National Textile University, Faisalabad

Department of Computer Science
Course: Operating Systems (CSC-3075)
Semester: Fall 2025
BSSE 5th A
Instructor: Mr. Nasir Mahmood

Assignment-1

Section-A: Programming Tasks

- Instructions:

- Complete all tasks in C using the pthread library.
- Properly comment on your code and include your name, registration number, and task title at the top of each file.
- Use clear screenshots of both code and execution output (CodeSnap preferred).

Task 1 – Thread Information Display

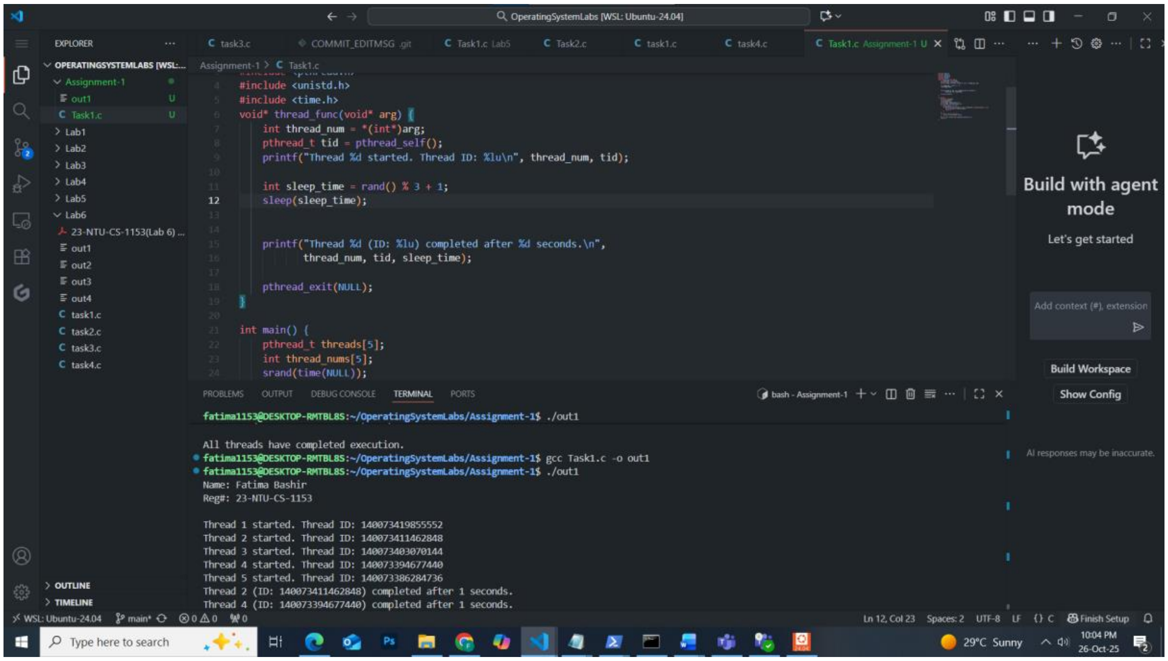
Write a program that creates 5 threads. Each thread should:
- Print its thread ID using `pthread_self()`.
- Display its thread number (1st, 2nd, etc.).
- Sleep for a random time between 1–3 seconds.
- Print a completion message before exiting.

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

Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#include <time.h>

void* thread_func(void* arg) {

    int thread_num = *(int*)arg;

    pthread_t tid = pthread_self();

    printf("Thread %d started. Thread ID: %lu\n", thread_num, tid);
```

```c
    int sleep_time = rand() % 3 + 1;
    sleep(sleep_time);


    printf("Thread %d (ID: %lu) completed after %d seconds.\n",
           thread_num, tid, sleep_time);

    pthread_exit(NULL);
}

int main() {
    pthread_t threads[5];
    int thread_nums[5];
    srand(time(NULL));
    printf("Name: Fatima Bashir\n");
    printf("Reg#: 23-NTU-CS-1153\n\n");
    for (int i = 0; i < 5; i++) {
        thread_nums[i] = i + 1;
        if (pthread_create(&threads[i], NULL, thread_func, &thread_nums[i]) != 0) {
            perror("Failed to create thread");
            return 1;
        }
    }
    for (int i = 0; i < 5; i++) {
        pthread_join(threads[i], NULL);
    }
    printf("\nAll threads have completed execution.\n");
    return 0;
}
```



Task 2 – Personalized Greeting Thread

Write a C program that:

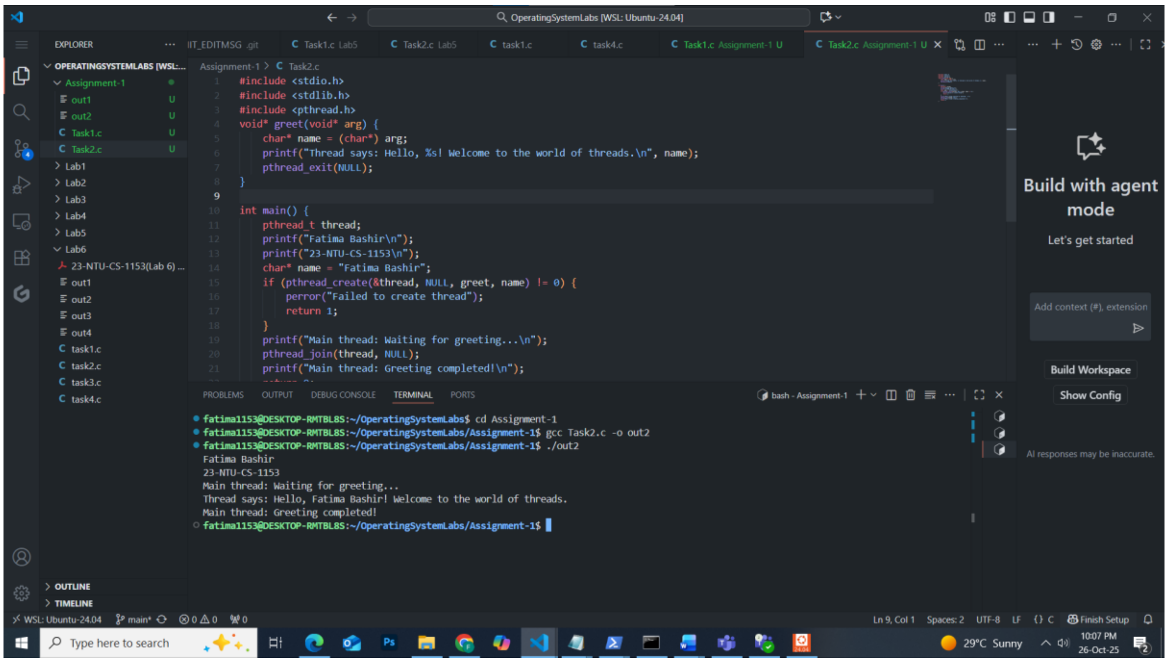
- Creates a thread that prints a personalized greeting message.
- The message includes the user's name passed as an argument to the thread.
- The main thread prints "Main thread: Waiting for greeting…" before joining the created thread.

Example Output:
Main thread: Waiting for greeting…
Thread says: Hello, Ali! Welcome to the world of threads. Main
thread: Greeting completed.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void* greet(void* arg) {
   char* name = (char*) arg;
   printf("Thread says: Hello, %s! Welcome to the world of threads.\n", name);
   pthread_exit(NULL);
}

int main() {
   pthread_t thread;
   printf("Fatima Bashir\n");
```

```c
    printf("23-NTU-CS-1153\n");
    char* name = "Fatima Bashir";
    if (pthread_create(&thread, NULL, greet, name) != 0) {
        perror("Failed to create thread");
        return 1;
    }
    printf("Main thread: Waiting for greeting...\n");
    pthread_join(thread, NULL);
    printf("Main thread: Greeting completed!\n");
    return 0;
}
```



Task 3 – Number Info Thread

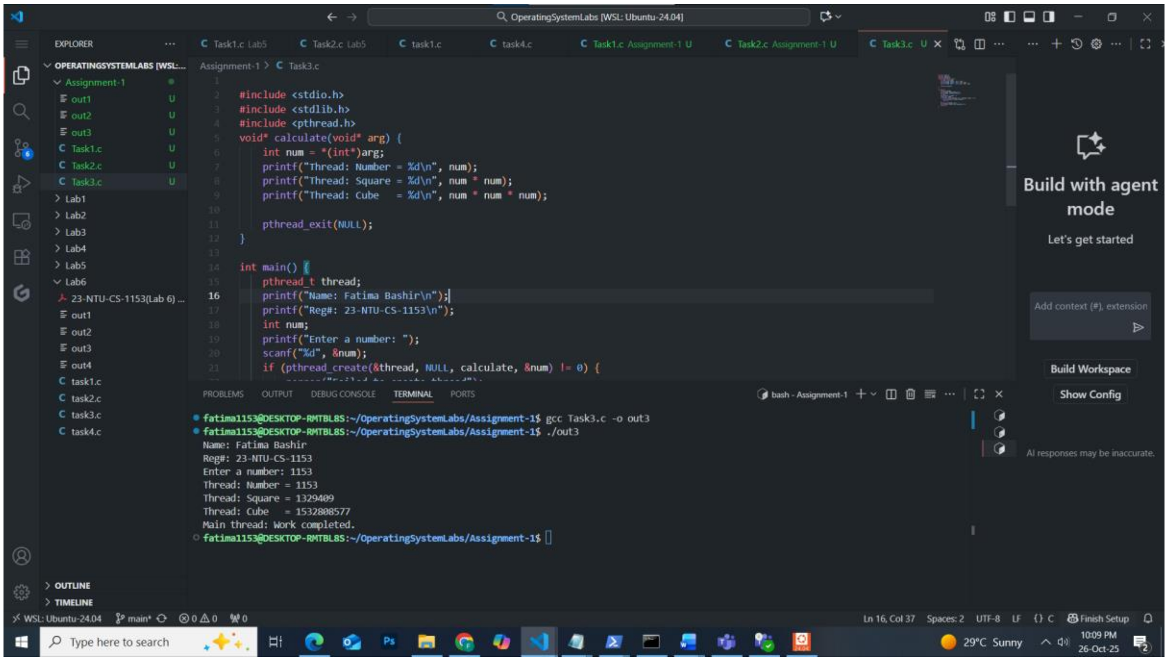
Write a program that:

- 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
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

void* calculate(void* arg) {
```

```
    int num = *(int*)arg;
    printf("Thread: Number = %d\n", num);
    printf("Thread: Square = %d\n", num * num);
    printf("Thread: Cube   = %d\n", num * num * num);

    pthread_exit(NULL);
}

int main() {
    pthread_t thread;
    printf("Name: Fatima Bashir\n");
    printf("Reg#: 23-NTU-CS-1153\n");
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    if (pthread_create(&thread, NULL, calculate, &num) != 0) {
        perror("Failed to create thread");
        return 1;
    }
    pthread_join(thread, NULL);
    printf("Main thread: Work completed.\n");
    return 0;
}
```



Task 4 – Thread Return Values

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

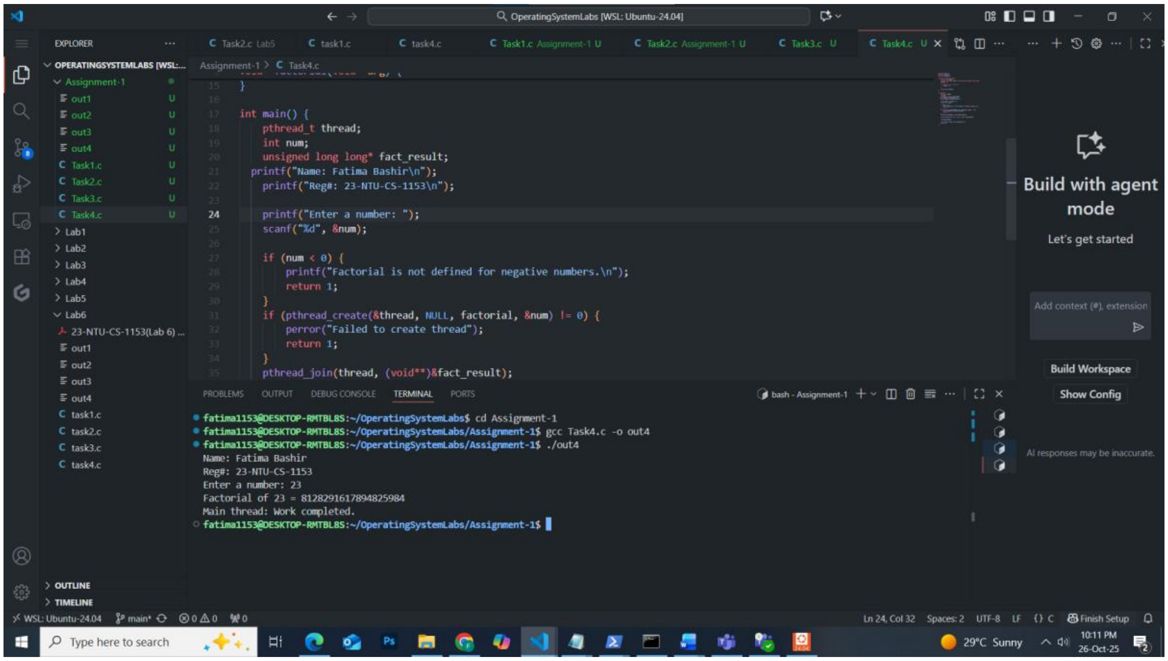
- The thread should return the result using a pointer.
- The main thread prints the result after joining.

Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>


void* factorial(void* arg) {

   int n = *(int*)arg;

   unsigned long long* result = malloc(sizeof(unsigned long long));

   *result = 1;


   for (int i = 1; i <= n; i++) {

      *result *= i;

   }
```

```c
    pthread_exit(result);
}

int main() {
    pthread_t thread;
    int num;
    unsigned long long* fact_result;
  printf("Name: Fatima Bashir\n");
    printf("Reg#: 23-NTU-CS-1153\n");

    printf("Enter a number: ");
    scanf("%d", &num);

    if (num < 0) {
        printf("Factorial is not defined for negative numbers.\n");
        return 1;
    }
    if (pthread_create(&thread, NULL, factorial, &num) != 0) {
        perror("Failed to create thread");
        return 1;
    }
    pthread_join(thread, (void**)&fact_result);

    printf("Factorial of %d = %llu\n", num, *fact_result);

    free(fact_result);

    printf("Main thread: Work completed.\n");
    return 0;
```

```
}
```



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.

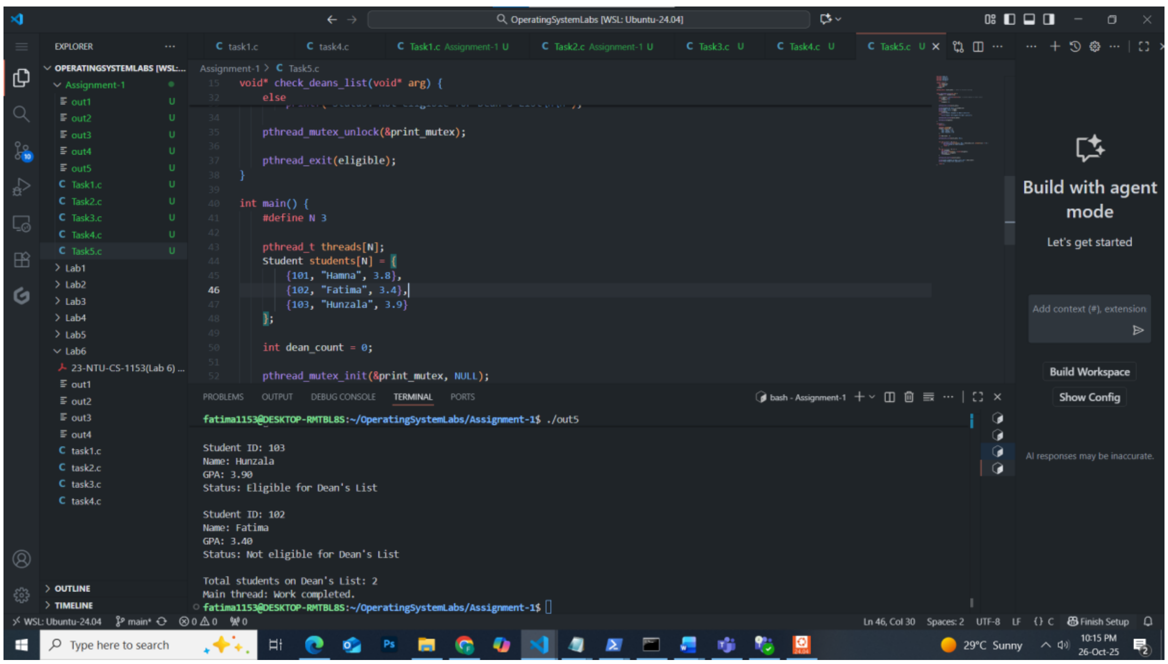
Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <string.h>


typedef struct {

    int student_id;

    char name[50];

    float gpa;

} Student;
```

```c
pthread_mutex_t print_mutex; // mutex to serialize printing


void* check_deans_list(void* arg) {
    Student* s = (Student*)arg;


    int* eligible = malloc(sizeof(int)); // allocate memory to return result
    if (s->gpa >= 3.5)
        *eligible = 1;
    else
        *eligible = 0;


    pthread_mutex_lock(&print_mutex);


    printf("Student ID: %d\n", s->student_id);
    printf("Name: %s\n", s->name);
    printf("GPA: %.2f\n", s->gpa);
    if (*eligible)
        printf("Status: Eligible for Dean's List\n\n");
    else
        printf("Status: Not eligible for Dean's List\n\n");


    pthread_mutex_unlock(&print_mutex);


    pthread_exit(eligible);
}
```

```
int main() {
  #define N 3


  pthread_t threads[N];
  Student students[N] = {
    {101, "Hamna", 3.8},
    {102, "Fatima", 3.4},
    {103, "Hunzala", 3.9}
  };


  int dean_count = 0;


  pthread_mutex_init(&print_mutex, NULL);


  for (int i = 0; i < N; i++) {
    if (pthread_create(&threads[i], NULL, check_deans_list, &students[i]) != 0) {
      perror("Failed to create thread");
      return 1;
    }
  }
  for (int i = 0; i < N; i++) {
    int* eligible;
    pthread_join(threads[i], (void**)&eligible);
    dean_count += *eligible;
    free(eligible);
  }
```

```
    pthread_mutex_destroy(&print_mutex);


    printf("Total students on Dean's List: %d\n", dean_count);

    printf("Main thread: Work completed.\n");


    return 0;

}
```



Section-B: Short Questions

1. **Define an Operating System in a single line.**
A software that manages computer hardware and provides services to programs and users.
2. **What is the primary function of the CPU scheduler?**
To select which process in the ready queue will execute next on the CPU.
3. **List any three states of a process.**
New, Ready, Running.
4. **What is meant by a Process Control Block (PCB)?**
A data structure that stores all information about a process.
5. **Differentiate between a process and a program.**
   - Program: A passive set of instructions.
   - Process: An active execution of a program.
6. **What do you understand by context switching?**
Saving the state of a running process and loading the state of another process.

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

- CPU Utilization: Percentage of time CPU is busy.
- Throughput: Number of processes completed per unit time.

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

Total time taken from process submission to completion.

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

Waiting Time = Turnaround Time − Burst Time.

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

Time from process submission to the first response/output.

11. **What is preemptive scheduling?**

CPU can be taken away from a running process before it finishes.

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

A running process keeps the CPU until it finishes or blocks.

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

1. Fair CPU sharing among processes.
2. Reduces process starvation.

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

Longer processes may suffer starvation.

16. **Define CPU idle time.**

Time during which the CPU is not executing any process.

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

1. Minimize waiting time.
2. Maximize CPU utilization.

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

1. Normal completion.
2. Error or illegal operation.

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

- wait(): Parent waits for child process to finish.
- exit(): Terminates a process and returns status.

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

- Shared memory: Processes communicate via a shared memory area.
- Message passing: Processes communicate by sending/receiving messages.

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

- Thread: Lightweight unit of execution within a process.
- Process: Independent program execution with its own memory.

22. **Define multithreading.**

Running multiple threads concurrently within a single process.

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

- CPU-bound: Requires more CPU time than I/O.
- I/O-bound: Performs more I/O than computation.

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

To perform context switching, switching CPU to the selected process, and resuming execution.

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

- Starvation: Process waits indefinitely due to low priority.
- Aging: Gradually increases priority to prevent starvation.

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

Maximum CPU time allotted to a process in Round Robin scheduling.

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

- Too large: Behaves like FCFS, poor response time.
- Too small: Frequent context switches, high overhead.

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

Ratio of Turnaround Time to Service Time; indicates scheduling efficiency.

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

To hold all processes that are ready to execute on the CPU.

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

- CPU burst: Time process spends using CPU.
- I/O burst: Time process spends performing I/O operations.

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

Round Robin, because every process gets CPU in a fixed cyclic order.

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

fork() then exec() then process runs then exit() then parent may use wait().

33. **Define zombie and orphan processes.**

- Zombie: Finished child process whose parent hasn't called wait().
- Orphan: Process whose parent has terminated, adopted by init.

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

- Priority: CPU allocated based on priority.
- SJF: CPU allocated to the process with the smallest burst time.

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

Time taken to save/restore process states; considered overhead because CPU isn't doing useful work.

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

1. Long-term: Selects processes from job pool to enter ready queue.
2. Medium-term: Temporarily suspends/resumes processes to improve performance.
3. Short-term: Selects ready processes to execute on CPU next.

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

- User Mode: Limited access to system resources.
- Kernel Mode: Full access to hardware and system resources.

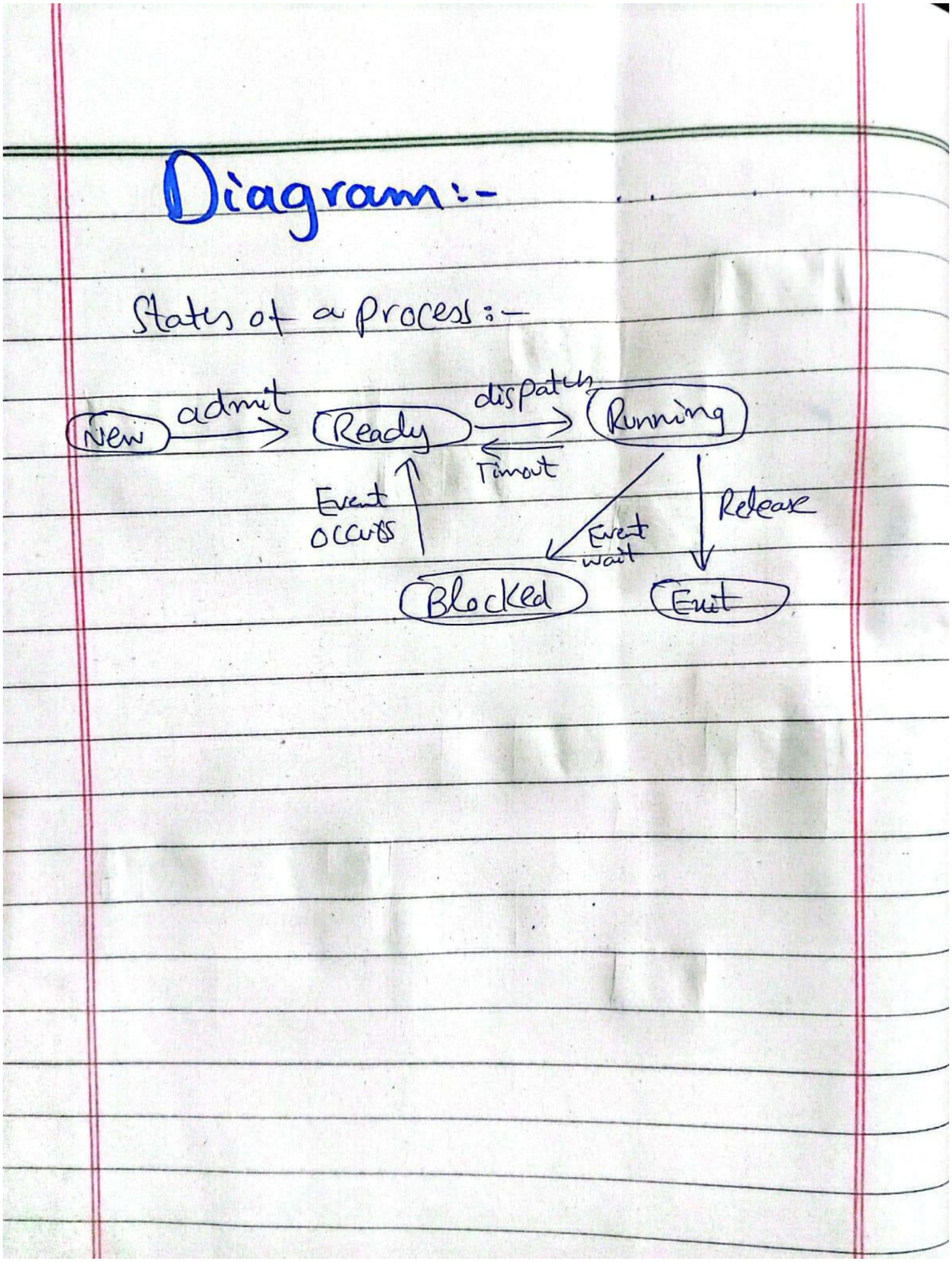
Section-C: Technical / Analytical Questions (4 marks each)

1. Describe the complete life cycle of a process with a diagram

A process goes through the following five states:

States Explanation:

- **New:** Process is being created.
- **Ready:** Process is prepared to run and waiting in the ready queue.
- **Running:** CPU executes the process instructions.
- **Waiting (Blocked):** Process waits for I/O or an event.
- **Terminated:** Process has completed execution.



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

Explanation:

- Context switching occurs when the CPU switches from one process to another.
- It does not perform useful work, hence considered overhead.
- **The state saved/restored includes:**

1. Program Counter (PC)
2. CPU registers
3. Memory management info (page tables)
4. Process state (Ready, Waiting, etc.)

Impact:

- Frequent switches reduce CPU efficiency.
- Short time quantum increases overhead.

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

Components of a Process Control Block (PCB)

A PCB contains all information needed to manage a process:

1. **Process ID (PID):** Unique identifier.
2. **Process state:** New, Ready, Running, Waiting, Terminated**.**
3. **Program counter:** Address of next instruction.
4. **CPU registers:** Contains temporary data, accumulators.
5. **Memory management info:** Base, limit registers, page tables**.**
6. **Scheduling info:** Priority, pointers for queues**.**
7. **I/O status info**: List of I/O devices allocated**.**
8. **Accounting info:** CPU used, time limits, process owner.

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

Scheduler	**Function**	**Example**
Long-Term	Controls admission of processes to the ready queue; manages degree of multiprogramming.	Job scheduler in batch systems.
Medium-Term	Temporarily suspends/resumes processes to improve performance or balance load.	Swapping a process to disk.
Short-Term	Selects a process from the ready queue to execute next on CPU.	Dispatcher in time-sharing systems.

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

CPU Scheduling Criteria

CPU scheduling is evaluated using the following metrics:

1. **CPU Utilization:** Keep CPU as busy as possible (goal: maximize).
2. **Throughput:** Number of processes completed per unit time (maximize).
3. **Turnaround Time:** Total time from submission to completion (minimize).
4. **Waiting Time:** Total time a process spends in ready queue (minimize).
5. **Response Time:** Time from submission to first response/output (minimize).

Optimization Goals:

- Minimize waiting, turnaround, and response times.
- Maximize CPU utilization and throughput.

Section-D: CPU Scheduling Calculations

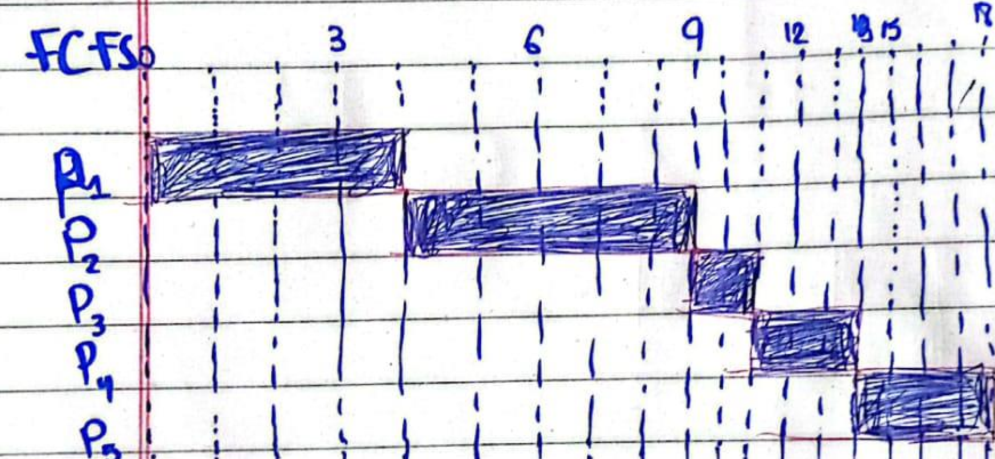
- Perform the following calculations for each part (A–C).
- a) Draw Gantt charts for FCFS, RR (Q=4), SJF, and SRTF.
- b) Compute Waiting Time, Turnaround Time, TR/TS ratio, and CPU Idle Time.
- c) Compare average values and identify which algorithm performs best.

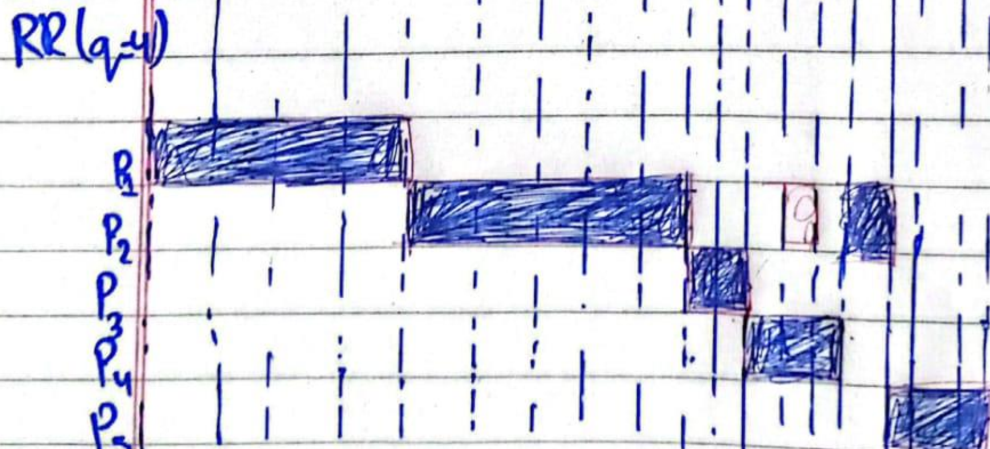
Part-A

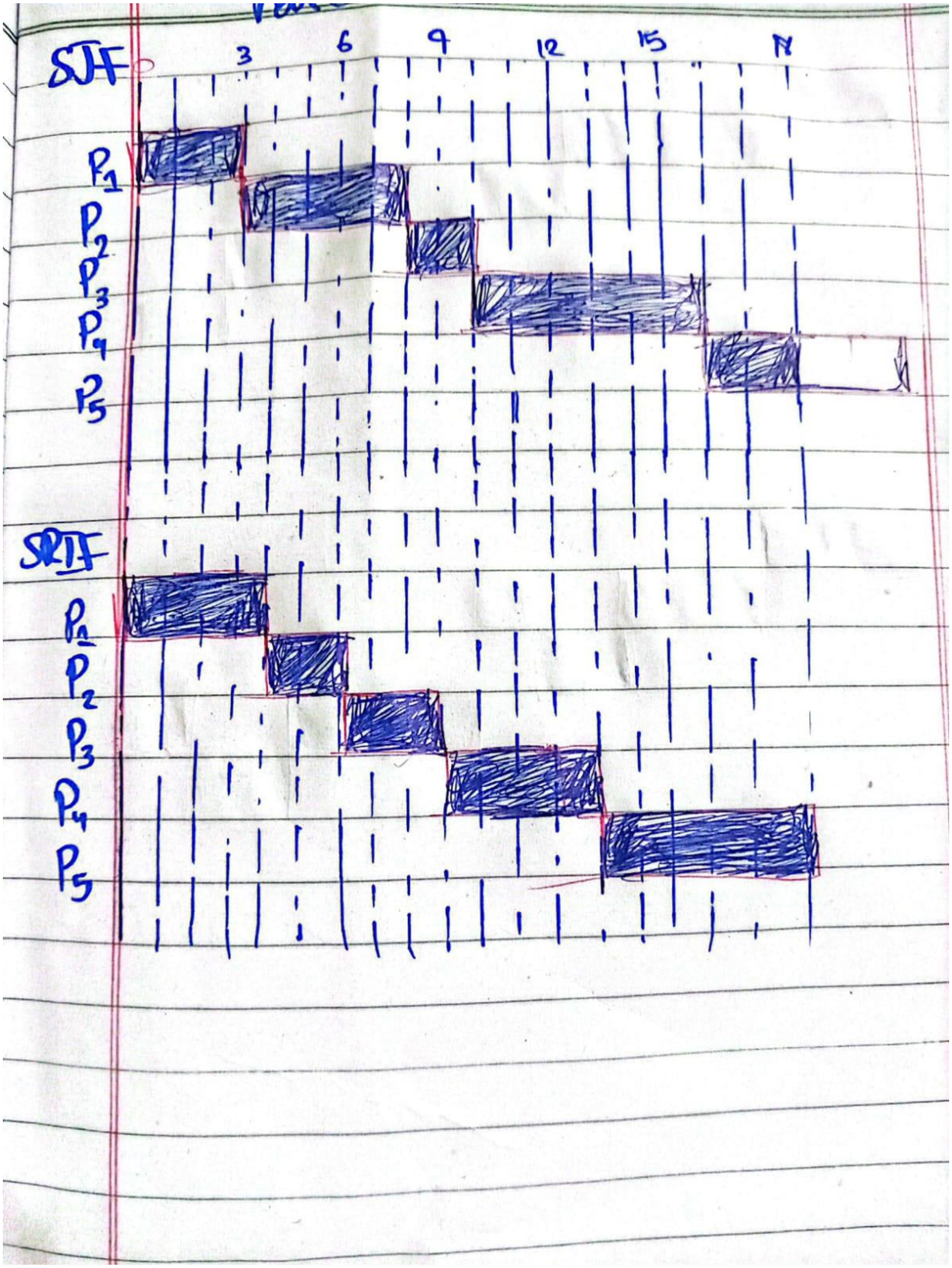
Process	Arrival Time	Service Time
P1	0	4
P2	2	5
P3	4	2
P4	6	3
P5	9	4

PART A
a) Gantt Charts

First Come First Serve (FCFS):
Round Robbin RR (q=4)
Shortest Job first (SJF):
Shortest Remaining Time first:-



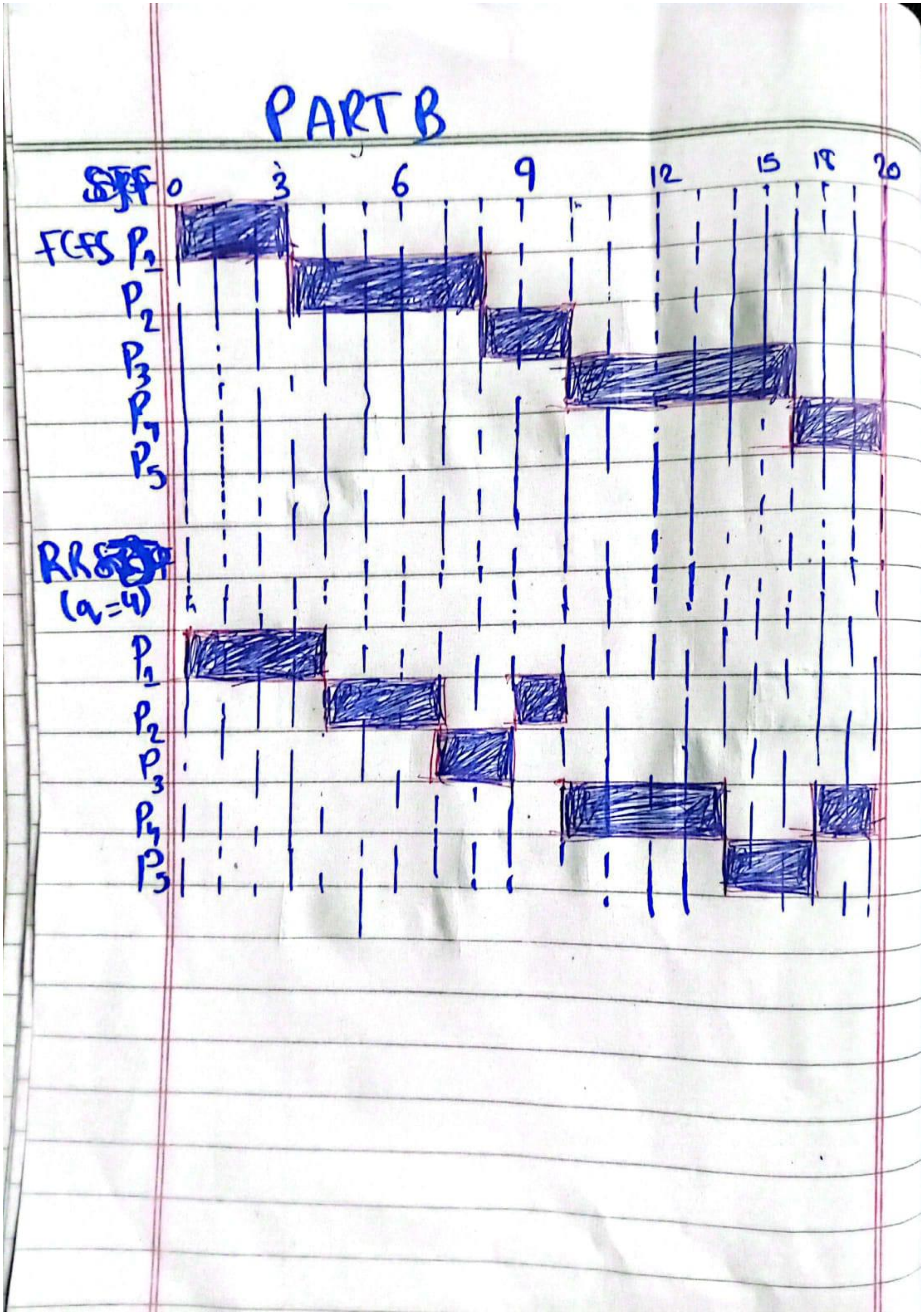


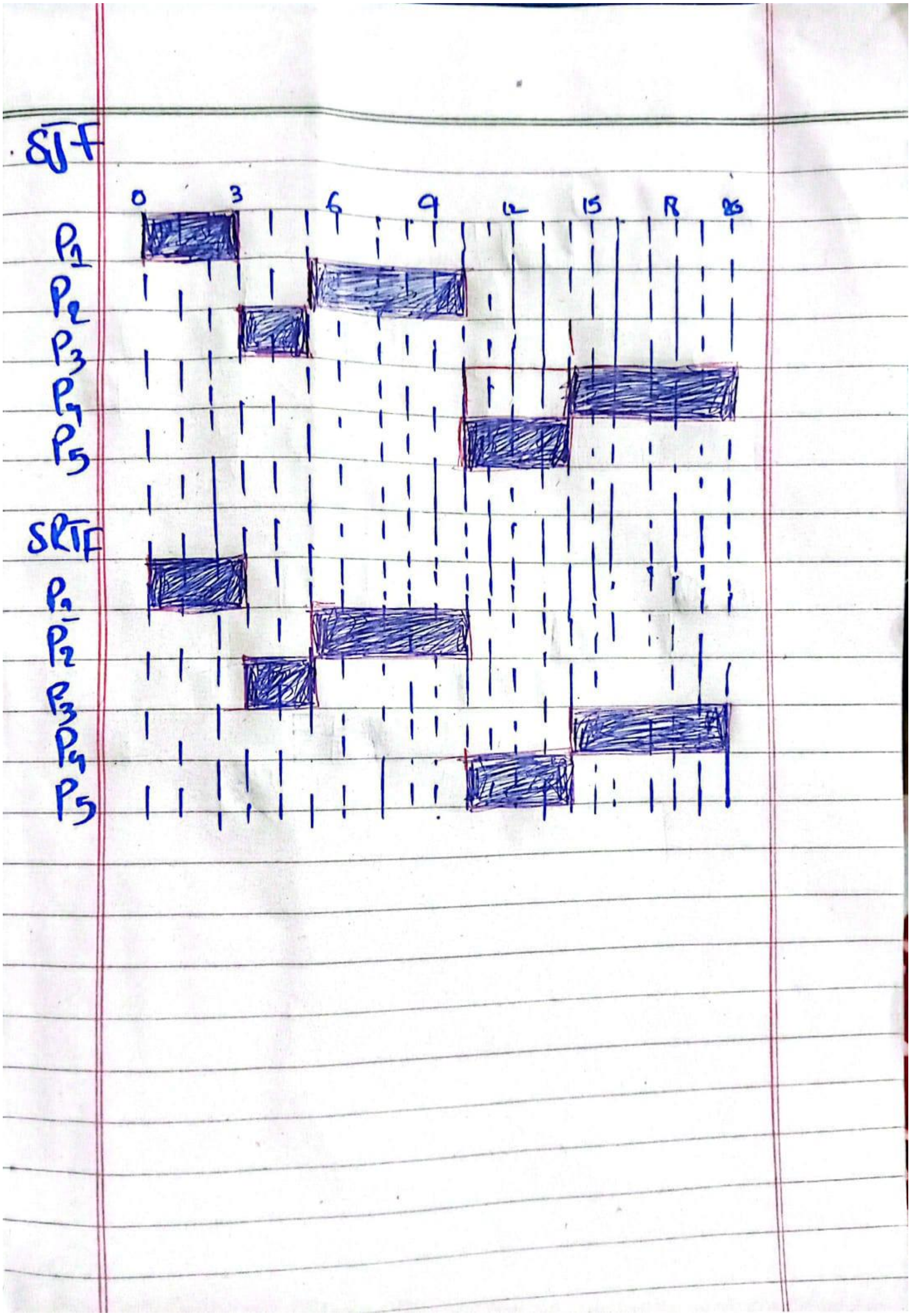


Part-B

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

P5	10	4



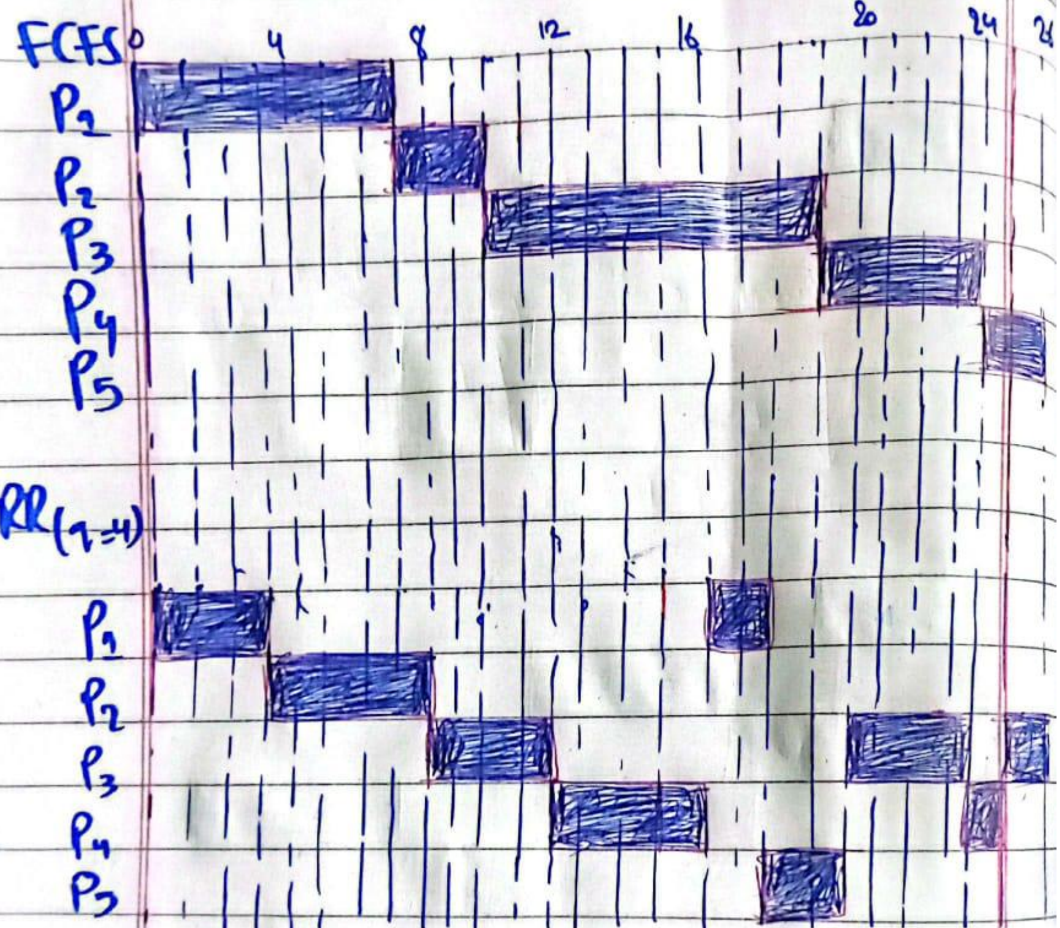


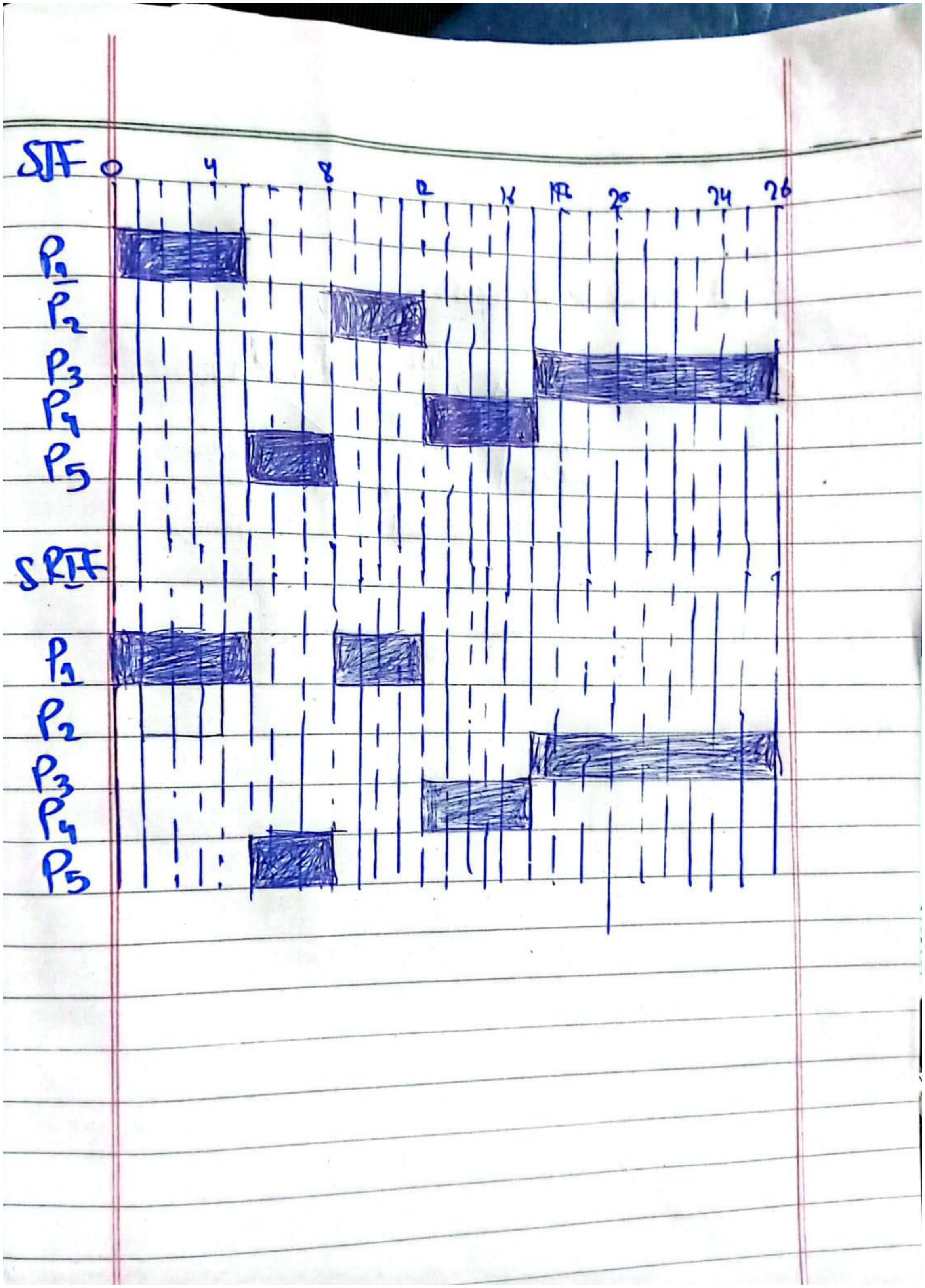
Part-C (Select Your own individual arrivals time and service time)

Process	Arrival Time	Service Time
P1	-	-

P2	-	-
P3	-	-
P4	-	-
P5	-	-

PART C





Submission Guidelines

- Submit a single PDF file on MS Teams including:

  – Screenshots of code and execution for all programming tasks. – Answers to all theory and analytical questions.

- Push all C source files and the PDF to your GitHub repository.
- Late submissions will not be accepted.
- Direct copied from any source will be penalized • VIVA will be held in coming week (week-7)
- Deadline: 26th October 2025, 11:59 PM.