National Textile University, Faisalabad

Department of Computer Science

Name	Dawood Saif
Class	SE-5th (A)
Reg. No.	23-NTU-CS-1145
Course	Operating system
Submitted To	Sir Nasir Mehmood
Submission Date	26/10/2025
Assignment	1st Assignment before Mid

Operating System: 1st Assignment

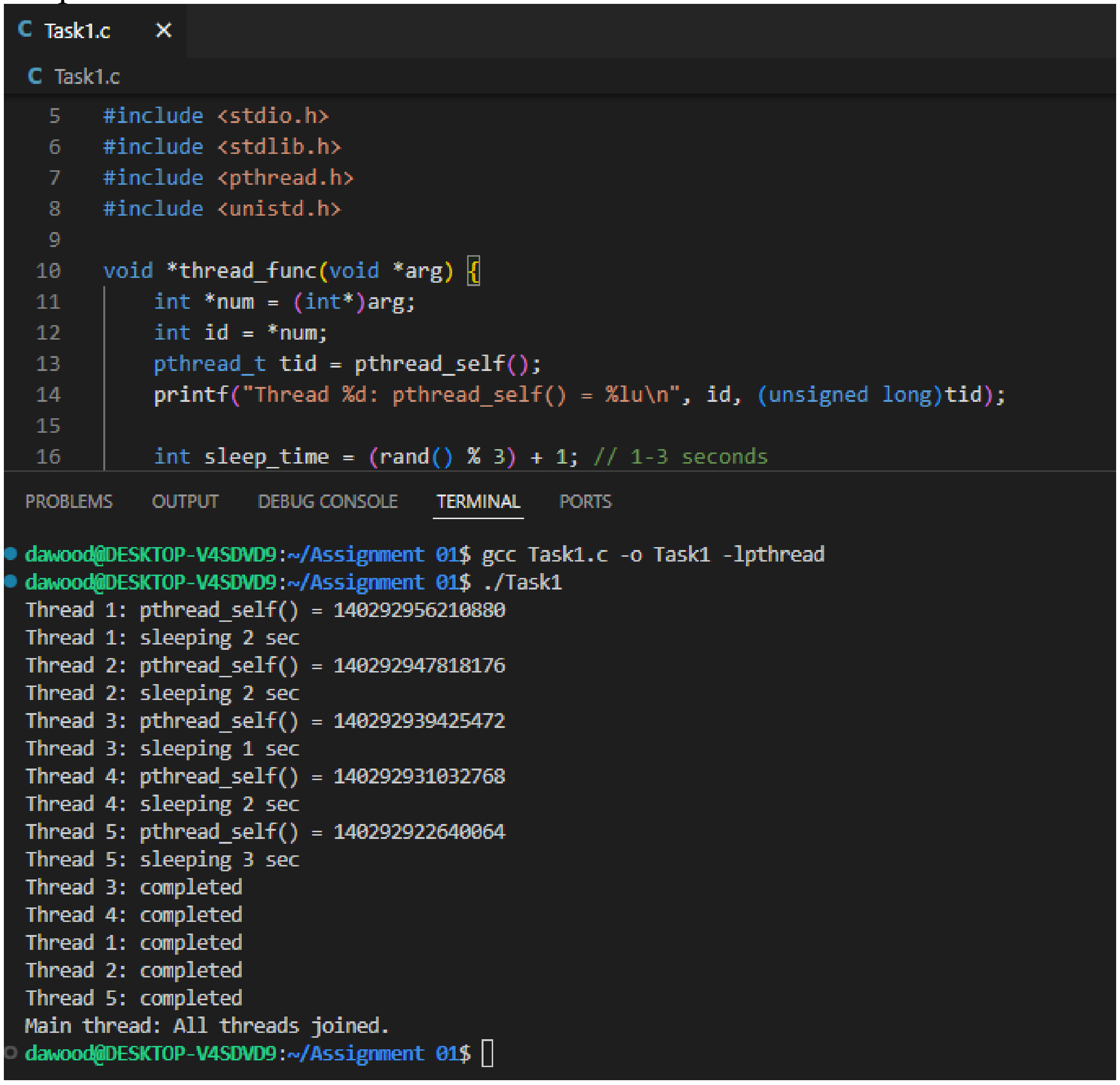
Section -A (Programing Tasks)

Task 1 – Thread Information Display
Write a program that creates 5 threads. Each thread should:
Print its thread ID using `pthread_self()`. Display its thread number (1st, 2nd, etc.). Sleep for a random time between 1–3 seconds. Print a completion message before exiting.

Code:

```c
// Task 1 – Thread Information Display
// Dawood Saif
// Reg. No. 23-NTU-CS-1145

#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);
    return NULL;
}

int main() {
    pthread_t threads[5];
    for (int i = 0; i < 5; i++) {
        int *arg = malloc(sizeof(int));
        *arg = i + 1;
        pthread_create(&threads[i], NULL, thread_func, arg);

    }

    for (int i = 0; i < 5; i++) {
        pthread_join(threads[i], NULL);
    }
    printf("Main thread: All threads joined.\n");
    return 0;
}
```

Output Screenshot:



```c
#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
```

```
dawood@DESKTOP-V4SDVD9:~/Assignment 01$ gcc Task1.c -o Task1 -lpthread
dawood@DESKTOP-V4SDVD9:~/Assignment 01$ ./Task1
Thread 1: pthread_self() = 140292956210880
Thread 1: sleeping 2 sec
Thread 2: pthread_self() = 140292947818176
Thread 2: sleeping 2 sec
Thread 3: pthread_self() = 140292939425472
Thread 3: sleeping 1 sec
Thread 4: pthread_self() = 140292931032768
Thread 4: sleeping 2 sec
Thread 5: pthread_self() = 140292922640064
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.
dawood@DESKTOP-V4SDVD9:~/Assignment 01$ 
```

Task 2 – Personalized Greeting Thread

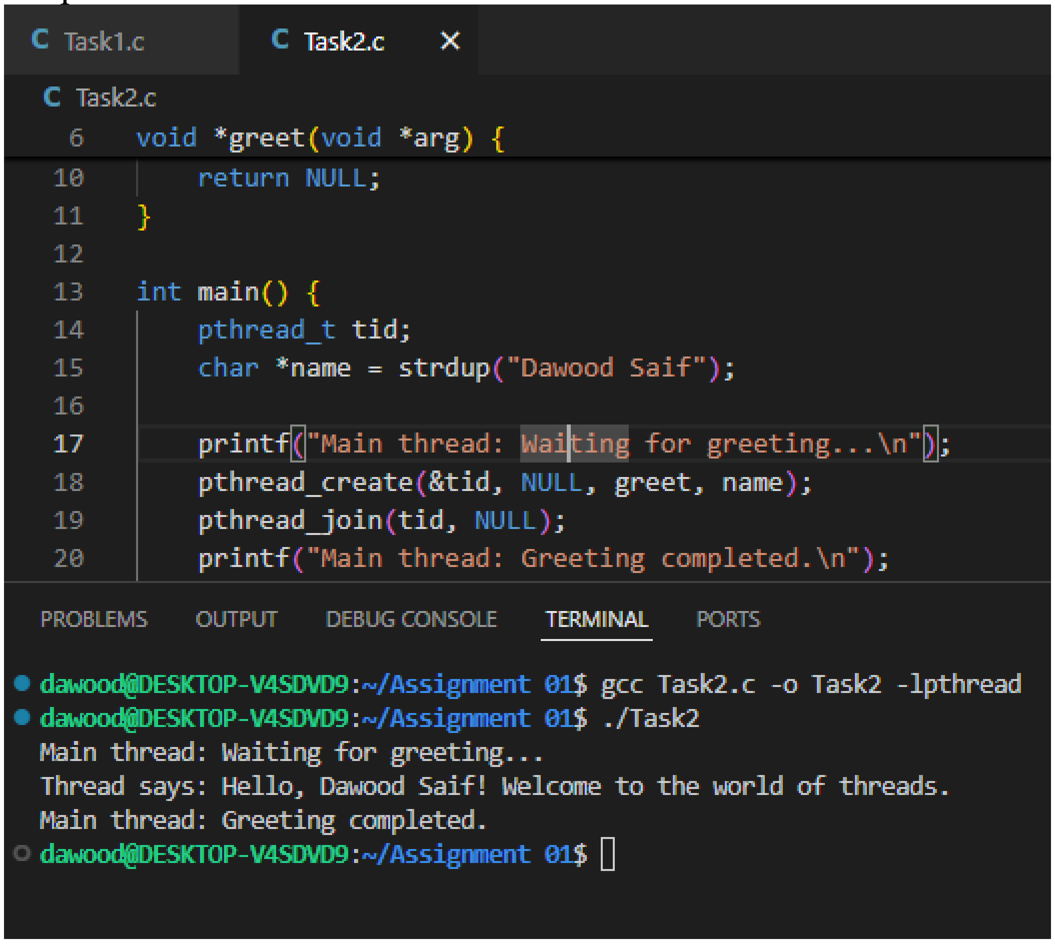
Write a C program that:

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

Code:

```c
// Task 2 – Personalized Greeting Thread
// Dawood Saif
// Reg. No. 23-NTU-CS-1145

#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("Dawood Saif");

    printf("Main thread: Waiting for greeting...\n");
    pthread_create(&tid, NULL, greet, name);
    pthread_join(tid, NULL);
    printf("Main thread: Greeting completed.\n");
    return 0;
}
```

Output Screenshot:



Task1.c Task2.c

```c
void *greet(void *arg) {
    return NULL;
}

int main() {
    pthread_t tid;
    char *name = strdup("Dawood Saif");

    printf("Main thread: Waiting for greeting...\n");
    pthread_create(&tid, NULL, greet, name);
    pthread_join(tid, NULL);
    printf("Main thread: Greeting completed.\n");
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
dawood@DESKTOP-V4SDVD9:~/Assignment 01$ gcc Task2.c -o Task2 -lpthread
dawood@DESKTOP-V4SDVD9:~/Assignment 01$ ./Task2
Main thread: Waiting for greeting...
Thread says: Hello, Dawood Saif! Welcome to the world of threads.
Main thread: Greeting completed.
dawood@DESKTOP-V4SDVD9:~/Assignment 01$ 
```

Task 3 – Number Info Thread

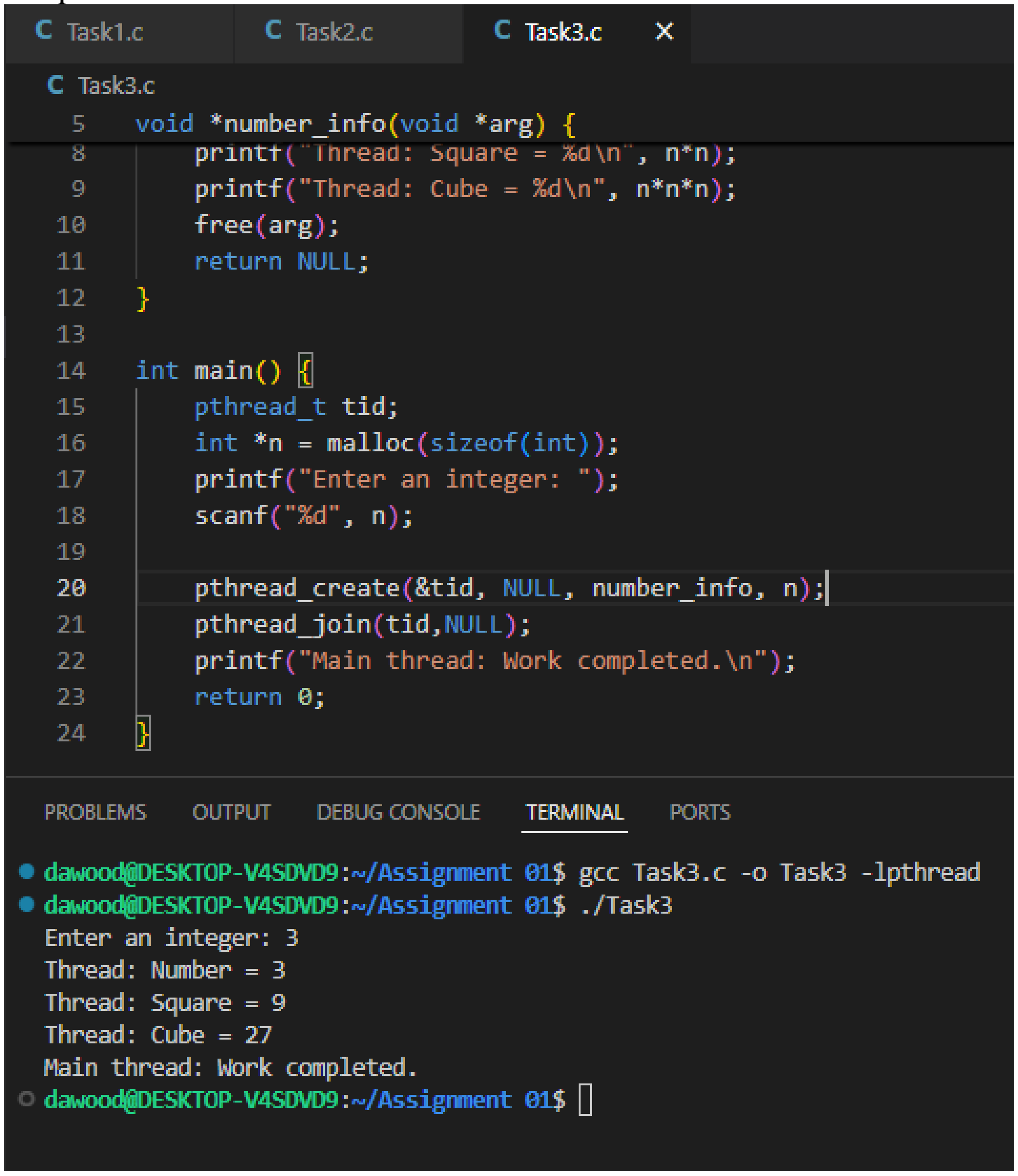
Write a program that:

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
//  Task 3 - Number Information Display
//  Dawood Saif
//  Reg. No. 23-NTU-CS-1145

#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: ");
    scanf("%d", n);

    pthread_create(&tid, NULL, number_info, n);
    pthread_join(tid,NULL);
    printf("Main thread: Work completed.\n");
    return 0;
}
```

Output Screenshot:



```c
void *number_info(void *arg) {
    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: ");
    scanf("%d", n);

    pthread_create(&tid, NULL, number_info, n);
    pthread_join(tid,NULL);
    printf("Main thread: Work completed.\n");
    return 0;
}
```

```
dawood@DESKTOP-V4SDVD9:~/Assignment 01$ gcc Task3.c -o Task3 -lpthread
dawood@DESKTOP-V4SDVD9:~/Assignment 01$ ./Task3
Enter an integer: 3
Thread: Number = 3
Thread: Square = 9
Thread: Cube = 27
Main thread: Work completed.
dawood@DESKTOP-V4SDVD9:~/Assignment 01$
```

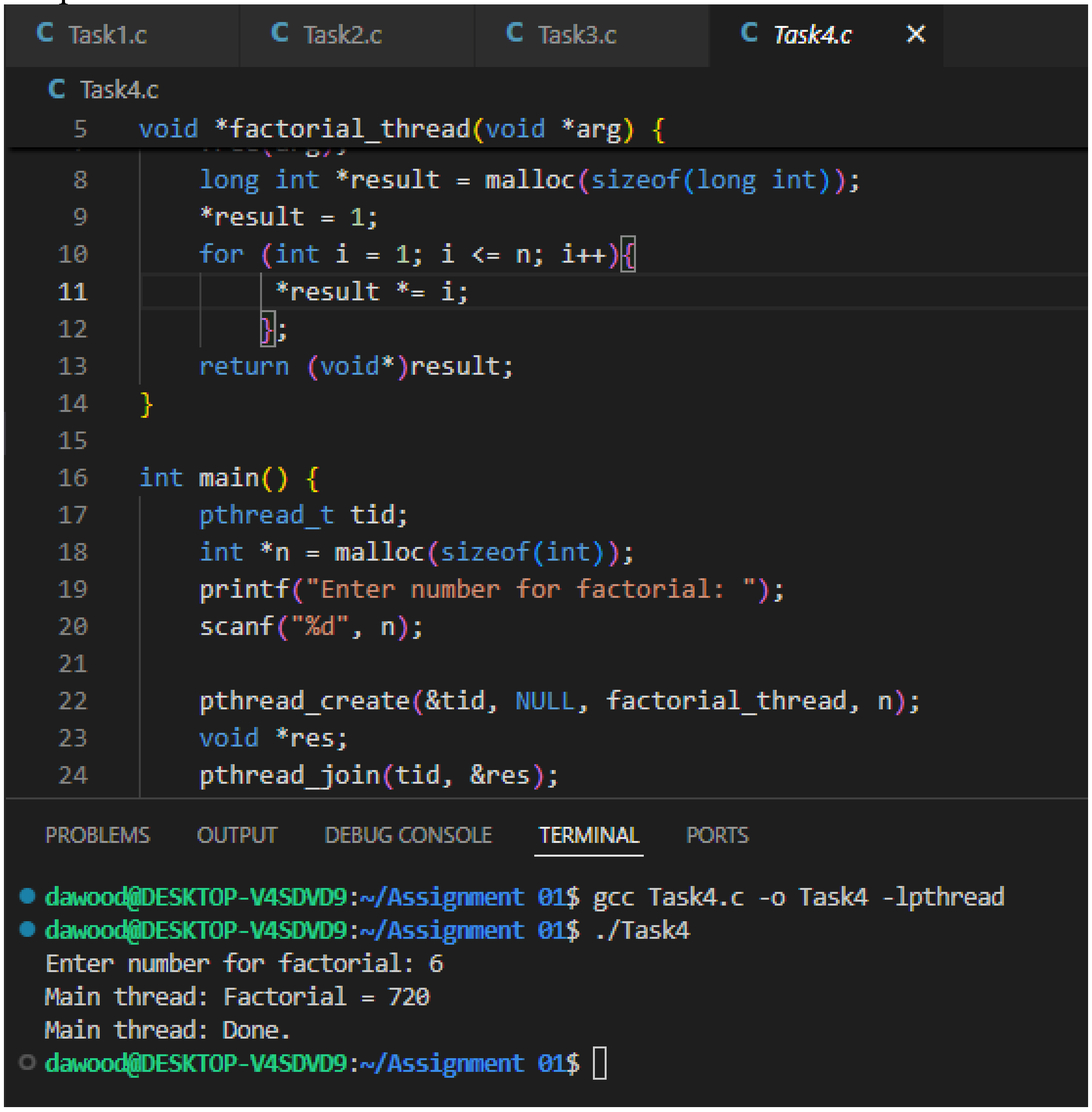
Task 4 – Thread Return Values

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:

```c
//Task 4 – Thread Return Values
//Dawood Saif
//Reg. No. 23-NTU-CS-1145

#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;
    int *n = malloc(sizeof(int));
    printf("Enter number for factorial: ");
    scanf("%d", n);

    pthread_create(&tid, NULL, factorial_thread, n);
    void *res;
    pthread_join(tid, &res);
    long int *fact = (long int*)res;
    printf("Main thread: Factorial = %ld\n", *fact);
    free(fact);
    printf("Main thread: Done.\n");
    return 0;
}
```

Output Screenshot:



```
void *factorial_thread(void *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;
    int *n = malloc(sizeof(int));
    printf("Enter number for factorial: ");
    scanf("%d", n);

    pthread_create(&tid, NULL, factorial_thread, n);
    void *res;
    pthread_join(tid, &res);
```

```
dawood@DESKTOP-V4SDVD9:~/Assignment 01$ gcc Task4.c -o Task4 -lpthread
dawood@DESKTOP-V4SDVD9:~/Assignment 01$ ./Task4
Enter number for factorial: 6
Main thread: Factorial = 720
Main thread: Done.
dawood@DESKTOP-V4SDVD9:~/Assignment 01$
```

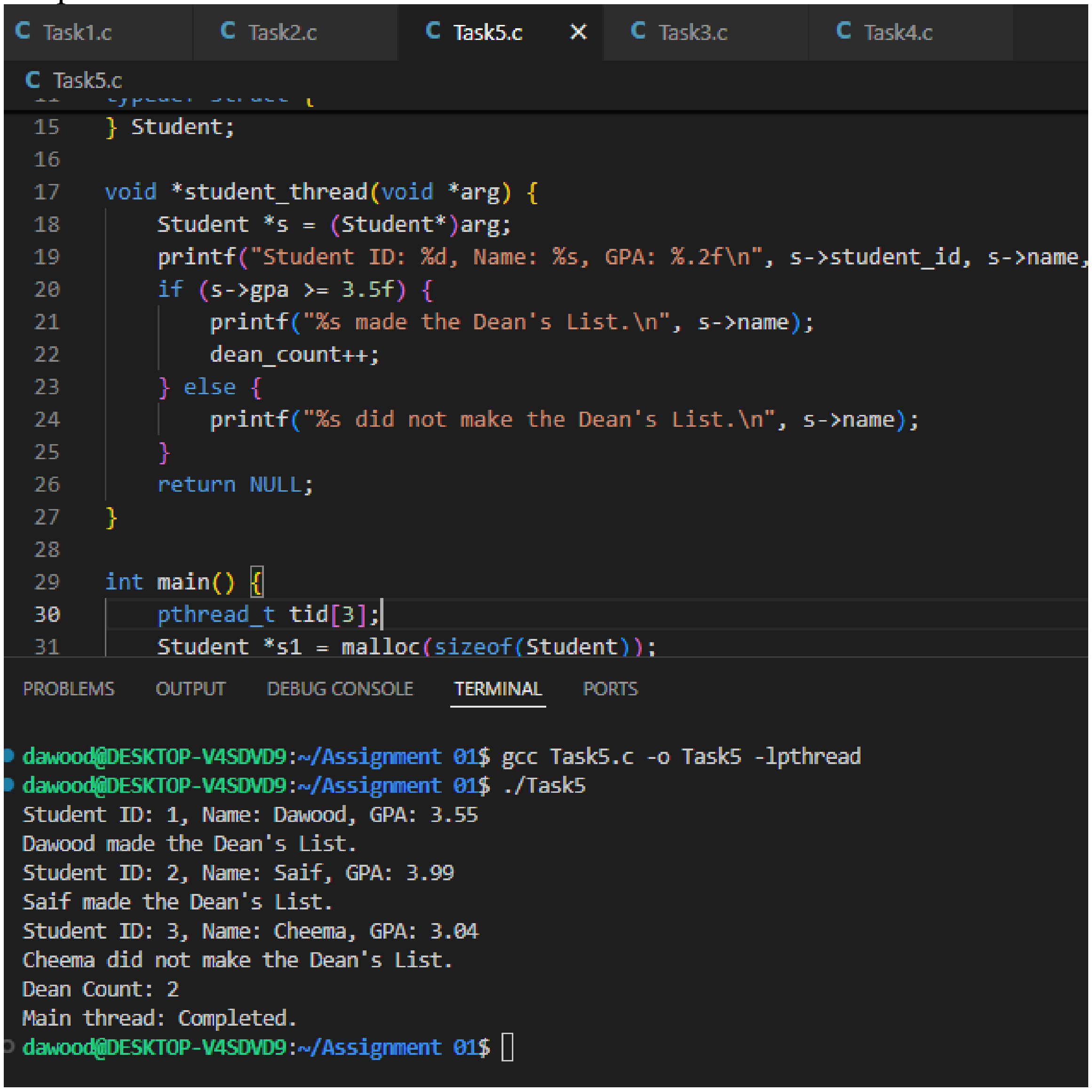
Task 5 – Struct-Based Thread Communication
Create a program that simulates a simple student database system.
Define a struct: typedef struct {int, student-id ; char name[50]; float gpa; } Student;`
Create 3 threads, each receiving a different Student struct. Each thread prints student info and checks Dean's List eligibility (GPA ≥ 3.5). The main thread counts how many students made the Dean's List.

Code:

```
// Task 5 – Struct-Based Thread Communication
// Dawood Saif
// Reg. No. 23-NTU-CS-1145

#include <stdio.h>
#include <stdlib.h>
```

```c
#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);
        dean_count++;
    } else {
        printf("%s did not make the Dean's List.\n", s->name);
    }
    return NULL;
}

int main() {
    pthread_t tid[3];
    Student *s1 = malloc(sizeof(Student));
    Student *s2 = malloc(sizeof(Student));
    Student *s3 = malloc(sizeof(Student));

    // Example students - change as required
    s1->student_id = 1; strcpy(s1->name, "Dawood"); s1->gpa = 3.55f;
    s2->student_id = 2; strcpy(s2->name, "Saif");  s2->gpa = 3.99f;
    s3->student_id = 3; strcpy(s3->name, "Cheema"); s3->gpa = 3.04f;

    pthread_create(&tid[0], NULL, student_thread, s1);
    pthread_create(&tid[1], NULL, student_thread, s2);
    pthread_create(&tid[2], NULL, student_thread, s3);


    // Wait and count in main thread after join
    for (int i = 0; i < 3; i++) {
        pthread_join(tid[i], NULL);
    }

    printf("Dean Count: %d \nMain thread: Completed.\n",dean_count);

    // Free memory
    free(s1);
    free(s2);
    free(s3);
    return 0;
}
```

Output Screenshot:



```c
} Student;

void *student_thread(void *arg) {
    Student *s = (Student*)arg;
    printf("Student ID: %d, Name: %s, GPA: %.2f\n", s->student_id, s->name,
    if (s->gpa >= 3.5f) {
        printf("%s made the Dean's List.\n", s->name);
        dean_count++;
    } else {
        printf("%s did not make the Dean's List.\n", s->name);
    }
    return NULL;
}

int main() {
    pthread_t tid[3];
    Student *s1 = malloc(sizeof(Student));
```

```
dawood@DESKTOP-V4SDVD9:~/Assignment 01$ gcc Task5.c -o Task5 -lpthread
dawood@DESKTOP-V4SDVD9:~/Assignment 01$ ./Task5
Student ID: 1, Name: Dawood, GPA: 3.55
Dawood made the Dean's List.
Student ID: 2, Name: Saif, GPA: 3.99
Saif made the Dean's List.
Student ID: 3, Name: Cheema, GPA: 3.04
Cheema did not make the Dean's List.
Dean Count: 2
Main thread: Completed.
dawood@DESKTOP-V4SDVD9:~/Assignment 01$
```

Section-B (Theory Questions)

Answer all questions briefly and clearly.

1) Define an Operating System in a single line.

Answer:
An Operating System (OS) is a system software that manages computer hardware, software resources, and provides services for computer programs. It acts as a bridge between the user and the computer hardware.

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

Answer:
The main function of the CPU scheduler is to select which process from the ready queue should be executed next by the CPU. It helps in maintaining efficiency and ensures fair CPU utilization.

3) List any three states of a process.

Answer:
The three common states of a process are:

- Ready (waiting to be assigned to CPU),
- Running (currently being executed),
- Waiting/Blocked (waiting for some I/O or event to complete).

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

Answer:
A Process Control Block (PCB) is a data structure maintained by the operating system to store information about a process such as its process ID, state, CPU registers, program counter, and

memory allocation details.

5) Differentiate between a process and a program.

Process	**Program**
A process is a program in execution.	A program is a passive set of instructions stored on disk.
It has a specific state and memory allocation.	It does not have a state until executed.
Processes are dynamic and temporary.	Programs are static and permanent.

6) What do you understand by context switching?

Answer:
Context switching is the process of saving the state of a running process and loading the state of another process. It allows multiple processes to share a single CPU effectively and supports multitasking.

7) Define CPU utilization and throughput.

Answer:
CPU utilization is the percentage of time the CPU is actively executing processes instead of being idle. Throughput refers to the number of processes completed by the system in a given period.

8) What is the turnaround time of a process?

Answer:
Turnaround time is the total time taken for a process to complete from the moment it enters the system until it finishes execution. It includes waiting time, execution time, and I/O time.

9) How is waiting time calculated in process scheduling?

Answer:
Waiting time is the total time a process spends in the ready queue waiting for CPU allocation. It

is calculated as: Waiting Time = Turnaround Time - Burst Time.

10) Define response time in CPU scheduling.

Answer:
Response time is the time interval between the submission of a process and the first response produced by the system. It is important in interactive systems for user experience.

11) What is preemptive scheduling?

Answer:
Preemptive scheduling allows the CPU to interrupt a currently running process to assign the CPU to another process with higher priority or when the time slice expires. It improves responsiveness.

12) What is non-preemptive scheduling?

Answer:
non-preemptive scheduling means that once a process starts execution, it runs until it finishes or voluntarily releases the CPU. It is simpler but can cause poor responsiveness.

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

Answer:
1. Each process gets an equal share of CPU time, ensuring fairness.
2. It provides good response time for interactive systems by using time slices.

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

Answer:
The main drawback of SJF is that long processes may starve if shorter ones keep arriving. It also requires prior knowledge of the burst time, which is not always possible.

15) Define CPU idle time.

Answer:
CPU idle time refers to the period when the CPU is not executing any process. This usually happens when there are no processes in the ready queue or when it waits for I/O operations.

16) State two common goals of CPU scheduling algorithms.

Answer:
1. To maximize CPU utilization and system throughput.
2. To minimize waiting time, response time, and turnaround time for all processes.

17) List two possible reasons for process termination.

Answer:
1. The process completes its execution successfully.
2. The process is terminated by the system or user due to an error or lack of resources.

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

Answer:
The wait() system call makes a parent process wait until its child process finishes execution, while exit() allows a process to terminate and release all its allocated resources.

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

Shared Memory	**Message Passing**
Processes share a common memory space for communication.	Processes exchange data by sending and receiving messages.
Faster as it doesn't involve kernel intervention.	Slower due to kernel involvement.
Suitable for large data exchange.	Better for small data or distributed systems.

20) Differentiate between a thread and a process.

Thread	**Process**
A thread is a lightweight unit of a process.	A process is a heavy, independent unit of execution.

Threads share memory and resources within the same process.	Each process has its own memory space and resources.
Switching between threads is faster.	Switching between processes is slower.

21) Define multithreading.

Answer:
Multithreading allows multiple threads within a single process to run concurrently. It helps improve performance and CPU utilization by executing multiple tasks at once.

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

CPU-bound Process	**I/O-bound Process**
Spends most of its time performing computations.	Spends most of its time waiting for I/O operations.
Requires more CPU time.	Requires more I/O time.
Example: Mathematical calculations.	Example: Reading files from disk.

23) What are the main responsibilities of the dispatcher?

Answer:
The dispatcher is responsible for giving control of the CPU to the process selected by the scheduler. It performs context switching, changes process states, and starts execution at the correct instruction.

24) Define starvation and aging in process scheduling.

Starvation	**Aging**
When a process waits indefinitely for CPU allocation.	Gradual increase in process priority to prevent starvation.
Caused by low-priority processes never getting CPU time.	Ensures fairness by boosting waiting process priority.

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

Answer:
A time quantum or time slice is a fixed amount of CPU time given to each process in Round Robin scheduling before the CPU switches to the next process.

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

Answer:
If the time quantum is too large, response time becomes poor. If it's too small, there are too many context switches, reducing CPU efficiency.

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

Answer:
The turnaround ratio is the ratio of turnaround time to service time (TR/TS). It helps to measure how efficiently the process was executed compared to its required CPU time.

28) What is the purpose of a ready queue?

Answer:
The ready queue stores all processes that are ready to execute but are waiting for CPU allocation. The CPU scheduler picks processes from this queue for execution.

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

CPU Burst	**I/O Burst**
Time during which a process uses CPU for computation.	Time during which a process waits for I/O operations.
Occurs when CPU executes instructions.	Occurs when data is read or written to a device.
Example: Arithmetic calculation.	Example: Disk read or write operation.

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

Answer:
Round Robin scheduling is starvation-free because every process gets a fixed time slice to run.

No process is left waiting indefinitely.

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

Answer:
1. The parent process uses fork() to create a child.
2. The child gets a copy of the parent's memory.
3. exec() replaces the child's code with a new program.
4. The process starts execution independently.

32) Define zombie and orphan processes.

Zombie Process	**Orphan Process**
A process that has completed execution but still has an entry in the process table.	A child process whose parent has finished execution.
Exists until the parent calls wait().	Adopted by the init process.

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

Priority Scheduling	**Shortest Job First (SJF)**
Selects process based on highest priority value.	Selects process based on shortest CPU burst time.
Can cause starvation of low-priority processes.	Can cause starvation of long processes.
Used when importance matters.	Used when time efficiency matters.

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

Answer:
Context switch time is the time taken to save the state of one process and load another's. It is considered overhead because no useful work is done during this time.

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

Answer:

1. **Long-term scheduler:** Decides which processes enter the ready queue.
2. **Short-term scheduler:** Chooses which process runs next on the CPU.
3. **Medium-term scheduler:** Suspends and resumes processes to optimize performance.

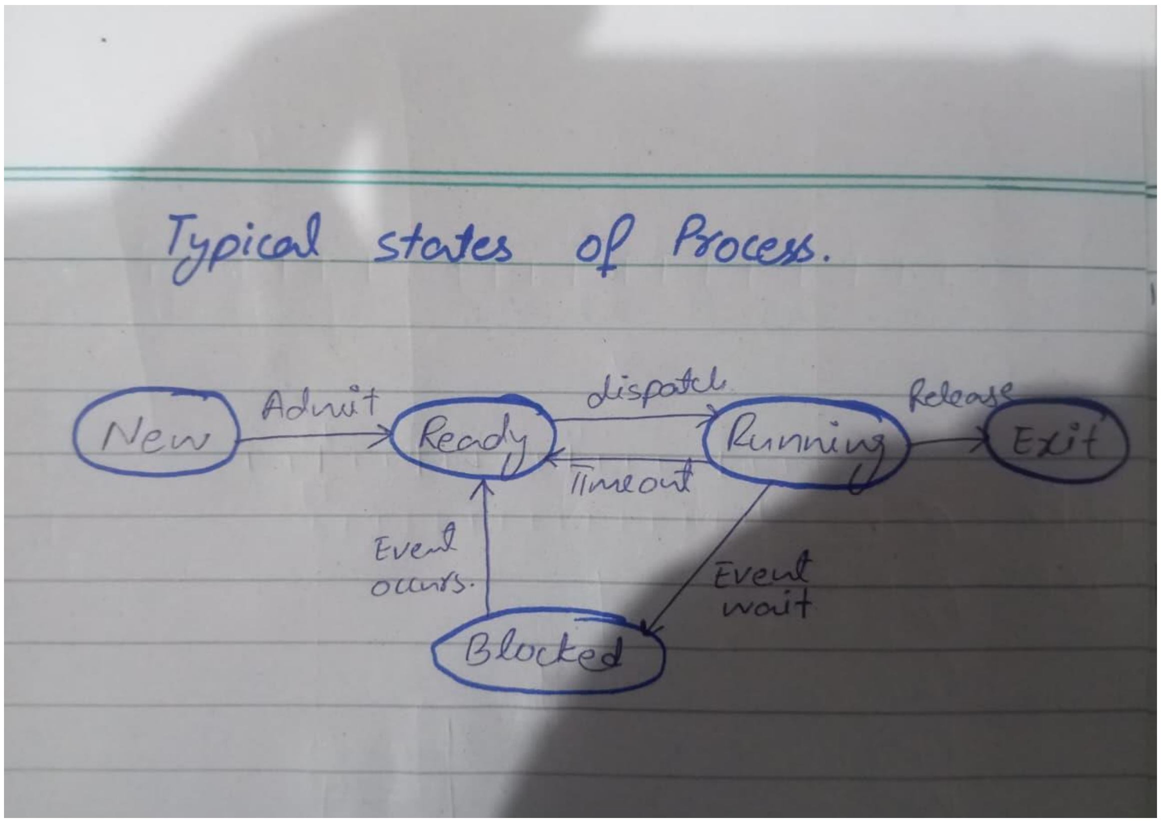
36) Differentiate between User Mode and Kernel Mode in an Operating System.

Answer:

User Mode	Kernel Mode
Used to run user applications with limited access to system resources.	Used by the operating system to execute system-level tasks.
Cannot directly access hardware or memory.	Has full access to system hardware and memory.
Safer but less powerful mode.	Powerful but riskier if errors occur.

Section-C (Logical and 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.



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

Answer:

Context Switch Overhead:

This is the extra time and work the CPU spends during the switch — time when no useful work (no process execution) is happening.
This overhead includes saving and restoring registers, memory maps, and CPU state.

So, context switch overhead = lost CPU time due to switching.

What Information Must Be Saved and Restored?

When switching, the CPU must save the current process's state in its PCB (Process Control Block) and load the next process’s state from its PCB.

Saved/restored information includes:

- CPU registers (Program Counter, Stack Pointer, etc.)
- Process state (Running, Ready, Waiting)

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

Answer:

Component	Description
Process ID (PID)	Unique number for identifying the process.
Process State	Current status (New, Ready, Running, Waiting, Terminated).
Program Counter (PC)	Holds the address of the next instruction to execute.
CPU Registers	Stores temporary data for process execution.
Memory Management Info	Includes base/limit registers, page tables, or segment tables.
Accounting Info	Used for scheduling — CPU time used, process priority, etc.
I/O Status Info	List of open files, devices assigned, pending I/O requests.

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

Long-Term Scheduler	Medium-Term Scheduler	Short-Term Scheduler
Also called Job Scheduler.	Also called Swapper.	Also called CPU Scheduler.
Decides which new processes are admitted into the ready queue from the job pool.	Temporarily removes or resumes processes from main memory to reduce load.	Decides which ready process gets the CPU next.
Controls the degree of multiprogramming (number of processes in memory).	Helps improve memory utilization and system performance.	Handles process switching very frequently (in milliseconds).
Works less frequently (when new jobs arrive).	Works occasionally, depending on memory pressure.	Works most frequently and must be very fast.
Example: Loading 5 jobs from 20 waiting in disk to memory.	**Example:** Swapping out an idle process to make room for an active one.	**Example:** Choosing the next process using Round Robin or FCFS.

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

Answer:

These criteria measure how efficiently CPU resources are being used.

1. CPU Utilization

Shows how busy the CPU is.
Goal → Maximize it so the CPU stays active instead of idle.

2. Throughput

Number of processes completed per unit time.
Goal → Maximize it to finish more work quickly.

3. Turnaround Time

Total time from process submission to completion.
Goal → Minimize it so results are produced faster.

4. Waiting Time

Time a process spends waiting in the ready queue.
Goal → Minimize it to reduce delays.

5. Response Time

Time from submission to first output or response.
Goal → Minimize it for better user experience (especially in interactive systems).

Section-D (CPU Scheduling Calculations)

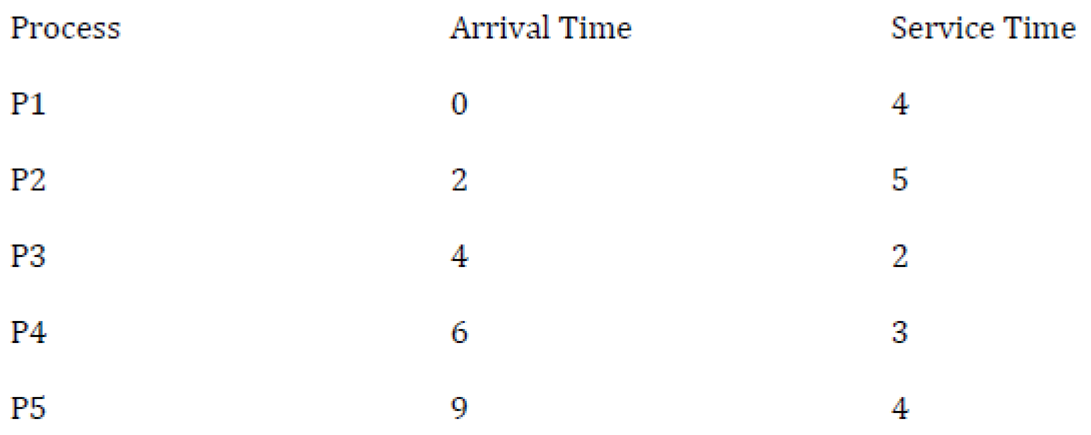
Perform Calculations for Part (A to C)

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

Gantt Chart:

(PART - A)

Gantt chart.

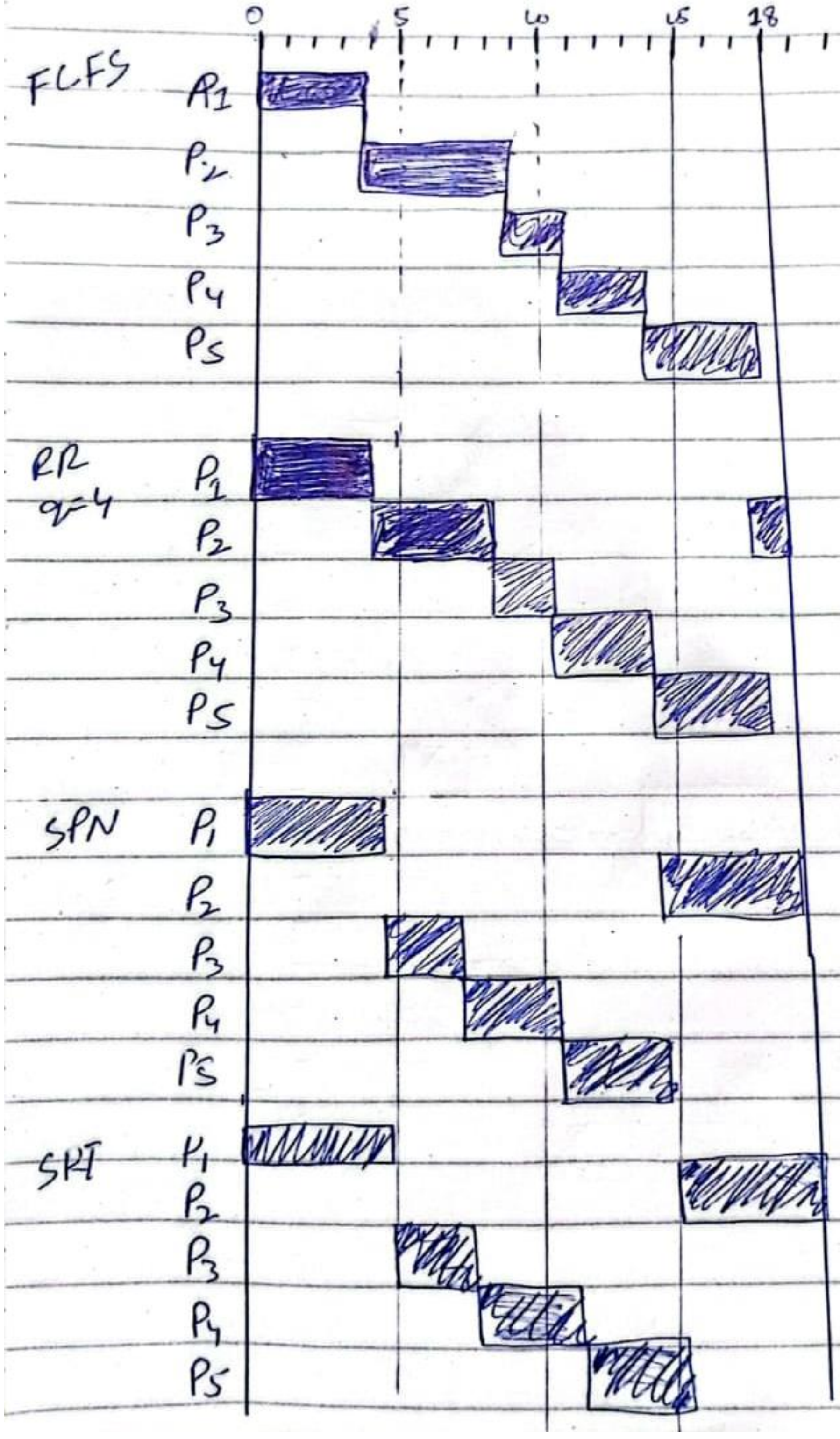
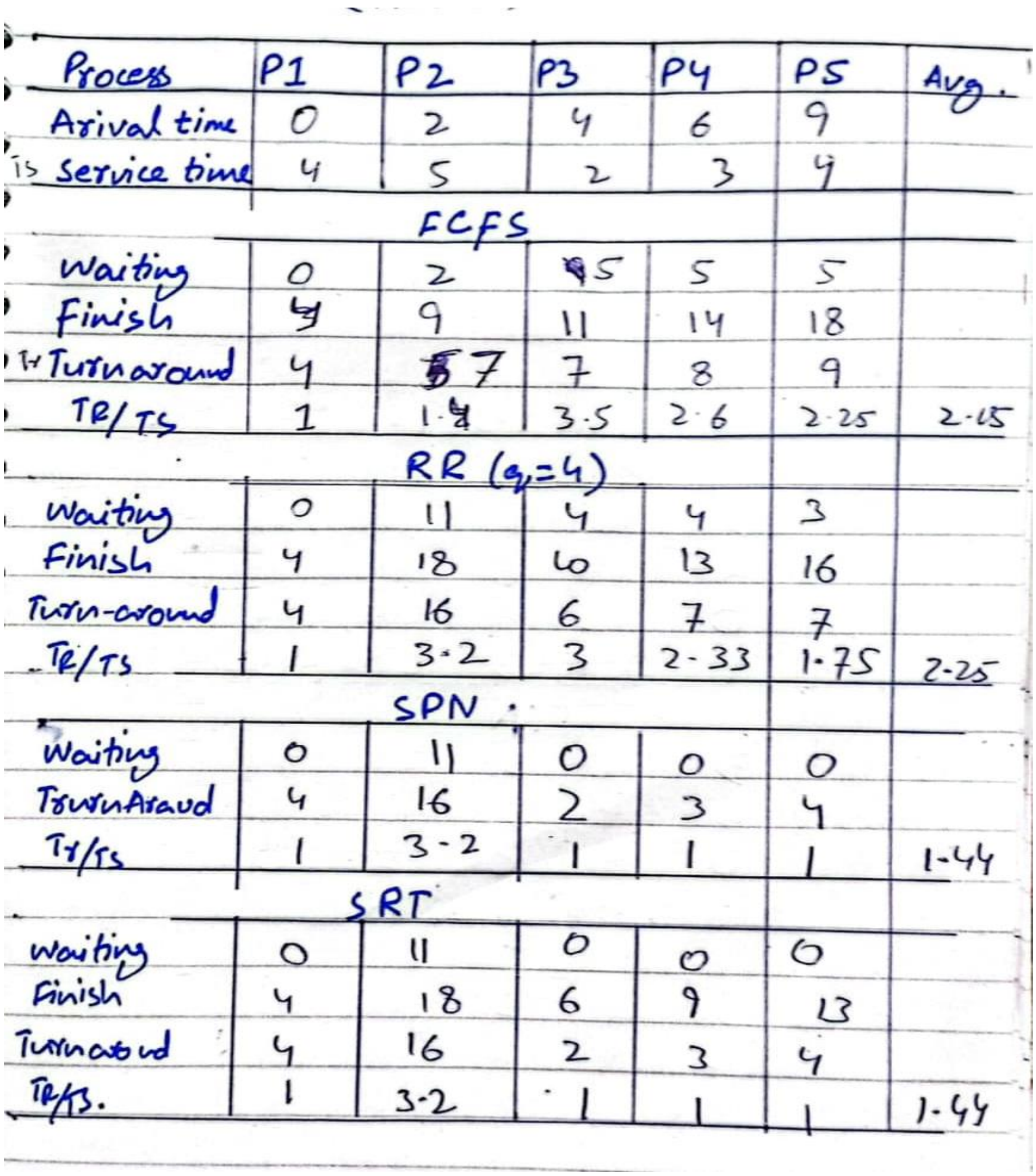


Table for Tr/Ts, Turnaround time, waiting time:



Process	P1	P2	P3	P4	P5	Avg.
Arival time	0	2	4	6	9	
Ts Service time	4	5	2	3	4	
			FCFS			
Waiting	0	2	5	5	5	
Finish	4	9	11	14	18	
Tr Turnaround	4	7	7	8	9	
TR/TS	1	1.4	3.5	2.6	2.25	2.15
			RR (q=4)			
Waiting	0	11	4	4	3	
Finish	4	18	10	13	16	
Turn-around	4	16	6	7	7	
TR/TS	1	3.2	3	2.33	1.75	2.25
			SPN			
Waiting	0	11	0	0	0	
TurnAraud	4	16	2	3	4	
Tr/Ts	1	3.2	1	1	1	1.44
			SRT			
Waiting	0	11	0	0	0	
Finish	4	18	6	9	13	
Turnaround	4	16	2	3	4	
TR/TS.	1	3.2	1	1	1	1.44

In Part A, two algorithms are best **Shortest Process next** and **Shortest Remaining time** both of them have value 1.44 Units that is close to 1

Part-B

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

Gantt Chart:

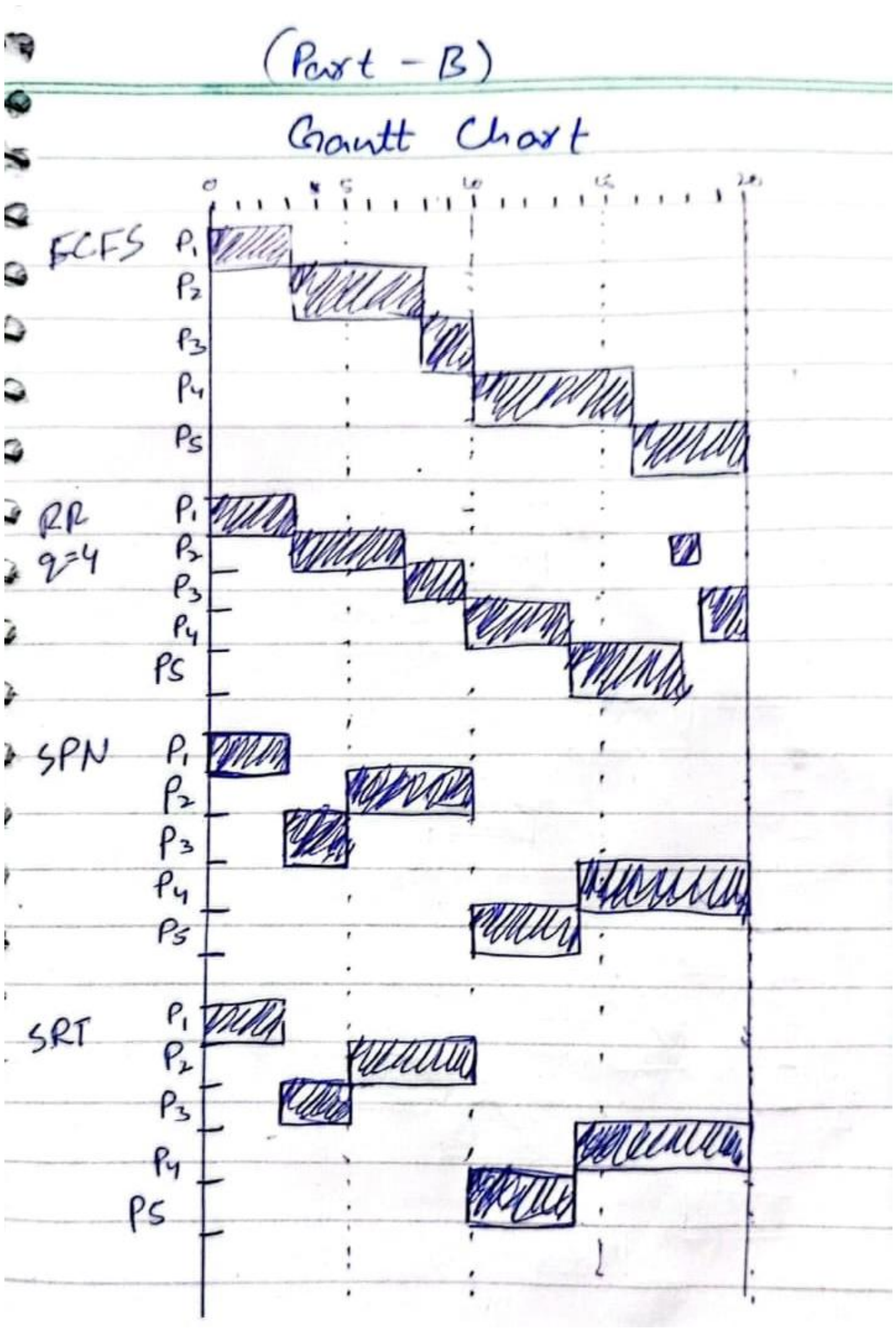
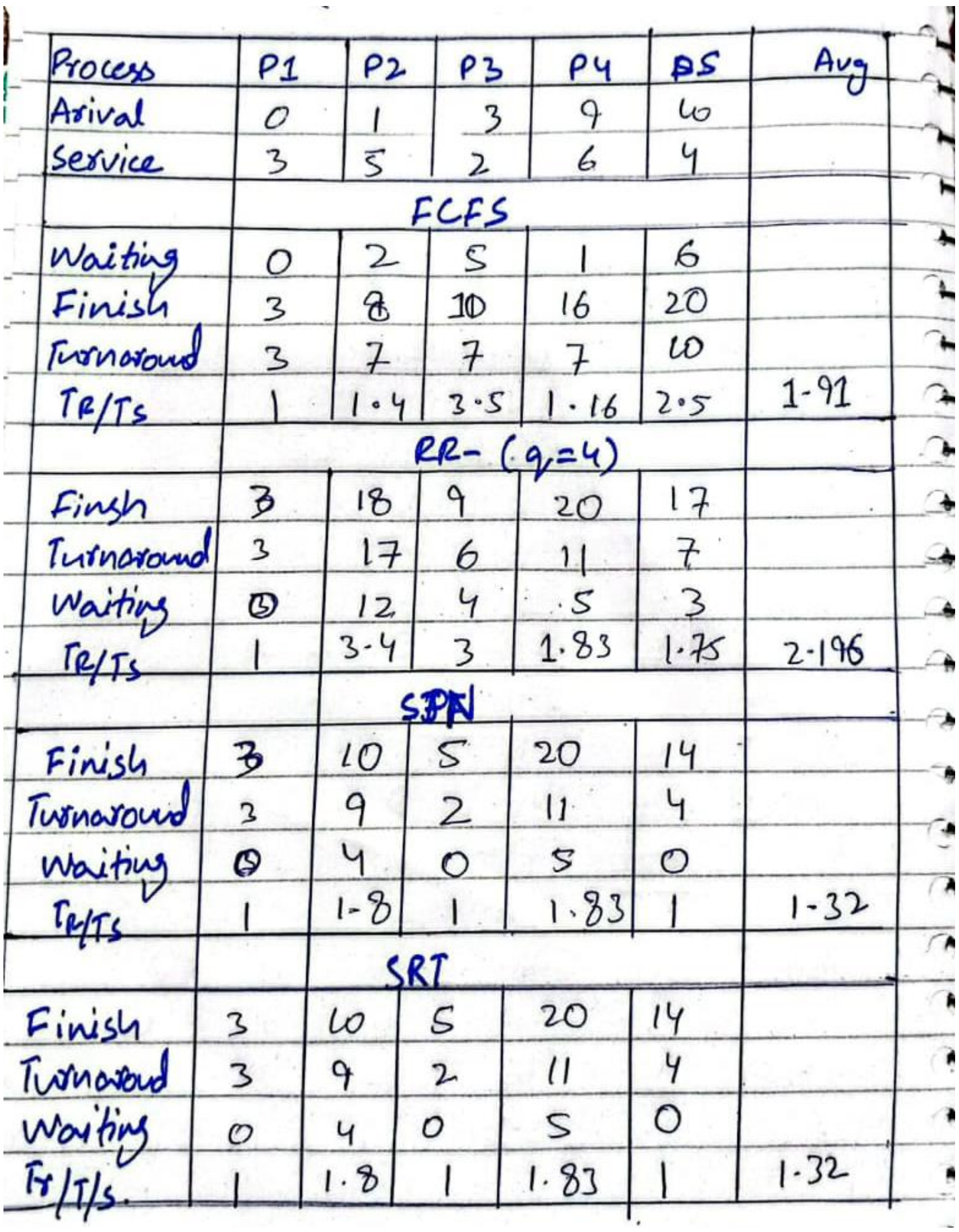


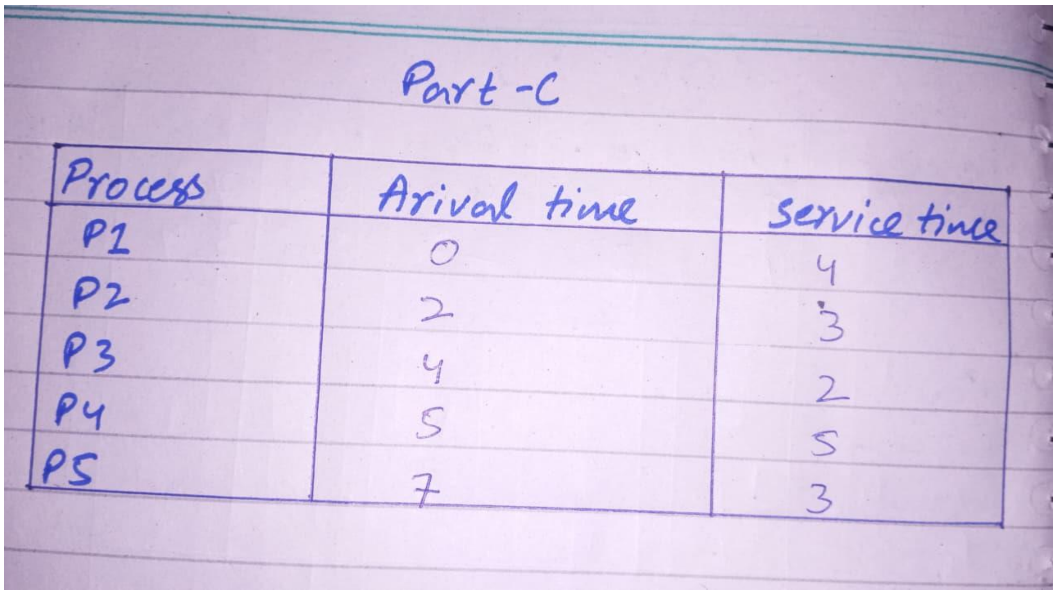
Table for Tr/Ts, Turnaround time, waiting time:



Process	P1	P2	P3	P4	P5	Avg
Arival	0	1	3	9	10	
Service	3	5	2	6	4	
			FCFS			
Waiting	0	2	5	1	6	
Finish	3	8	10	16	20	
Turnaround	3	7	7	7	10	
TR/Ts	1	1.4	3.5	1.16	2.5	1.91
			RR-(q=4)			
Finsh	3	18	9	20	17	
Turnaround	3	17	6	11	7	
Waiting	0	12	4	5	3	
TR/Ts	1	3.4	3	1.83	1.75	2.196
			SPN			
Finish	3	10	5	20	14	
Turnaround	3	9	2	11	4	
Waiting	0	4	0	5	0	
TR/Ts	1	1.8	1	1.83	1	1.32
			SRT			
Finish	3	10	5	20	14	
Turnaround	3	9	2	11	4	
Waiting	0	4	0	5	0	
Tr/T/s.	1	1.8	1	1.83	1	1.32

In Part B, Also SPN and SRT are the two algorithms that are best performing because the TR/TS ratio is close to 1 and it is 1.32 Units

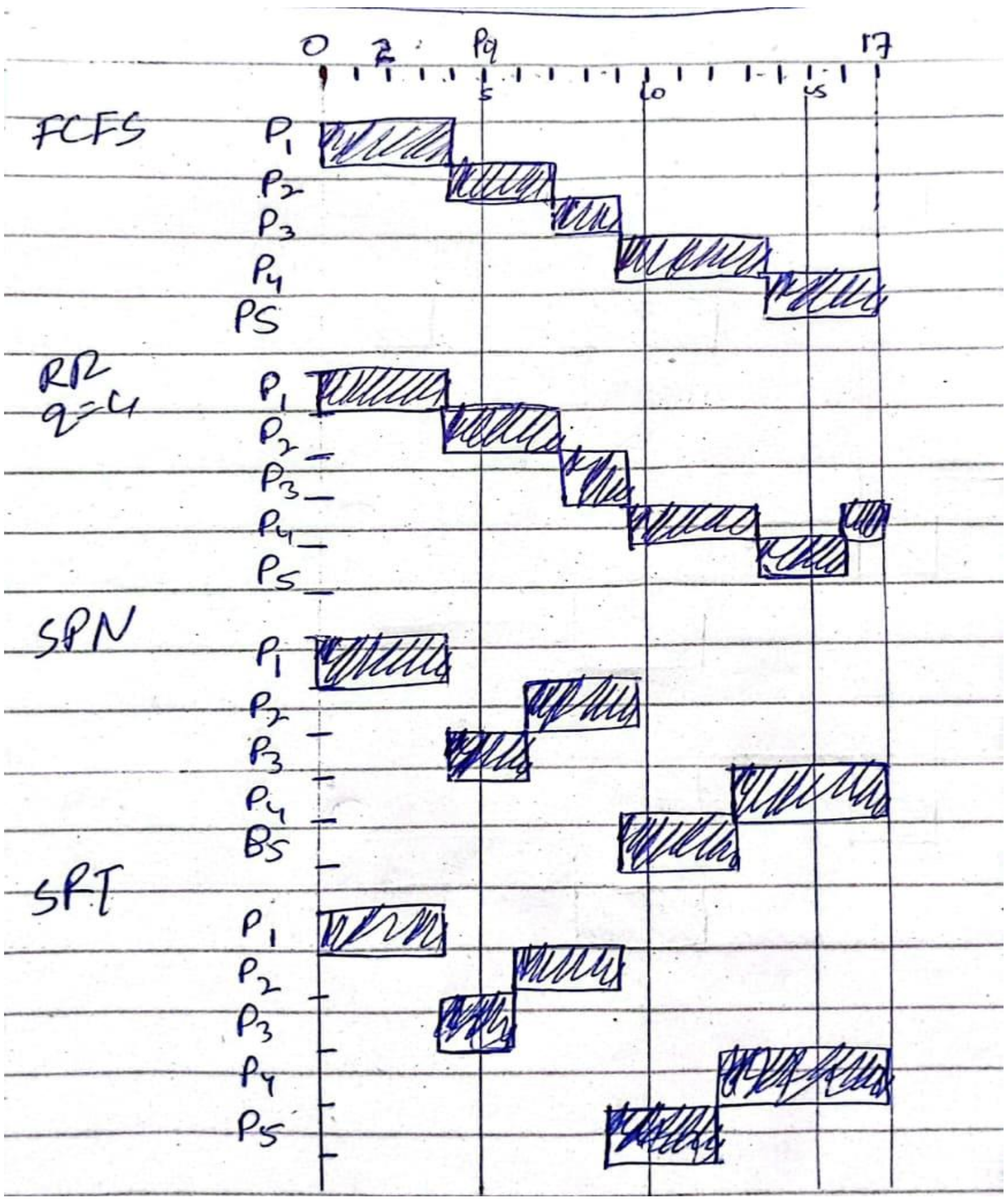
Part-C:



Part -C

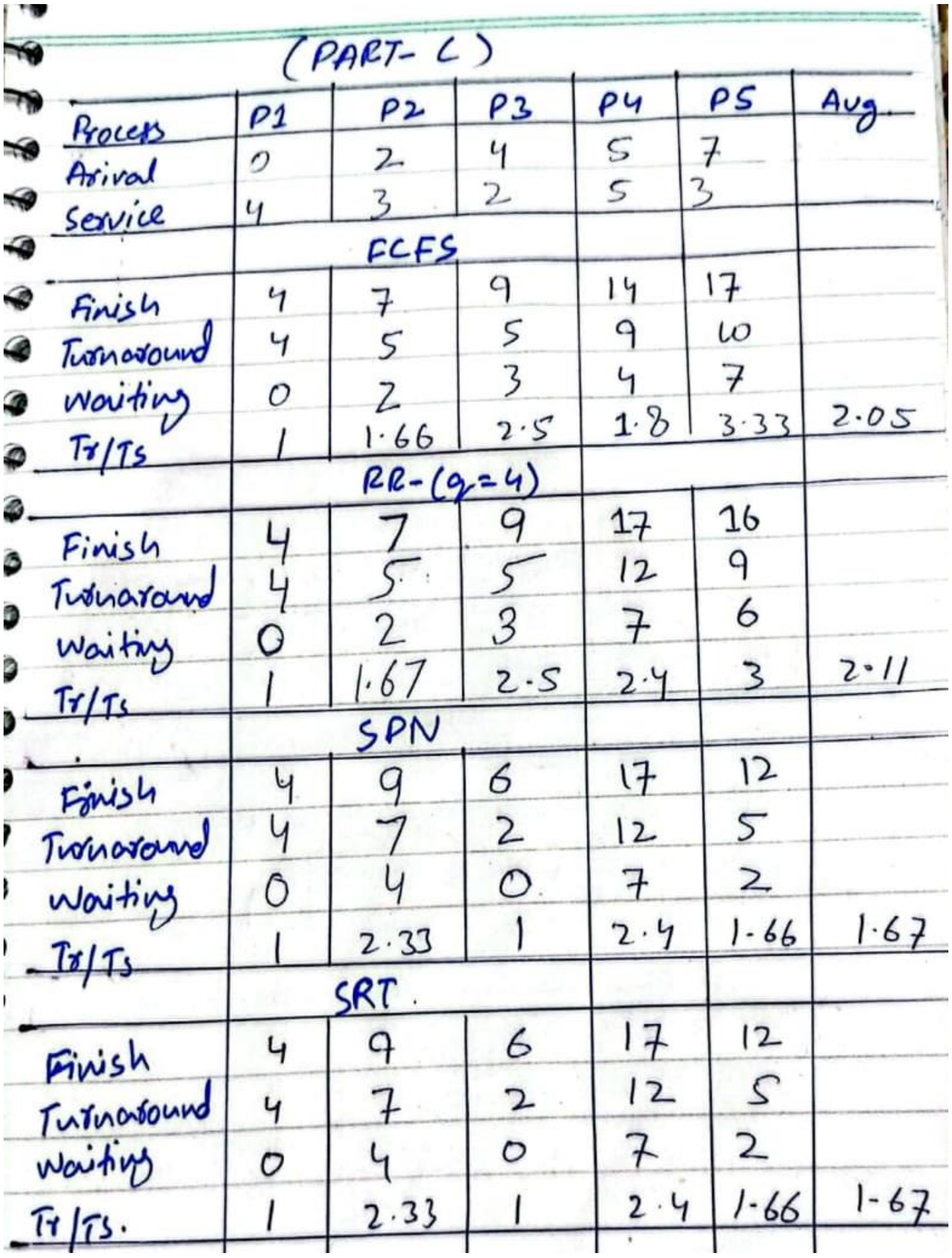
Process	Arival time	service time
P1	0	4
P2	2	3
P3	4	2
P4	5	5
P5	7	3

Gantt Chart:



0
2
P9
17
5
10
15
FCFS
P1
P2
P3
P4
P5
RR
q=4
P1
P2
P3
P4
P5
SPN
P1
P2
P3
P4
P5
SRT
P1
P2
P3
P4
P5

Table for Tr/Ts, Turnaround time, waiting time:



(PART- C)

Process	P1	P2	P3	P4	P5	Avg.
Arival	0	2	4	5	7	
Service	4	3	2	5	3	
	FCFS					
Finish	4	7	9	14	17	
Turnaround	4	5	5	9	10	
Waiting	0	2	3	4	7	
Tr/Ts	1	1.66	2.5	1.8	3.33	2.05
	RR-(q=4)					
Finish	4	7	9	17	16	
Turnaround	4	5	5	12	9	
Waiting	0	2	3	7	6	
Tr/Ts	1	1.67	2.5	2.4	3	2.11
	SPN					
Finish	4	9	6	17	12	
Turnaround	4	7	2	12	5	
Waiting	0	4	0	7	2	
Tr/Ts	1	2.33	1	2.4	1.66	1.67
	SRT.					
Finish	4	9	6	17	12	
Turnaround	4	7	2	12	5	
Waiting	0	4	0	7	2	
Tr/Ts.	1	2.33	1	2.4	1.66	1.67

In Part C, Also SPN and SRT are the two algorithms that are best performing because the TR/TS ratio is close to 1 and it is 1.67 Units