



**National Textile University**  
**Department of Computer Science**

**Subject:**

Operating System

---

**Submitted to:**

Sir Nasir Mehmood

---

**Submitted by:**

Ghulam Mohyuddin

---

**Reg number:**

23-NTU-CS-1158

---

**Assignment number:**

1<sup>st</sup>

---

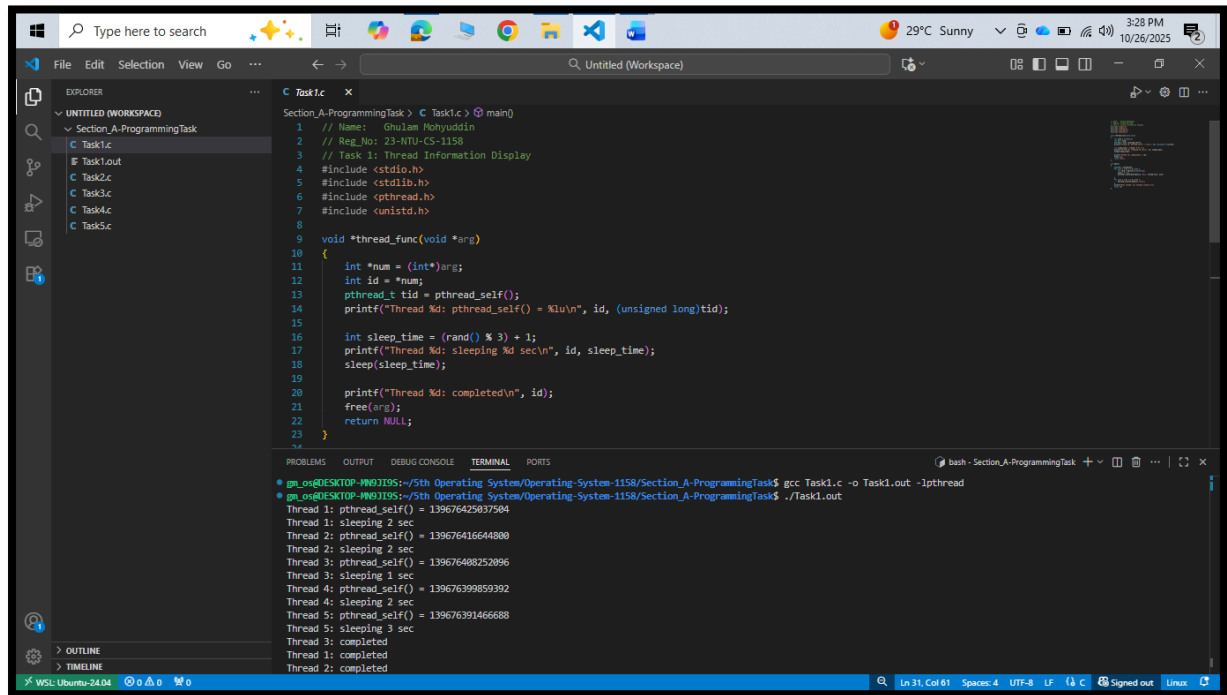
**Semester:**

5<sup>th</sup>

---

# Section-A: Programming Tasks

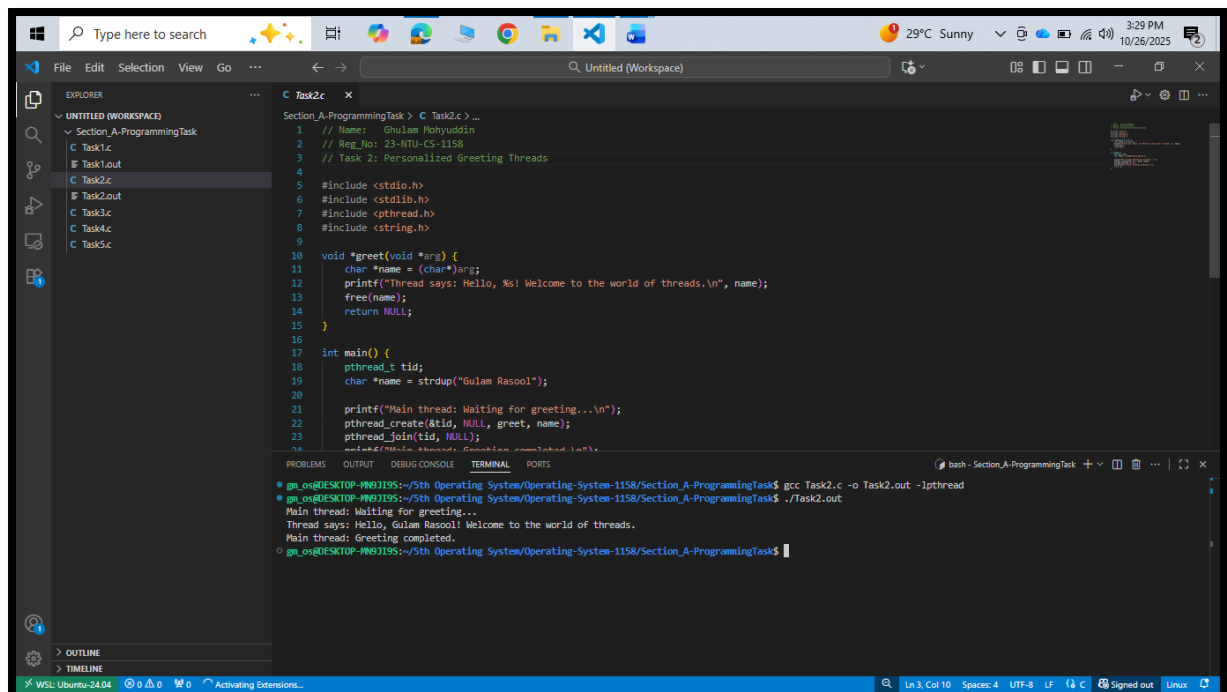
## Task 1: Thread Information Display



```
Section_A-ProgrammingTask > C Task1.c > main()
1 // Name: Ghulam Mohyuddin
2 // Reg.No: 23-NTU-CS-1158
3 // Task 1: Thread Information Display
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <pthread.h>
7 #include <unistd.h>
8
9 void *thread_func(void *arg)
10 {
11     int *num = (int*)arg;
12     int id = *num;
13     pthread_t tid = pthread_self();
14     printf("Thread %d: pthread_self() = %lu\n", id, (unsigned long)tid);
15
16     int sleep_time = (rand() % 3) + 1;
17     printf("Thread %d: sleeping %d sec\n", id, sleep_time);
18     sleep(sleep_time);
19
20     printf("Thread %d: completed\n", id);
21     free(arg);
22     return NULL;
23 }
24
25 int main()
26 {
27     pthread_t tid;
28     int num = 1;
29     while (num <= 5)
30     {
31         pthread_create(&tid, NULL, thread_func, (void *)num);
32         num++;
33     }
34     pthread_join(tid, NULL);
35 }
```

```
bash - Section_A-ProgrammingTask
gcc Task1.c -o Task1.out -lpthread
./Task1.out
Thread 1: pthread_self() = 139676425837584
Thread 1: sleeping 2 sec
Thread 2: pthread_self() = 139676416644880
Thread 2: sleeping 2 sec
Thread 3: pthread_self() = 139676488252896
Thread 3: sleeping 1 sec
Thread 4: pthread_self() = 139676399859392
Thread 4: sleeping 2 sec
Thread 5: pthread_self() = 139676391466688
Thread 5: sleeping 3 sec
Thread 3: completed
Thread 1: completed
Thread 2: completed
```

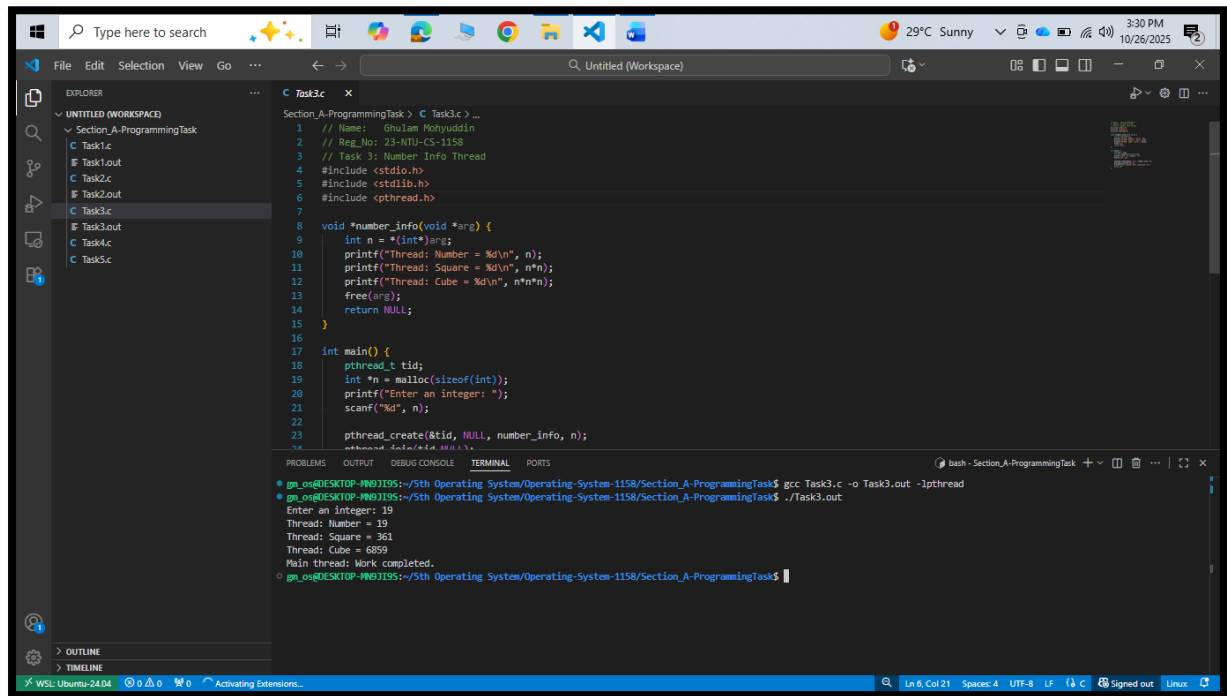
## Task 2: Personalized Greeting Thread



```
Section_A-ProgrammingTask > C Task2.c > ...
1 // Name: Ghulam Mohyuddin
2 // Reg.No: 23-NTU-CS-1158
3 // Task 2: Personalized Greeting Threads
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <pthread.h>
8 #include <string.h>
9
10 void *greet(void *arg) {
11     char *name = (char*)arg;
12     printf("Thread says: Hello, %s! Welcome to the world of threads.\n", name);
13     free(name);
14     return NULL;
15 }
16
17 int main() {
18     pthread_t tid;
19     char *name = strdup("Gulam Rasool");
20
21     printf("Main thread: Waiting for greeting...\n");
22     pthread_create(&tid, NULL, greet, name);
23     pthread_join(tid, NULL);
24     printf("Main thread: Greeting completed.\n");
25 }
```

```
bash - Section_A-ProgrammingTask
gcc Task2.c -o Task2.out -lpthread
./Task2.out
Main thread: Waiting for greeting...
Thread says: Hello, Ghulam Rasool! Welcome to the world of threads.
Main thread: Greeting completed.
```

## Task 3: Number Info Thread



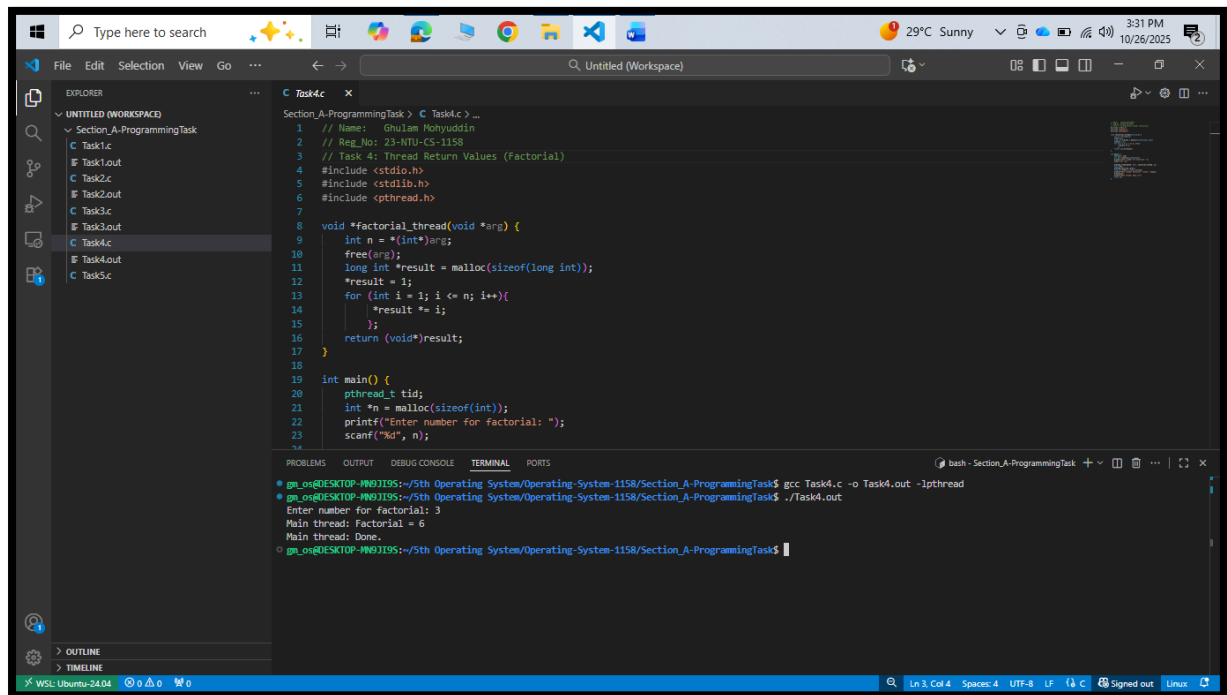
The screenshot shows a Visual Studio Code editor window with a workspace named 'Untitled (Workspace)'. The Explorer panel on the left shows a project structure with a folder 'Section\_A-ProgrammingTask' containing files 'Task1.c', 'Task1.out', 'Task2.c', 'Task2.out', 'Task3.c', 'Task3.out', 'Task4.c', and 'Task5.c'. The main editor displays the code for 'Task3.c'.

```
1 // Name: Ghulam Mohyuddin
2 // Reg.No: 23-NTU-CS-1158
3 // Task 3: Number Info Thread
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <pthread.h>
7
8 void *number_info(void *arg) {
9     int n = *(int*)arg;
10    printf("Thread: Number = %d\n", n);
11    printf("Thread: Square = %d\n", n*n);
12    printf("Thread: Cube = %d\n", n*n*n);
13    free(arg);
14    return NULL;
15 }
16
17 int main() {
18     pthread_t tid;
19     int *n = malloc(sizeof(int));
20     printf("Enter an integer: ");
21     scanf("%d", n);
22
23     pthread_create(&tid, NULL, number_info, n);
24 }
```

The TERMINAL panel at the bottom shows the execution of the program:

```
gn_osBESKTOP-11931195:~/5th Operating System/Operating-System-1158/Section_A-ProgrammingTask$ gcc Task3.c -o Task3.out -lpthread
gn_osBESKTOP-11931195:~/5th Operating System/Operating-System-1158/Section_A-ProgrammingTask$ ./Task3.out
Enter an integer: 19
Thread: Number = 19
Thread: Square = 361
Thread: Cube = 6859
Main thread: Work completed.
gn_osBESKTOP-11931195:~/5th Operating System/Operating-System-1158/Section_A-ProgrammingTask$
```

## Task 4: Thread Return Values



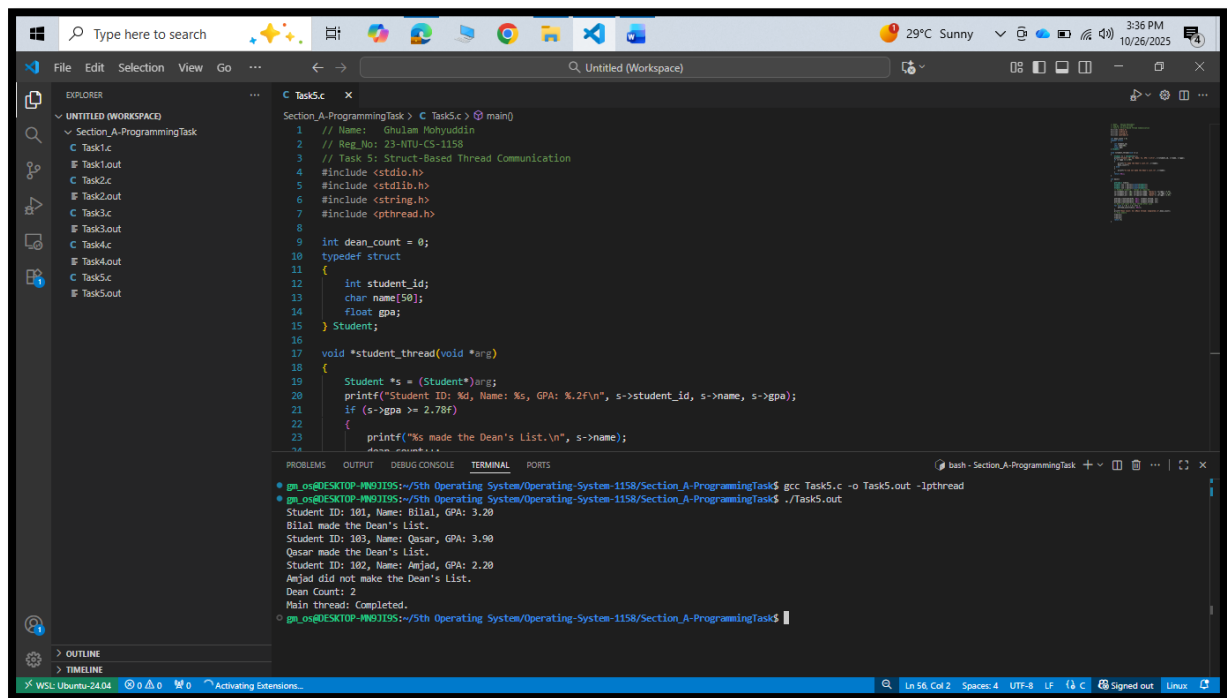
The screenshot shows a Visual Studio Code editor window with a workspace named 'Untitled (Workspace)'. The Explorer panel on the left shows a project structure with a folder 'Section\_A-ProgrammingTask' containing files 'Task1.c', 'Task1.out', 'Task2.c', 'Task2.out', 'Task3.c', 'Task3.out', 'Task4.c', and 'Task5.c'. The main editor displays the code for 'Task4.c'.

```
1 // Name: Ghulam Mohyuddin
2 // Reg.No: 23-NTU-CS-1158
3 // Task 4: Thread Return Values (Factorial)
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <pthread.h>
7
8 void *factorial_thread(void *arg) {
9     int n = *(int*)arg;
10    free(arg);
11    long int *result = malloc(sizeof(long int));
12    *result = 1;
13    for (int i = 1; i <= n; i++){
14        *result *= i;
15    }
16    return (void*)result;
17 }
18
19 int main() {
20     pthread_t tid;
21     int *n = malloc(sizeof(int));
22     printf("Enter number for factorial: ");
23     scanf("%d", n);
24 }
```

The TERMINAL panel at the bottom shows the execution of the program:

```
gn_osBESKTOP-11931195:~/5th Operating System/Operating-System-1158/Section_A-ProgrammingTask$ gcc Task4.c -o Task4.out -lpthread
gn_osBESKTOP-11931195:~/5th Operating System/Operating-System-1158/Section_A-ProgrammingTask$ ./Task4.out
Enter number for factorial: 3
Main thread: Factorial = 6
Main thread: Done.
gn_osBESKTOP-11931195:~/5th Operating System/Operating-System-1158/Section_A-ProgrammingTask$
```

## Task 5: Struct-Based Thread Communication



```
Section_A-ProgrammingTask > C Task5.c > main()
1 // Name: Ghulam Honyuddin
2 // Reg.No: 23-ITU-CS-1158
3 // Task 5: Struct-Based Thread Communication
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <string.h>
7 #include <pthread.h>
8
9 int dean_count = 0;
10 typedef struct
11 {
12     int student_id;
13     char name[50];
14     float gpa;
15 } Student;
16
17 void *student_thread(void *arg)
18 {
19     Student *s = (Student*)arg;
20     printf("Student ID: %d, Name: %s, GPA: %.2f\n", s->student_id, s->name, s->gpa);
21     if (s->gpa >= 2.78f)
22     {
23         printf("%s made the Dean's List.\n", s->name);
24     }
25 }
26
27 int main()
28 {
29     pthread_t t1, t2;
30     Student s1, s2;
31     s1.student_id = 101;
32     s1.name = "Bilal";
33     s1.gpa = 3.20;
34     s2.student_id = 102;
35     s2.name = "Amjad";
36     s2.gpa = 2.20;
37     pthread_create(&t1, NULL, student_thread, &s1);
38     pthread_create(&t2, NULL, student_thread, &s2);
39     pthread_join(t1, NULL);
40     pthread_join(t2, NULL);
41     printf("Dean Count: %d\n", dean_count);
42     printf("Main thread: Completed.\n");
43     return 0;
44 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
gn_os@DESKTOP-PM9J1195:~/5th Operating System/Operating-System-1158/Section_A-ProgrammingTask$ gcc Task5.c -o Task5.out -lpthread
gn_os@DESKTOP-PM9J1195:~/5th Operating System/Operating-System-1158/Section_A-ProgrammingTask$ ./Task5.out
Student ID: 101, Name: Bilal, GPA: 3.20
Bilal made the Dean's List.
Student ID: 102, Name: Amjad, GPA: 2.20
Amjad did not make the Dean's List.
Dean Count: 2
Main thread: Completed.
```

gn\_os@DESKTOP-PM9J1195:~/5th Operating System/Operating-System-1158/Section\_A-ProgrammingTask\$

## Section-B: Short Questions

**1. Define an Operating System in a single line.**

- ❖ An operating system is a type of system software that controls and coordinates hardware and software resources.
- ❖ It serves as a link between the user and the hardware to ensure smooth execution of programs.

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

- ❖ The CPU scheduler determines which process should run next on the CPU.
- ❖ It helps improve efficiency by applying scheduling algorithms to select the most suitable process.

**3. List any three states of a process.**

- ❖ A process passes through several states during its life cycle.
- ❖ The three key states are:
  - Ready.
  - Running.
  - Waiting (or Blocked).

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

- ❖ A PCB is a structure that keeps details about a process such as its ID, state, CPU registers, and memory usage.
- ❖ It allows the operating system to track and manage multiple processes.

**5. Differentiate between a process and a program.**

- ❖ A program is a set of instructions stored on disk.
- ❖ A process is the active execution of that program in memory.

**6. What do you understand by context switching?**

- ❖ Context switching occurs when the CPU switches from one process to another.
- ❖ It involves saving the current process state and restoring another so each can continue properly.

**7. Define CPU utilization and throughput.**

- ❖ CPU utilization measures the percentage of time the CPU is performing useful work.

- ❖ Throughput indicates how many processes finish execution within a specific time.

**8. What is the turnaround time of a process?**

- ❖ Turnaround time is the total duration from a process's arrival to its completion.
- ❖ It includes all waiting, execution, and I/O times.

**9. How is waiting time calculated in process scheduling?**

- ❖ Waiting time is the period a process remains in the ready queue before execution.
- ❖ It equals turnaround time minus burst time.

**10. Define response time in CPU scheduling.**

- ❖ Response time is the duration from when a process is submitted until it produces its first output.
- ❖ It is a key factor in interactive systems.

**11. What is preemptive scheduling?**

- ❖ Preemptive scheduling allows the CPU to interrupt a running process and assign the CPU to another.
- ❖ This ensures better responsiveness in multitasking environments.

**12. What is non-preemptive scheduling?**

- ❖ In non-preemptive scheduling, once a process starts on the CPU, it continues until it completes or blocks 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 all processes equal CPU time, preventing starvation.
- ❖ It is efficient for time-sharing systems and gives faster responses to smaller tasks.

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

- ❖ SJF can cause starvation for longer tasks as shorter ones are always prioritized.
- ❖ It also needs exact burst time knowledge, which is not always available.

**15. Define CPU idle time.**

- ❖ CPU idle time is when the processor is not running any task.
- ❖ It usually occurs when no process is ready to execute.

**16. State two common goals of CPU scheduling algorithms.**

- ❖ One goal is to maximize CPU utilization and throughput.
- ❖ Another is to minimize turnaround and waiting times.

**17. List two possible reasons for process termination.**

- ❖ A process may end normally after completing its task.
- ❖ It may also end abnormally because of errors or external termination signals.

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

- ❖ The exit() system call terminates a process and sends its status to the OS.
- ❖ The wait() call makes the parent wait until its child process finishes.

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

- ❖ In shared memory, multiple processes communicate by accessing a common memory space.
- ❖ In passing messages, processes exchange data using system calls like send and receive.

**20. Differentiate between a thread and a process.**

- ❖ A thread is a smaller unit within a process sharing memory and resources.
- ❖ A process operates independently with its own memory and system resources.

**21. Define multithreading.**

- ❖ Multithreading allows several threads within a single process to run simultaneously.
- ❖ It increases CPU efficiency and improves performance.

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

- ❖ A CPU-bound process spends most of its time performing computations.
- ❖ An I/O-bound process waits mainly for input and output operations.

**23. What are the main responsibilities of the dispatcher?**

- ❖ The dispatcher transfers CPU control to the selected process.
- ❖ It performs context switching, switches CPU modes, and starts execution at the correct location.

**24. Define starvation and aging in process scheduling.**

- ❖ Starvation occurs when a process never gets CPU time due to low priority.
- ❖ Aging gradually raises a process's priority to prevent starvation.

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

- ❖ A time quantum is a fixed duration for which a process is allowed to run.
- ❖ After it expires, the CPU switches to the next process in sequence.

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

- ❖ If too large, it behaves like FCFS and causes longer waiting times.
- ❖ If too small, it leads to frequent context switches and CPU overhead.

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

- ❖ The turnaround ratio compares turnaround time to service time.
- ❖ It shows how much total time was taken relative to the process's burst time.

**28. What is the purpose of a ready queue?**

- ❖ The ready queue stores processes waiting for CPU time.
- ❖ The scheduler chooses the next process from this queue for execution.

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

- ❖ A CPU burst happens when a process is executing instructions.
- ❖ 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 free from starvation because each process gets an equal share of CPU time.
- ❖ This ensures fair execution for all processes.



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

- ❖ The fork() system call creates a new child process.
- ❖ The exec() call replaces its memory with a new program.
- ❖ Finally, the parent uses wait() to get the child's termination status.

**32. Define zombie and orphan processes.**

- ❖ A zombie process has completed execution but remains in the process table.
- ❖ An orphan process is a child left running after its parent has terminated.

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

- ❖ Priority scheduling selects processes based on priority values.
- ❖ SJF chooses the process with the smallest burst time for execution.

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

- ❖ Context switch time is the period needed to save and load process information during switching.
- ❖ It is treated as overhead since no actual computation happens during this time.

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

- ❖ The long-term scheduler manages which jobs enter the system.
- ❖ The medium-term scheduler swaps processes in and out of memory.
- ❖ The short-term scheduler selects which ready process runs next on the CPU.

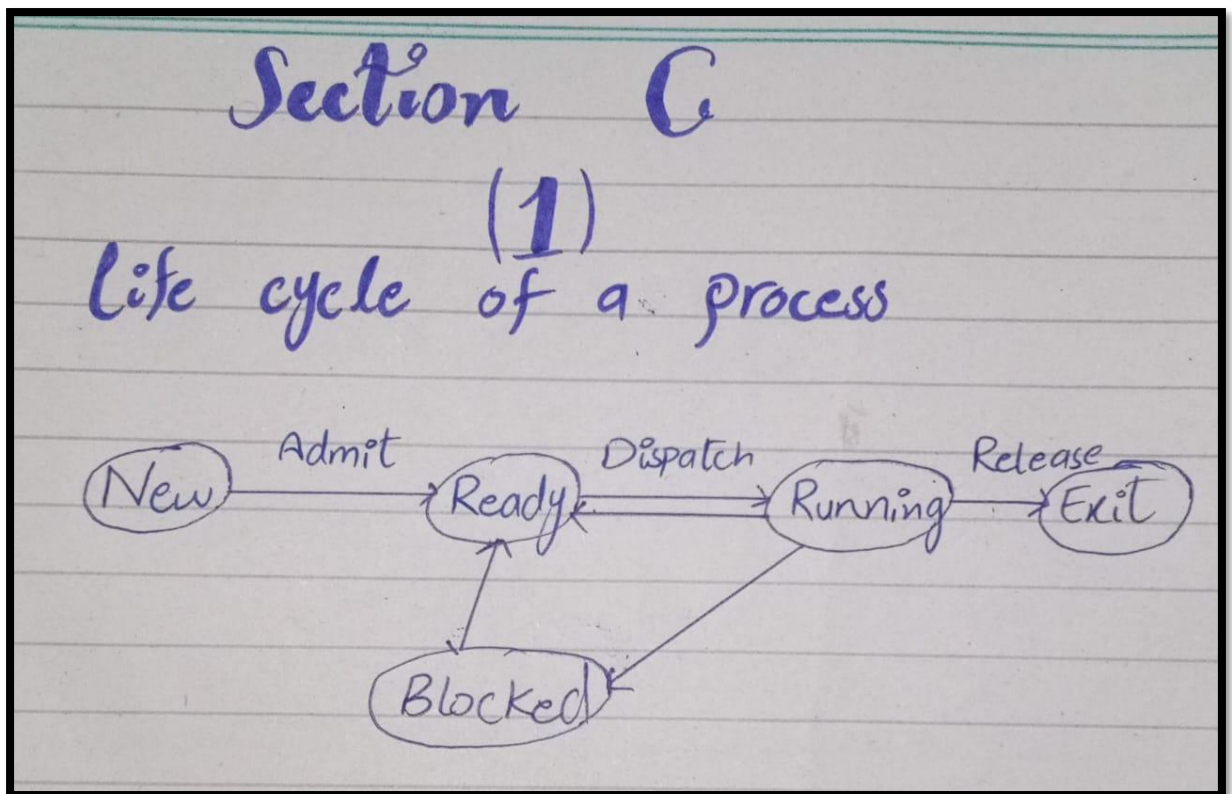
**36. Differentiate between User Mode and Kernel Mode in an OS.**

- ❖ User mode is where regular applications run with limited privileges.
- ❖ Kernel mode allows the operating system full control over hardware and critical resources.

## 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.



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

#### Context switch overhead:

- ❖ A context switch occurs when the CPU switches from executing one process to another.
- ❖ The time taken to save the current process's state and load the next process's state is called **context switch overhead**.
- ❖ This period is considered non-productive because no actual computation happens — only process management.

## What must be saved & restored:

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

### 3. Components of a Process Control Block (PCB)

- I. **Process state** shows the current status of the process such as running, ready, or waiting.
- II. **Program counter** holds the address of the next instruction the process will execute.
- III. **CPU registers:** store the values of all registers specific to that process.
- IV. **CPU scheduling information** includes process priority and pointers to scheduling queues.
- V. **Memory-management information** records the memory space assigned to the process.
- VI. **Accounting information:** track CPU usage, total runtime, and assigned time limits.
- VII. **I/O status information** contains information about allocated I/O devices and the list of open files.

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

Aspect	Long-Term Scheduler (Job Scheduler)	Short-Term Scheduler (CPU Scheduler)	Medium-Term Scheduler (Swapper)
Function	Chooses processes from the job pool stored in secondary memory and loads them into main memory for execution.	Selects one ready process from memory to run on the CPU.	Temporarily removes processes from memory and brings them back later to manage system load.

<b>Speed</b>	Operates at the slowest rate among all schedulers.	Works at the fastest rate and is invoked frequently.	Functions at a moderate speed compared to the other two.
<b>Use in Systems</b>	Often absent in time-sharing systems or rarely used.	Always present and crucial for time-sharing operations.	Commonly used in time-sharing systems for swapping processes in and out.
<b>Key Operation</b>	Brings new processes into main memory for execution.	Assigns the CPU to one of the ready processes.	Suspends and resumes processes based on system resource availability.

## 5. CPU Scheduling Criteria

### Main criteria & optimization goal:

- I. **CPU Utilization:** Aim to keep the processor active most of the time. Goal to achieve the highest possible percentage.
- II. **Throughput:** Measure of how many processes are completed per time unit. Goal to maximize completion rate.
- III. **Turnaround Time:** Time difference between process completion and arrival. Goal to reduce average turnaround time.
- IV. **Waiting Time:** Total time a process stays in the ready queue before execution. Goal to minimize average waiting time.
- V. **Response Time:** Duration from process submission to its first response. Goal to minimize, especially vital for interactive systems.

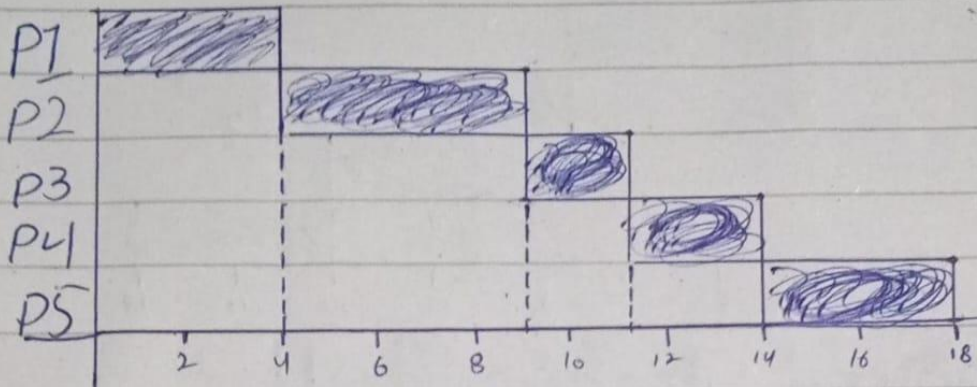
## Section-D: CPU Scheduling Calculations

### Part-A

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

## FCFS

### Grant:



### Completion, turnaround, waiting:

P1: Finish=4, Turnaround=4, waiting=0

P2: Finish=9, Turnaround=7, waiting=2

P3: Finish=11, Turnaround=1, waiting=5

P4: Finish=14, Turnaround=8, waiting=5

P5: Finish=18, Turnaround=9, waiting=5

### Averages:

Avg wait time =  $(0+2+5+5+5)/5 = 3.4$

Avg Turnaround time =  $(4+7+7+8+9)/5 = 7.0$

Avg TR/Ts = 2.16

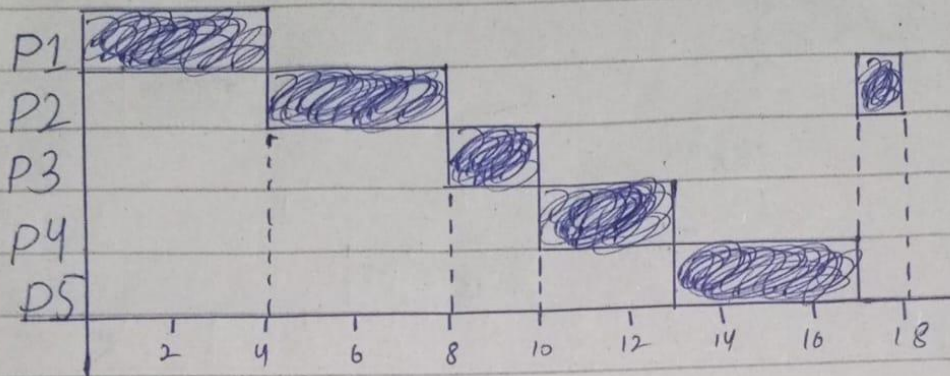
CPU idle = 0



## Round Robin (Q = 4)

### Round Robin (Q = 4)

#### Gantt:



#### Completion, turn around, waiting:

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:

Arg Turn around time =  $(4+16+6+7+8)/5 = 8.2$

Arg wait time =  $(0+11+4+4+4)/5 = 4.6$

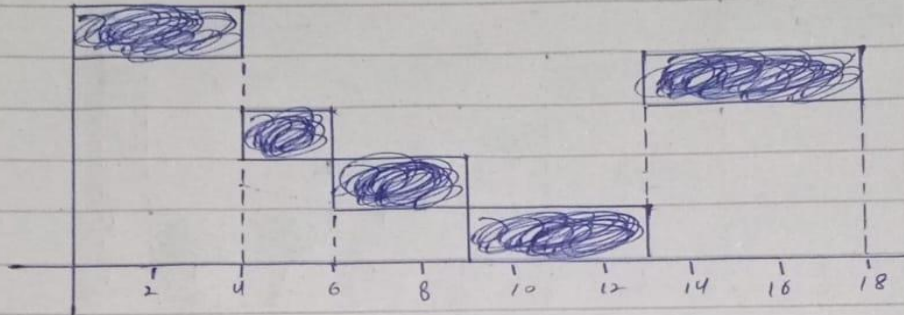
Arg TR/TS = 2.31

CPU idle = 0

## SJF (non-preemptive)

SJF

Gantt:



Completion, turn around, waiting:

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=8, Turn around=3, Waiting=0

P5: Finish=13, Turn around=4, Waiting=0

Averages:

Avg turn around Time =  $(4+16+2+3+4)/5 = 5.8$

Avg wait Time =  $(0+11+0+0+0)/5 = 2.2$

Avg TRTS = 1.44

CPU idle = 0

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



## Conclusion:

SJF/STRF give best average turn around and smallest average waiting time for these arrivals. RR improves response but increase average turn around vs SJF. FCFS is middle.

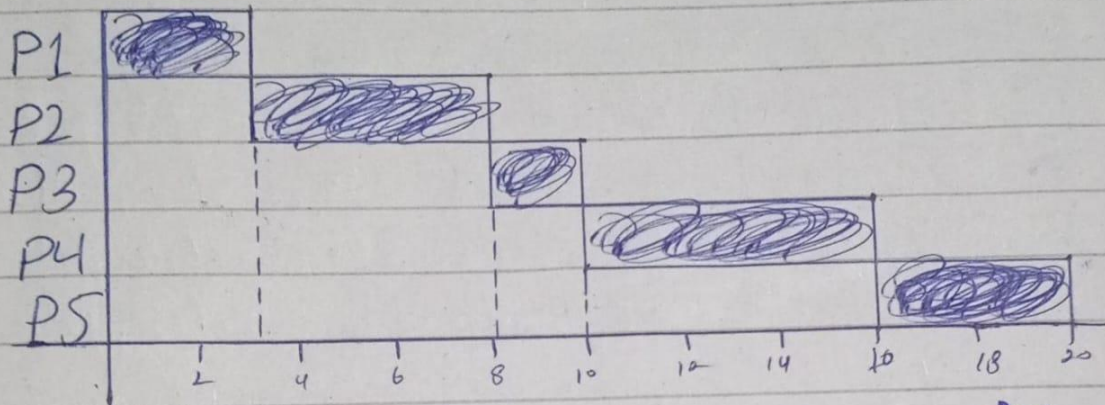
### Part-B

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

## FCFS

Grantt<sub>3</sub>

FCFS



Completion, turn around, waiting:

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:

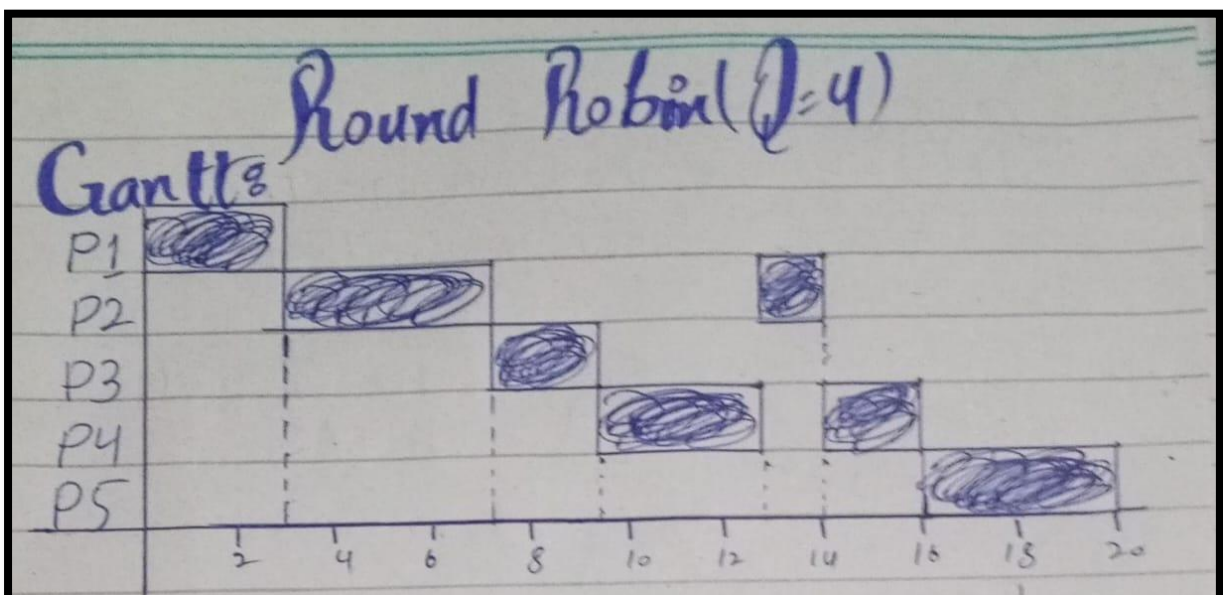
Avg wait =  $(0+2+5+11+6)/5 = 2.8$

Avg turn around =  $(3+7+7+7+10)/5 = 6.8$

Avg TR/Ts = 1.91

CPU idle = 0

## Round Robin (Q = 4)



**Completion, turn around, waiting:**

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=20, Waiting=6

**Averages:**

Avg Turn around =  $(3+13+6+7+20)/5 = 7.8$

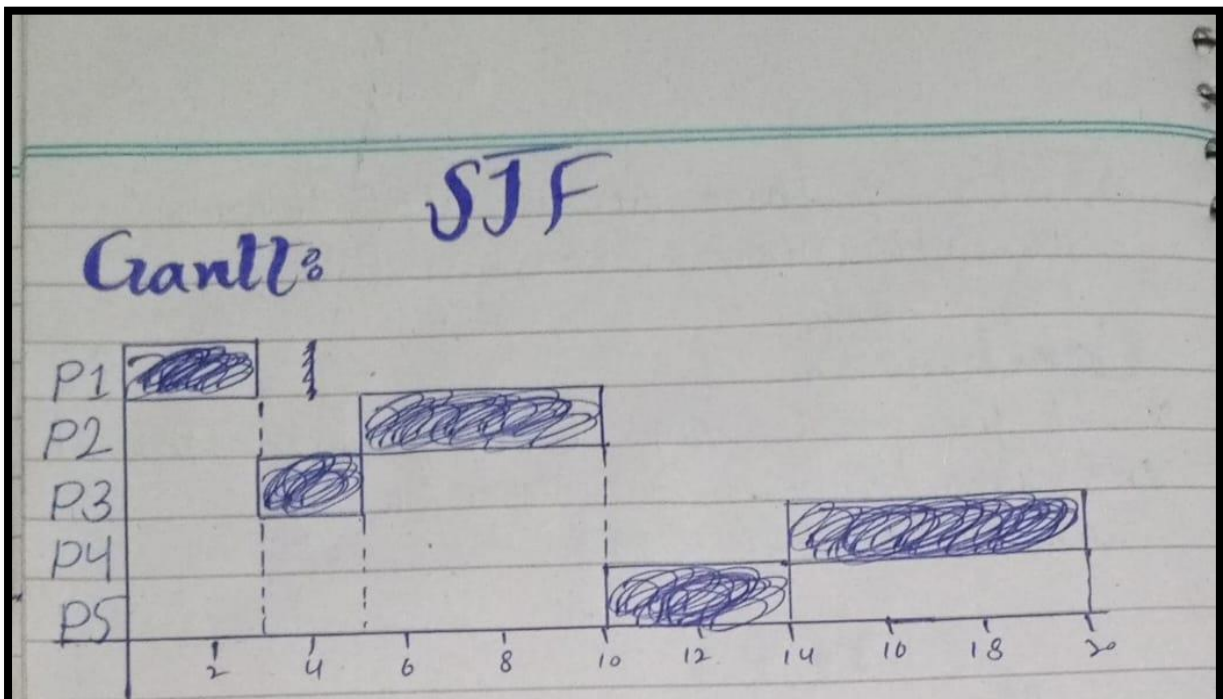
Avg Wait =  $(0+8+4+1+6)/5 = 3.8$

Avg TR/TS = 2.05

CPU idle = 0



## SJF (non-preemptive)



Completion, turn around, waiting:

P1	: Finish=3	Turn around=3	Waiting=0
P2	: Finish=10	Turn around=9	Waiting=4
P3	: Finish=4	Turn around=2	Waiting=0
P4	: Finish=14	Turn around=11	Waiting=5
P5	: Finish=20	Turn around=6	Waiting=0

Averages:

$$\begin{aligned}\text{Avg Turn around} &= (3+9+2+11+6) / 5 = 5.8 \\ \text{Avg wait} &= (0+4+0+5+0) / 5 = 1.8 \\ \text{Avg TR/Ts} &= 1.33 \\ \text{CPU idle} &= 0\end{aligned}$$

**STRF:** For these arrivals, STRF behaves like SJF above same metrics.

### Conclusion:

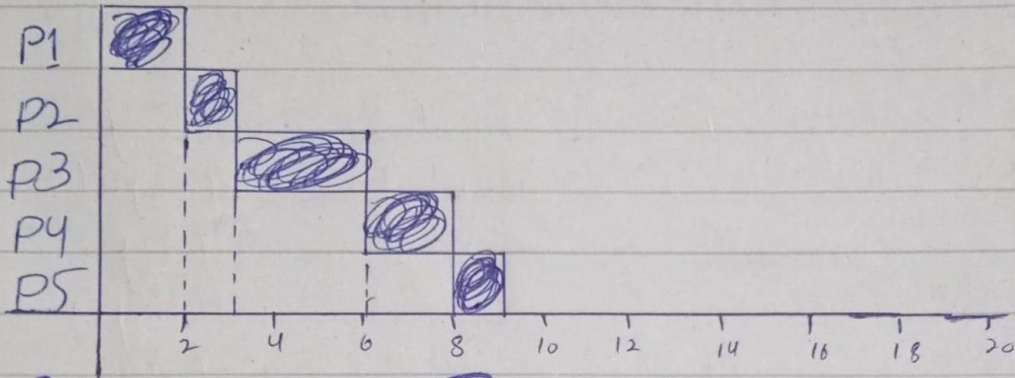
- ❖ SJF give best average waiting & turnaround.
- ❖ RR gives fairness but higher average turnaround then SJF.

#### Part-C (Your own data)

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

## FCFS

FCFS  
Gantt:



Completion, Turn around, Waiting:

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

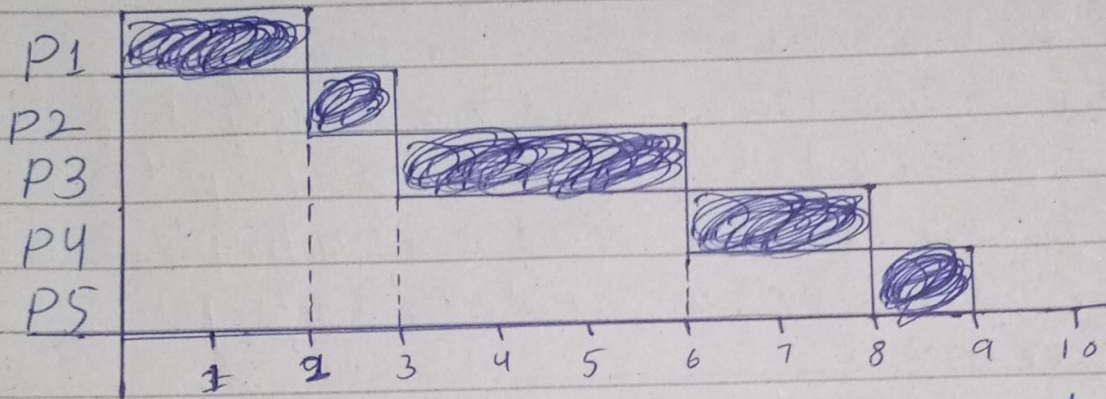
Averages:

Avg wait =  $(0+1+1+3+4)/5 = 1.8$   
Avg Turn around =  $(2+3+4+5+5)/5 = 3.6$   
Avg TR/Ts = 2.37  
CPU idle = 0



RR (Q=4)

## Round Robin (Q=4) Gantt:

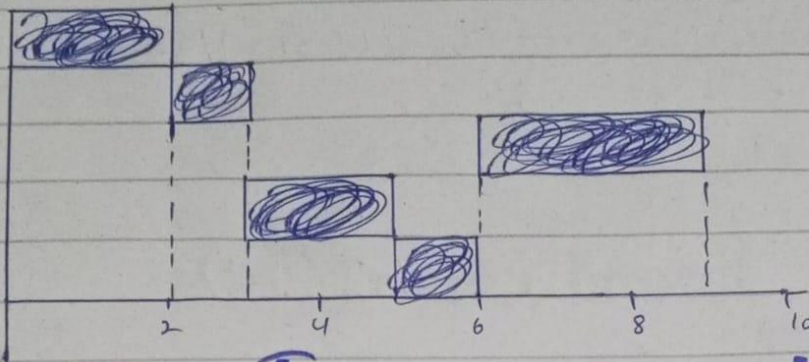


All burst less than or equal 4 and arrival sequence continuous, so schedule same as FCFS here.

## SJF (non-preemptive)

SJF

Grant:



Completion, Turn around, wait:

P1: Finish=2, Turn around=2, waiting=0

P2: Finish=3, Turn around=2, waiting=1

P3: Finish=6, Turn around=7, waiting=4

P4: Finish=7, Turn around=2, waiting=0

P5: Finish=11, Turn around=2, waiting=1

Averages:

$$\text{Avg Turn around} = (2+2+7+2+2)/5 = 3.0$$

$$\text{Avg wait} = (0+1+4+0+1)/5 = 1.2$$

STRF: Preemptions produce results like SJF for this small example; metrics close to SJF.

Conclusion: with these sample times SJF gives lower waiting times than FCFS.