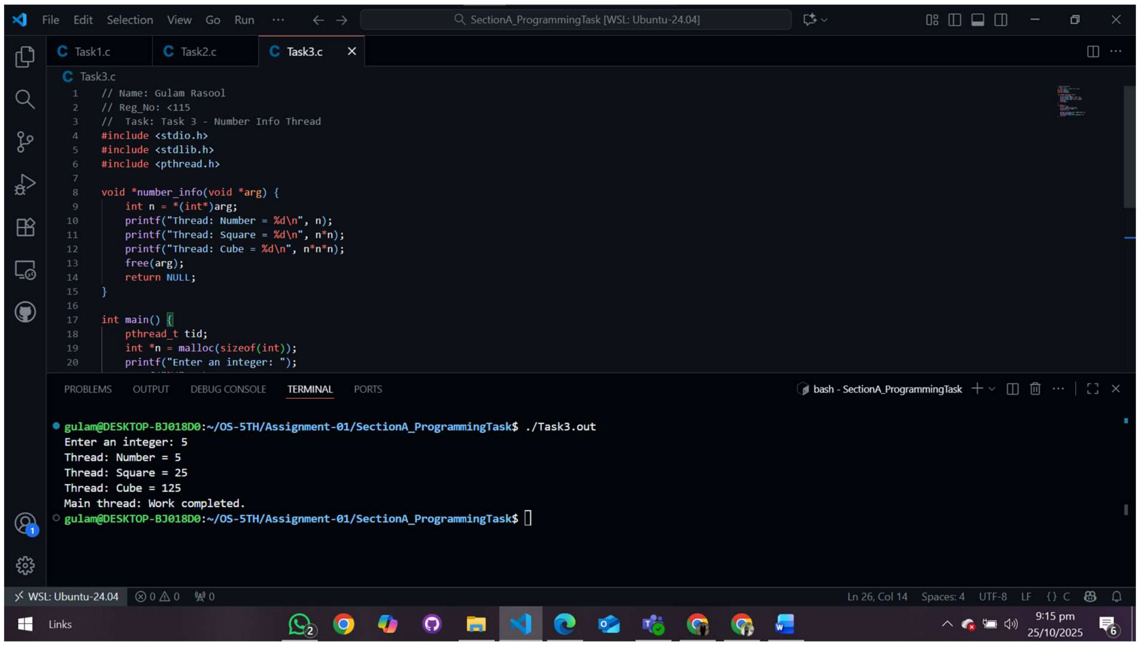
```
gulam@DESKTOP-BJ018D0:~/OS-5TH/Assignment-01/SectionA_ProgrammingTask$ ./Task1.out
Thread 4: sleeping 2 sec
Thread 5: pthread_self() = 135381311805120
Thread 5: sleeping 3 sec
Thread 3: completed
Thread 4: completed
Thread 1: completed
Thread 2: completed
Thread 5: completed
Main thread: All threads joined.
gulam@DESKTOP-BJ018D0:~/OS-5TH/Assignment-01/SectionA_ProgrammingTask$ 
```

Task 2 – Personalized Greeting Thread


```c
// Name: Gulam Rasool
// Reg_No: <1159
// Task: Task 1 - Personalized Greeting Threads

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

void *greet(void *arg) {
    char *name = (char*)arg;
    printf("Thread says: Hello, %s! Welcome to the world of threads.\n", name);
    free(name);
    return NULL;
}

int main() {
    pthread_t tid;
    char *name = strdup("Gulam Rasool");

```

```
gulam@DESKTOP-BJ018D0:~/OS-5TH/Assignment-01/SectionA_ProgrammingTask$ ./Task2.out
Main thread: Waiting for greeting...
Thread says: Hello, Gulam Rasool! Welcome to the world of threads.
Main thread: Greeting completed.
gulam@DESKTOP-BJ018D0:~/OS-5TH/Assignment-01/SectionA_ProgrammingTask$ 
```

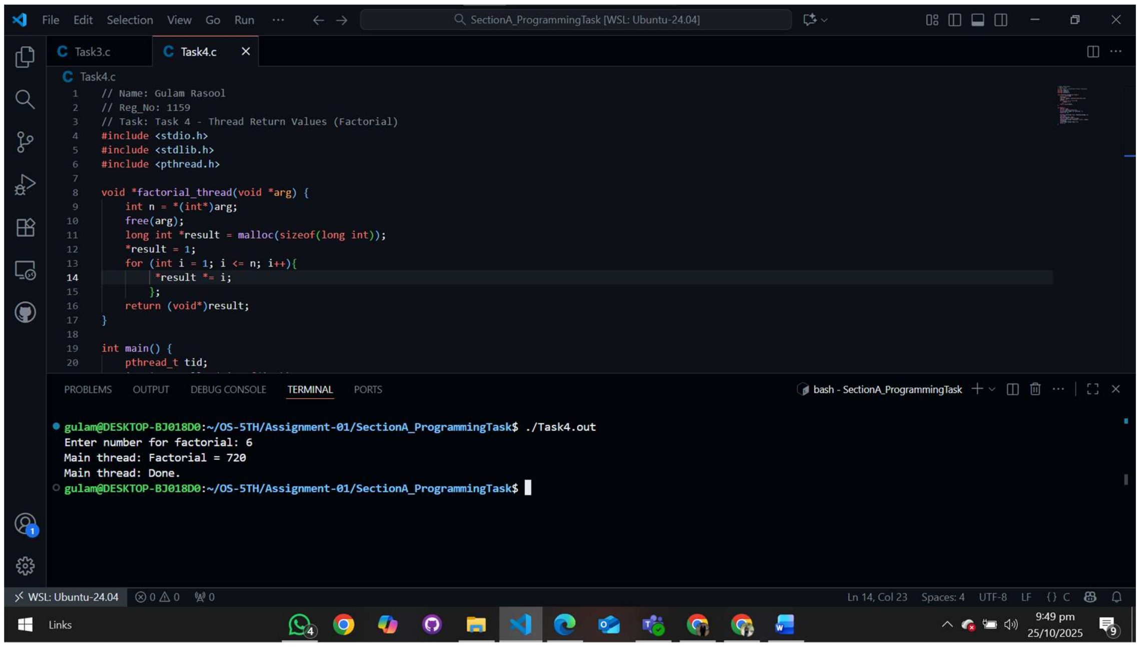
Task 3 – Number Info Thread



```c
// Name: Gulam Rasool
// Reg_No: <115
//  Task: Task 3 - Number Info Thread
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

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

int main() {
    pthread_t tid;
    int *n = malloc(sizeof(int));
    printf("Enter an integer: ");
```

```
gulam@DESKTOP-BJ018D0:~/OS-5TH/Assignment-01/SectionA_ProgrammingTask$ ./Task3.out
Enter an integer: 5
Thread: Number = 5
Thread: Square = 25
Thread: Cube = 125
Main thread: Work completed.
gulam@DESKTOP-BJ018D0:~/OS-5TH/Assignment-01/SectionA_ProgrammingTask$
```

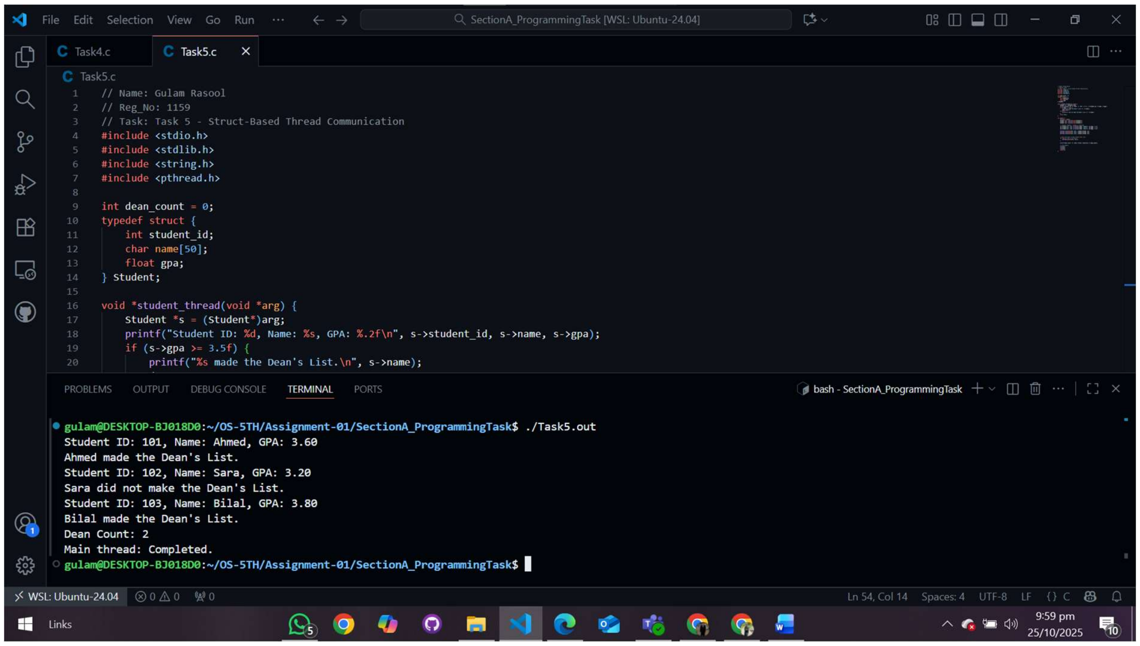
Task 4 – Thread Return Values



```c
// Name: Gulam Rasool
// Reg_No: 1159
// Task: Task 4 - Thread Return Values (Factorial)
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

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

int main() {
    pthread_t tid;
```

```
gulam@DESKTOP-BJ018D0:~/OS-5TH/Assignment-01/SectionA_ProgrammingTask$ ./Task4.out
Enter number for factorial: 6
Main thread: Factorial = 720
Main thread: Done.
gulam@DESKTOP-BJ018D0:~/OS-5TH/Assignment-01/SectionA_ProgrammingTask$
```

Task 5 – Struct-Based Thread Communication



```c
// Name: Gulam Rasool
// Reg_No: 1159
// Task: Task 5 - Struct-Based Thread Communication
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>

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

void *student_thread(void *arg) {
    Student *s = (Student*)arg;
    printf("Student ID: %d, Name: %s, GPA: %.2f\n", s->student_id, s->name, s->gpa);
    if (s->gpa >= 3.5f) {
        printf("%s made the Dean's List.\n", s->name);
```

```
gulam@DESKTOP-BJ018D0:~/OS-5TH/Assignment-01/SectionA_ProgrammingTask$ ./Task5.out
Student ID: 101, Name: Ahmed, GPA: 3.60
Ahmed made the Dean's List.
Student ID: 102, Name: Sara, GPA: 3.20
Sara did not make the Dean's List.
Student ID: 103, Name: Bilal, GPA: 3.80
Bilal made the Dean's List.
Dean Count: 2
Main thread: Completed.
gulam@DESKTOP-BJ018D0:~/OS-5TH/Assignment-01/SectionA_ProgrammingTask$
```

Section-B: Short Questions

1.Define an Operating System in a single line.

An operating system is system software that manages the computer's hardware and software resources. It acts as a bridge between the user and the hardware, making sure all programs run smoothly.

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

The CPU scheduler's main job is to decide which process should run next on the CPU. It improves CPU efficiency by selecting the best process according to a scheduling algorithm.

3. List any three states of a process.

A process mainly goes through three important states: Ready, Running, and Waiting (or Blocked). These states represent the process's position in its life cycle.

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

A PCB is a data structure used by the operating system to store information about a process such as its ID, state, CPU registers, and memory details. It helps the OS manage and switch between processes.

5. Differentiate between a process and a program.

A program is just a set of instructions stored on disk, while a process is the running instance of that program in memory. When a program starts execution, it becomes a process.

6. What do you understand by context switching?

Context switching is when the CPU changes from one process to another. It saves the current process's state and loads another process's state so that both can continue properly.

7. Define CPU utilization and throughput.

CPU utilization means how much time the CPU stays busy doing useful work. Throughput refers to how many processes are completed in a given time period.

8. What is the turnaround time of a process?

Turnaround time is the total time taken from when a process arrives until it completes execution. It includes waiting, execution, and I/O times.

9. How is waiting time calculated in process scheduling?

Waiting time is the total time a process spends in the ready queue before getting CPU time. It can be found by subtracting burst time from turnaround time.

10. Define response time in CPU scheduling.

Response time is the time between process submission and its first response or output. It’s important for user-interactive systems.

11. What is preemptive scheduling?

In preemptive scheduling, the CPU can be taken away from a running process to give another process a chance to run. It helps achieve better responsiveness.

12. What is non-preemptive scheduling?

In non-preemptive scheduling, once a process starts using the CPU, it runs until it finishes or waits for I/O. No other process can interrupt it in between.

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

Round Robin gives equal CPU time to all processes and avoids starvation. It is suitable for time-sharing systems and provides quick responses for small tasks.

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

SJF can cause starvation for long processes because shorter jobs keep getting preference. It also requires knowing the exact burst time in advance.

15. Define CPU idle time.

CPU idle time is the period when the CPU is not executing any process. This usually happens when no process is ready to run.

16. State two common goals of CPU scheduling algorithms.

The main goals are to maximize CPU utilization and throughput while minimizing waiting and turnaround times for processes.

17. List two possible reasons for process termination.

A process may terminate normally after completing its job or abnormally due to an error, such as memory overflow or being killed by the user/OS.

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

The exit() call ends a process and returns its status to the OS. The wait() call allows the parent process to pause until its child process finishes execution.

19. Differentiate between shared memory and message-passing models of IPC.

In shared memory, processes communicate by using a common memory area. In message passing, they exchange data through system calls like send and receive.

20. Differentiate between a thread and a process.

A thread is a smaller part of a process that shares memory and resources with other threads. A process, on the other hand, has its own memory and system resources.

21. Define multithreading.

Multithreading means running multiple threads within a single process to perform different tasks at the same time, improving CPU usage and performance.

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

A CPU-bound process mainly uses the processor for computation, while an I/O-bound process spends most time waiting for input/output operations to complete.

23. What are the main responsibilities of the dispatcher?

The dispatcher gives CPU control to the process chosen by the scheduler. It performs context switching, changes CPU mode, and jumps to the correct process location.

24. Define starvation and aging in process scheduling.

Starvation happens when a process never gets CPU time due to low priority. Aging is a technique that gradually increases a waiting process's priority to prevent starvation.

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

A time quantum is the fixed time a process is allowed to run before the CPU switches to another process. It's used in Round Robin scheduling.

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

If too large, it behaves like FCFS and increases waiting time. If too small, too many context switches occur, wasting CPU time.

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

Turnaround ratio is the ratio of turnaround time to service time. It shows how much total time a process took compared to its actual burst time.

28. What is the purpose of a ready queue?

The ready queue holds all processes that are loaded in memory and waiting for CPU time. The scheduler picks the next process from this queue.

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

A CPU burst is when a process actively uses the CPU to execute instructions, while an I/O burst occurs when it waits for input or output operations.

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

Round Robin is starvation-free because every process gets an equal time share in a cyclic order, so none are ignored.

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

In UNIX, fork() creates a new child process, then exec() replaces its memory with a new program, and finally the parent can use wait() to collect its status.

32. Define zombie and orphan processes.

A zombie process is one that has finished execution but still remains in the process table. An orphan process is a child whose parent has ended before it.

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

Priority scheduling selects the process with the highest priority, while SJF selects the one with the shortest CPU burst time. Both aim to improve efficiency.

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

Context switch time is the time spent saving and loading process information during switching. It's considered overhead because no actual processing work happens then.

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

Long-term scheduler controls job admission to the system. Medium-term handles swapping processes in and out of memory. Short-term decides which process runs next on the CPU.

36. Differentiate between User Mode and Kernel Mode in an OS.

User mode is where normal user programs run with limited access. Kernel mode is a privileged mode used by the OS to control hardware and perform critical operations.

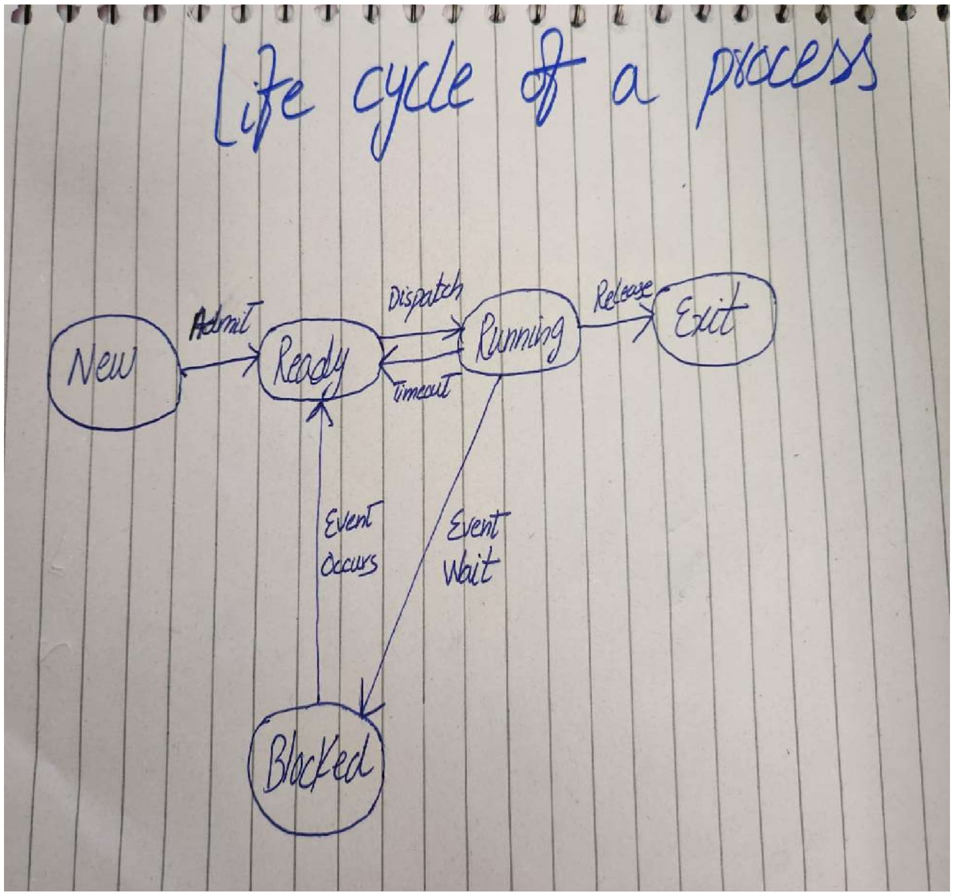
Section-C: Technical / Analytical Questions

1. Life cycle of a process

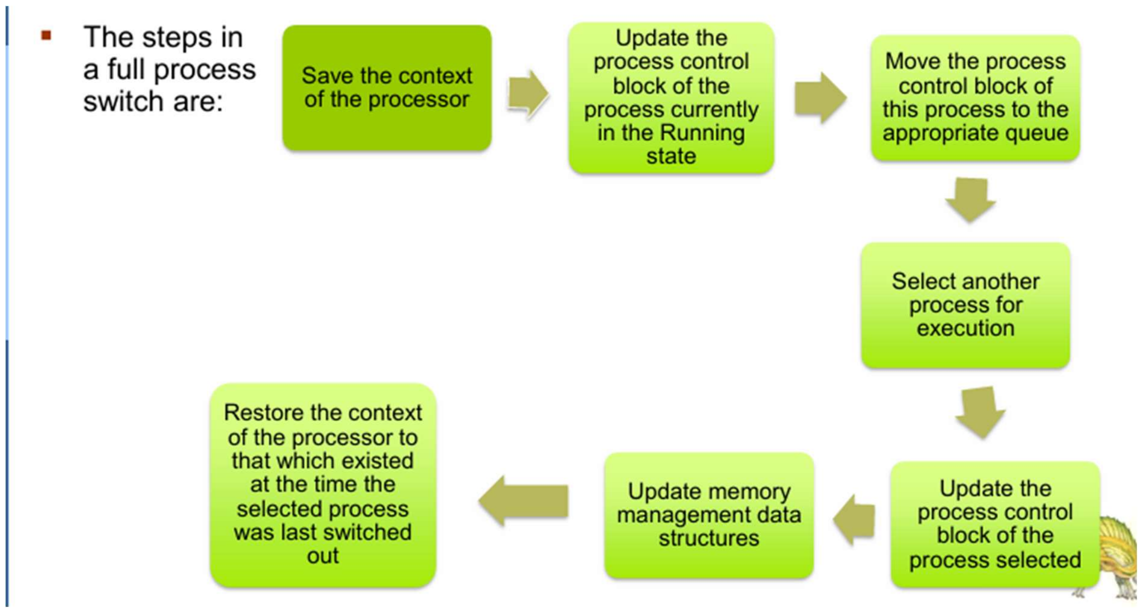
- **New**: Process is being created.
- **Ready**: Process is ready to run and waiting in ready queue.
- **Running**: CPU is executing the process.
- **Waiting/Blocked**: Process waits for I/O or event.
- **Terminated**: Process finished or killed.

Transitions:

- New → Ready (after creation).
- Ready → Running (dispatcher selects it).
- Running → Waiting (if process does I/O or needs event).
- Waiting → Ready (when I/O completes).
- Running → Ready.
- Running → Terminated (process finishes).



2. Context switch overhead — note and what must be saved/restored


The steps in a full process switch are:
Save the context of the processor
Update the process control block of the process currently in the Running state
Move the process control block of this process to the appropriate queue
Select another process for execution
Update the process control block of the process selected
Update memory management data structures
Restore the context of the processor to that which existed at the time the selected process was last switched out

Context switch overhead is the extra time the CPU spends saving the state of the current process and loading the state of the next process , during which no productive work of those processes is done. Minimizing unnecessary switches improves performance.

What must be saved/restored:

- Context of Processor.
- Process state.
- Memory management.
- Scheduling.

3. Components of a Process Control Block (PCB)

I. **Process state:** running, waiting, etc.

II. **Program counter:** location of instruction to next execute

III. **CPU registers:** contents of all process centric registers

IV. **CPU scheduling information:** priorities, scheduling queue pointers

V. **Memory-management information:** memory allocated to the process

VI. **Accounting information:** CPU used, clock time elapsed since start, time limits

VII. **I/O status information:** I/O devices allocated to process, list of open files

4. Long-Term, Medium-Term, Short-Term Schedulers

	Long-Term Scheduler (Job Scheduler)	**Short-Term Scheduler (CPU Scheduler)**	**Medium-Term Scheduler (Swapper)**
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.
Speed	Slowest of all schedulers**.**	Fastest: runs most frequently.	Intermediate speed between long- and short-term schedulers.
Presence in Systems	Barely present or non existent in time sharing systems.	Always present; key component of time-sharing systems.	Used mainly in time sharing systems for swapping.
Key Operation	Admits new jobs into memory.	Dispatches ready processes to CPU.	Suspends and later resumes processes.

5. CPU Scheduling Criteria

Main criteria & optimization goal:

I. **CPU Utilization:** keep CPU busy as much as possible. Goal: high %

II. **Throughput:** processes completed per time unit. Goal: maximize

III. **Turnaround Time:** finish time − arrival time. Goal: minimize average

IV. **Waiting Time:** total time in ready queue. Goal: minimize average

V. **Response Time:** time from submission to first response. Goal: minimize, important for interactive systems

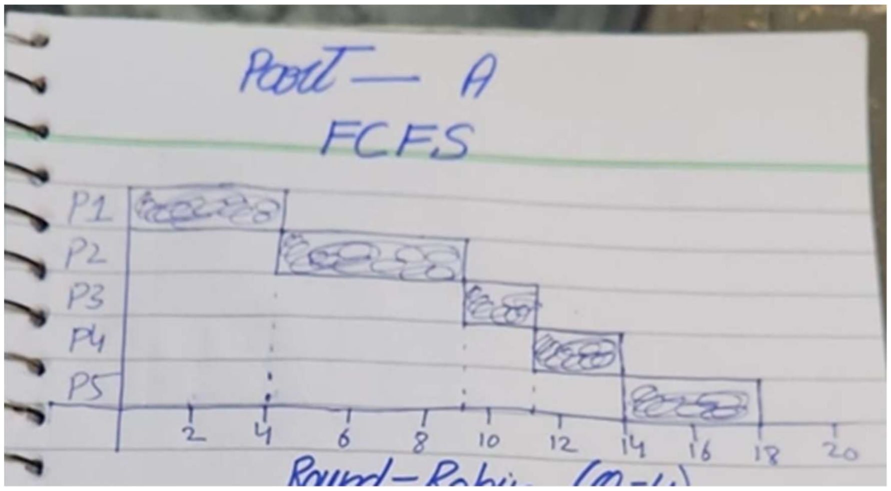
Section-D: CPU Scheduling Calculations

Part-A

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

FCFS

Gantt :



Completion, TurnAround, Waiting:

- **P1:**
    - Finish=4
    - Turn Around=4
    - Waiting=0

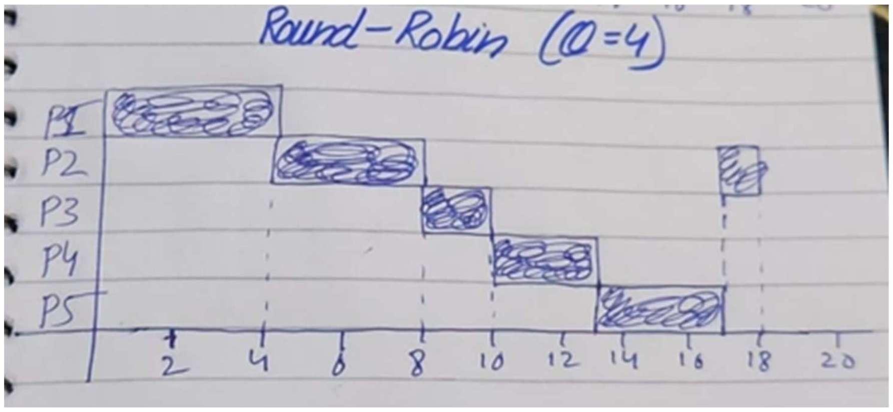
- **P2:**
  - Finish=9
  - Turn Around=7,
  - Waiting=2
- **P3:**
  - Finish=11
  - Turn Around=7
  - Waiting=5
- **P4:**
  - Finish=14
  - Turn Around=8
  - Waiting=5
- **P5:**
  - Finish=18
  - Turn Around=9
  - Waiting=5

Averages:

- Avg Wait Time = (0+2+5+5+5)/5 = 3.4
- Avg Turn Around = (4+7+7+8+9)/5 = 7.0
- Avg TR/TS = 2.16
- CPU Idle = 0

Round Robin (Q = 4)

Gantt:



Completion, Turn Around, Wait:

- **P1:**
    - Finish=4
    - Turn Around=4
    - Waiting=0
- **P2:**
    - Finish=18
    - Turn Around=16
    - Waiting=11
- **P3:**
    - Finish=10
    - Turn Around=6
    - Waiting=4
- **P4:**
    - Finish=13
    - Turn Around=7
    - Waiting=4

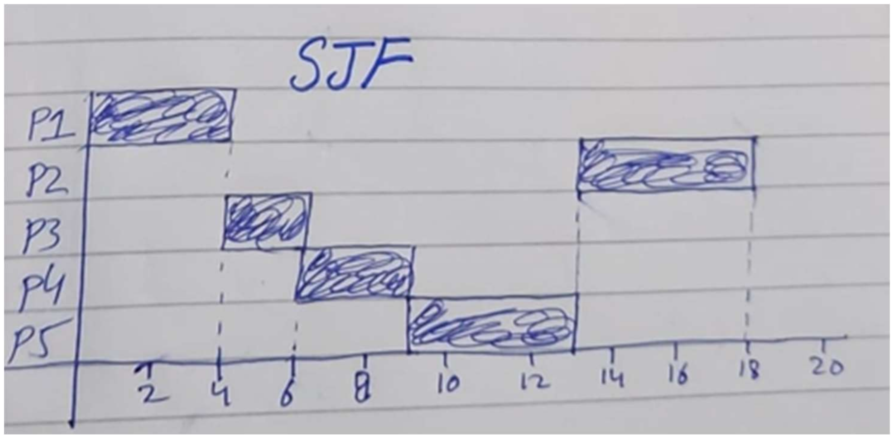
- **P5:**
  - Finish=17
  - Turn Around=8
  - Waiting=4

Averages:

- Avg Wait = (0+11+4+4+4)/5 = 4.6
- Avg Turn Around = (4+16+6+7+8)/5 = 8.2
- Avg TR/TS = 2.31
- CPU Idle = 0

SJF (non-preemptive)

Gantt:



Completion, Turn Around, Wait:

- **P1:**
  - Finish=4
  - Turn Around=4
  - Waiting=0
- **P2:**
  - Finish=18
  - Turn Around=16
  - Waiting=11
- **P3:**
  - Finish=6
  - Turn Around=2
  - Waiting=0
- **P4:**
  - Finish=9
  - Turn Around=3
  - Waiting=0

- **P5:**
    - Finish=13
    - Turn Around=4
    - Waiting=0

Averages:

- Avg Wait = (0+11+0+0+0)/5 = 2.2
- Avg Turn Around = (4+16+2+3+4)/5 = 5.8
- Avg TR/TS = 1.44
- CPU Idle = 0

SRTF (preemptive SJF)

For this dataset SRTF leads to same schedule as SJF because arrivals align. Metrics same as SJF.

Conclusion (Part-A):

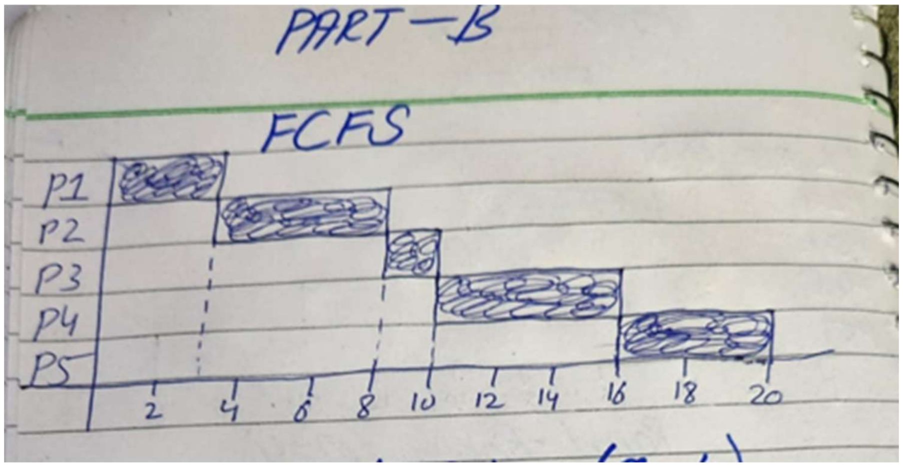
SJF/SRTF gives best average turnaround and smallest average waiting time for these arrivals. RR improves response but increases average turnaround vs SJF. FCFS is middle.

Part-B

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

FCFS

Gantt:



Completion, Turn Around, Wait:

- **P1:**
    - Finish=3
    - Turn Around=3
    - Waiting=0
- **P2:**
    - Finish=8
    - Turn Around=7
    - Waiting=2

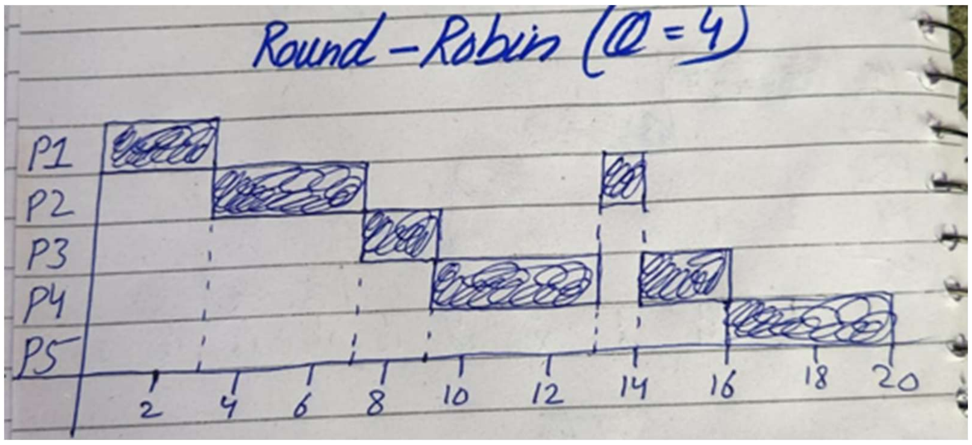
- **P3:**
    - Finish=10
    - Turn Around=7
    - Waiting=5
- **P4:**
    - Finish=16
    - Turn Around=7
    - Waiting=11
- **P5:**
    - Finish=20
    - Turn Around=10
    - Waiting=6

Averages:

- Avg Wait= 2.8
- Avg Turn Around = 6.8
- Avg TR/TS = 1.91
- CPU Idle = 0

Round Robin (Q = 4)

Gantt:



Completion, Turn Around, Wait:

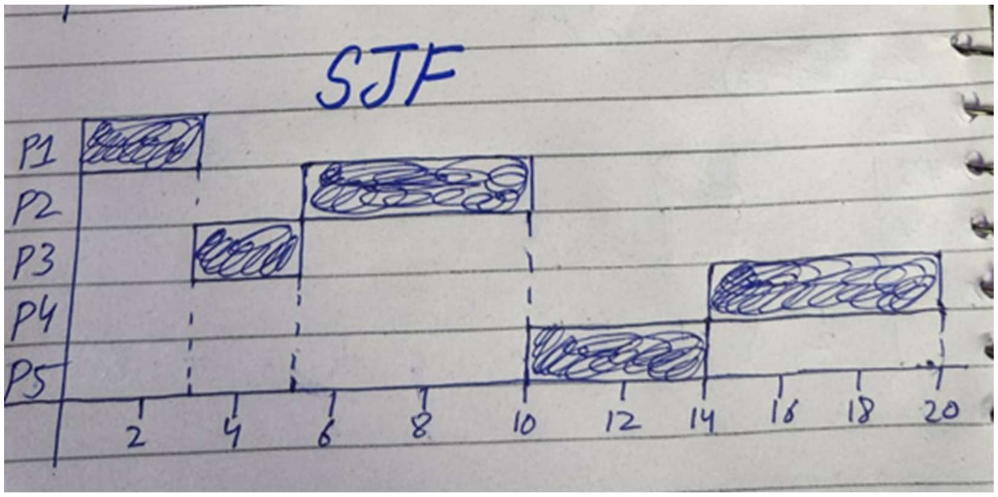
- **P1:**
    - Finish=3
    - Turn Around=3
    - Waiting=0
- **P2:**
    - Finish=14
    - Turn Around=13
    - Waiting=8
- **P3:**
    - Finish=9
    - Turn Around=6
    - Waiting=4
- **P4:**
    - Finish=16
    - Turn Around=7
    - Waiting=1
- **P5:**
    - Finish=20
    - Turn Around=10
    - Waiting=6

Averages:

- Avg Wait = 3.8
- Avg Turn Around = 7.8
- Avg TR/TS = 2.05
- CPU Idle = 0

SJF (non-preemptive)

Gantt:



Completion, Turn Around, Wait:

- **P1:**
  - Finish=3
  - Turn Around=3
  - Waiting=0
- **P2:**
  - Finish=10
  - Turn Around=9
  - Waiting=4
- **P3:**
  - Finish=5
  - Turn Around=2
  - Waiting=0
- **P4:**
  - Finish=20
  - Turn Around=11
  - Waiting=5

- **P5:**
    - Finish=14
    - Turn Around=4
    - Waiting=0

Averages:

- Avg Wait = 1.8
- Avg Turn Around = 5.8
- Avg TR/TS = 1.33
- CPU Idle = 0

SRTF

For these arrivals, SRTF behaves like SJF above same metrics.

Conclusion (Part-B):

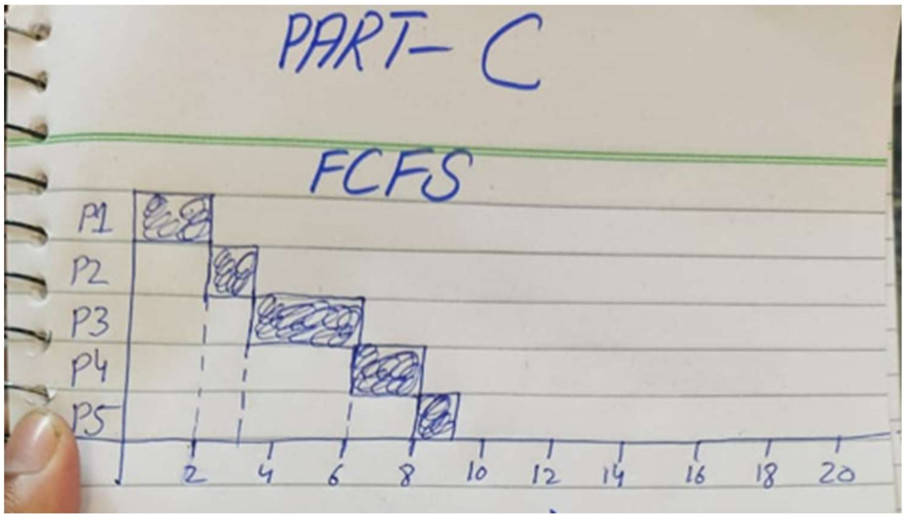
SJF gives best average waiting & turnaround; RR gives fairness but higher average turnaround than SJF.

Part-C (Your own data)

Process	**Arrival**	**Service**
P1	0	2
P2	1	1
P3	2	3
P4	3	2
P5	4	1

FCFS

Gantt:



Completion, Turn Around, Wait:

- **P1:**
  - Finish=2
  - Turn Around=2
  - Waiting=0
- **P2:**
  - Finish=3
  - Turn Around=3
  - Waiting=1
- **P3:**
  - Finish=6
  - Turn Around=4
  - Waiting=1
- **P4:**
  - Finish=8
  - Turn Around=5
  - Waiting=3

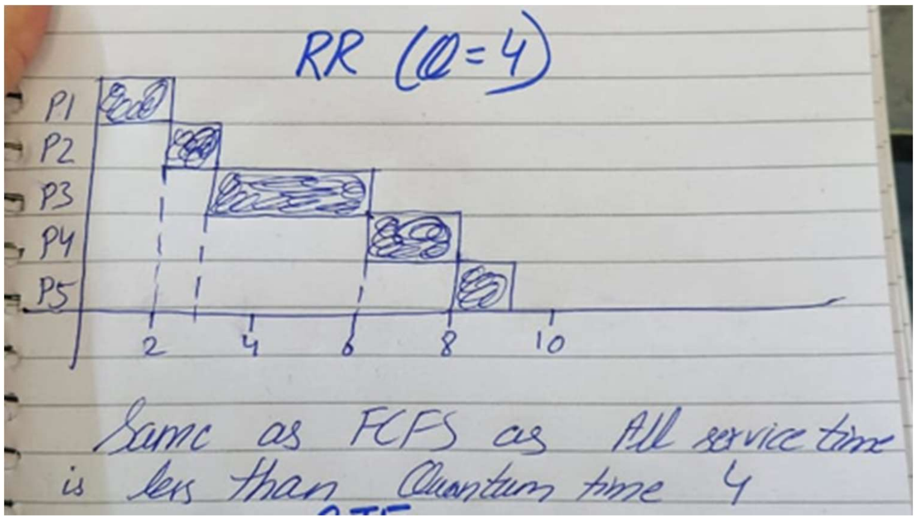
- **P5:**
  - Finish=9
  - Turn Around=5
  - Waiting=4

Averages:

- Avg Wait = 1.8
- Avg Turn Around = 3.6
- Avg TR/TS = 2.37
- CPU Idle = 0

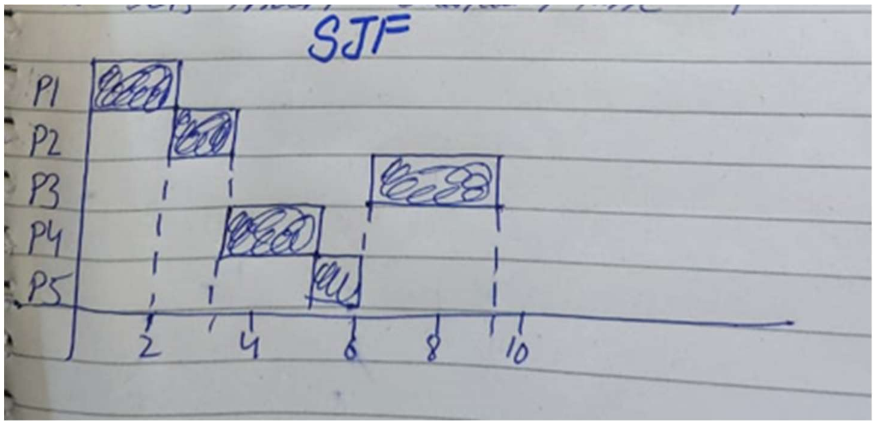
RR (Q=4)

All bursts ≤ 4 and arrival sequence continuous, so schedule same as FCFS here. Metrics same.



SJF (non-preemptive)

Gantt:



Completion, Turn Around, Wait:

- **P1:**
  - Finish=2
  - Turn Around=2
  - Waiting=0
- **P2:**
  - Finish=3
  - Turn Around=2
  - Waiting=1
- **P3:**
  - Finish=9
  - Turn Around=7
  - Waiting=4
- **P4:**
  - Finish=5
  - Turn Around=2
  - Waiting=0

- **P5:**
    - Finish=6
    - Turn Around=2
    - Waiting=1

Averages:

- Avg Wait = (0+1+4+0+1)/5 = 1.2
- Avg Turn Around = (2+2+7+2+2)/5 = 3.0

SRTF

Preemptions produce result like SJF for this small example; metrics close to SJF.

Conclusion (Part-C):

With these sample times SJF gives lower waiting times than FCFS.