National Textile University

Department of Computer Science

Subject: Operating System

Submitted to: Sir Nasir

Submitted by: Amina

Reg. number: 23-NTU-CS-1136

Lab no.: Assignment

Semester:5th

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:

```c
/*
------------------------------------------------------------
Name: Amina Asif
Registration No: 23-NTU-CS-1136
Task 1 – Thread Information Display
------------------------------------------------------------
*/

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

void* show_thread_info(void* arg) {
    int thread_num = *(int*)arg; // get thread number
    printf("Thread %d started with ID: %lu\n", thread_num, pthread_self());

    // Random sleep between 1–3 seconds
    int sleep_time = rand() % 3 + 1;
    sleep(sleep_time);

    printf("Thread %d completed after %d seconds.\n", thread_num, sleep_time);
    pthread_exit(NULL);
```

```c
}

int main() {
    pthread_t threads[5];
    int thread_nums[5];
    srand(time(NULL)); // for random sleep times

    for (int i = 0; i < 5; i++) {
        thread_nums[i] = i + 1;
        pthread_create(&threads[i], NULL, show_thread_info, &thread_nums[i]);
    }

    // Wait for all threads to finish
    for (int i = 0; i < 5; i++) {
        pthread_join(threads[i], NULL);
    }

    printf("\nAll threads have completed.\n");
    return 0;
}
```

Terminal:

```
amina@DESKTOP-SEP18NK:~/OSLabs$ cd 23-NTU-CS-1136\(Asssignment\)/
amina@DESKTOP-SEP18NK:~/OSLabs/23-NTU-CS-1136(Asssignment)$ gcc Task1.c -o task1 -lpthread
amina@DESKTOP-SEP18NK:~/OSLabs/23-NTU-CS-1136(Asssignment)$ ./task1
Thread 1 started with ID: 138551094146752
Thread 2 started with ID: 138551085754048
Thread 3 started with ID: 138551077361344
Thread 4 started with ID: 138551068968640
Thread 5 started with ID: 138551060575936
Thread 4 completed after 1 seconds.
Thread 1 completed after 2 seconds.
Thread 2 completed after 3 seconds.
Thread 3 completed after 3 seconds.
Thread 5 completed after 3 seconds.

All threads have completed
```

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:

```c
/*
-------------------------------------------------------------
Name: Amina Asif
Registration No: 23-NTU-CS-1136
Task 2 – Personalized Greeting Thread
-------------------------------------------------------------
*/

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

void* greeting(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;
    char name[50];

    printf("Enter your name: ");
    scanf("%s", name);

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

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

Terminal:

```
amina@DESKTOP-SEP18NK:~/OSLabs/23-NTU-CS-1136(Asssignment)$ gcc Task2.c -o task2 -lpthread
amina@DESKTOP-SEP18NK:~/OSLabs/23-NTU-CS-1136(Asssignment)$ ./task2
Enter your name: Amina
Main thread: Waiting for greeting...
Thread says: Hello, Amina! Welcome to the world of threads.
```

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
/*
------------------------------------------------------------
Name: Amina Asif
Registration No: 23-NTU-CS-1136
Task 3 – Number Info Thread
------------------------------------------------------------
*/
#include <stdio.h>
#include <pthread.h>
void* number_info(void* arg){
    int number =*(int*)arg;
    printf("\nNumber is : %d\n",number);
    printf("\nNumber is : %d\n",number*number);
    printf("\nNumber is : %d\n",number*number*number);

}

int main(){
    pthread_t thread;
    int number;
    printf("Enter a Number:");
    scanf("%d",&number);
    //create thread
    pthread_create(&thread,NULL,number_info,&number);
    //wait for thread to finish
    pthread_join(thread,NULL);
    return 0;
}
```

Terminal:

```
amina@DESKTOP-SEP18NK:~/OSLabs/23-NTU-CS-1136(Asssignment)$ gcc Task3.c -o task3 -lpthread
amina@DESKTOP-SEP18NK:~/OSLabs/23-NTU-CS-1136(Asssignment)$ ./task3
Enter a Number:4
/ Number is : 4
/ Number is : 16
/ Number is : 64
```

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
/*
------------------------------------------------------------
Name: Amina Asif
Registration No: 23-NTU-CS-1136
Task 4 – Thread Return Values (Factorial)
------------------------------------------------------------
*/

#include <stdio.h>
#include <pthread.h>
#include <stdlib.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;
    }

    pthread_exit((void*)result);
}

int main() {
    pthread_t thread;
    int num;
    unsigned long long* fact_result;

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

    pthread_create(&thread, NULL, factorial, &num);
    pthread_join(thread, (void**)&fact_result);

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

    return 0;
}
```

Terminal:

```
amina@DESKTOP-SEP18NK:~/OSLabs/23-NTU-CS-1136(Asssignment)$ gcc Task4.c -o t
ask4 -lpthread
amina@DESKTOP-SEP18NK:~/OSLabs/23-NTU-CS-1136(Asssignment)$ ./task4
Enter a number: 4
Factorial of 4 is 24
```

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:

```c
/*
------------------------------------------------------------
Name: Amina Asif
Registration No: 23-NTU-CS-1136
Task 5 – Struct-Based Thread Communication
------------------------------------------------------------
*/

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

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

int deanCount = 0;
pthread_mutex_t lock;

void* check_student(void* arg) {
    Student* s = (Student*)arg;
    printf("\nStudent ID: %d\nName: %s\nGPA: %.2f\n", s->student_id, s->name, s->gpa);

    if (s->gpa >= 3.5) {
        printf("Status: Eligible for Dean's List ☑\n");
        pthread_mutex_lock(&lock);
```

```
        deanCount++;
        pthread_mutex_unlock(&lock);
    } else {
        printf("Status: Not eligible ❌\n");
    }
    pthread_exit(NULL);
}

int main() {
    pthread_t threads[3];
    Student students[3] = {
        {101, "Amina Asif", 3.8},
        {102, "Ali Raza", 3.2},
        {103, "Sara Khan", 3.6}
    };

    pthread_mutex_init(&lock, NULL);

    for (int i = 0; i < 3; i++) {
        pthread_create(&threads[i], NULL, check_student, &students[i]);
    }

    for (int i = 0; i < 3; i++) {
        pthread_join(threads[i], NULL);
    }

    printf("\nTotal students on Dean’s List: %d\n", deanCount);
    pthread_mutex_destroy(&lock);
    return 0;
}
```

Terminal:

```
amina@DESKTOP-SEP18NK:~/OSLabs/23-NTU-CS-1136(Asssignment)$ gcc Task5.c -o task5 -lpthread
amina@DESKTOP-SEP18NK:~/OSLabs/23-NTU-CS-1136(Asssignment)$ ./task5

Student ID: 101
Name: Amina Asif
GPA: 3.80
Status: Eligible for Dean's List ✅

Student ID: 102
Name: Ali Raza
GPA: 3.20
Status: Not eligible ❌

Student ID: 103
Name: Sara Khan
GPA: 3.60
Status: Eligible for Dean's List ✅

Total students on Dean's List: 2
```

Section-B: Short Questions

1. Answer all questions briefly and clearly.

2. Define an Operating System in a single line.

Answer:

An Operating System is system software that manages computer hardware, software resources and provides services to users and programs.

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

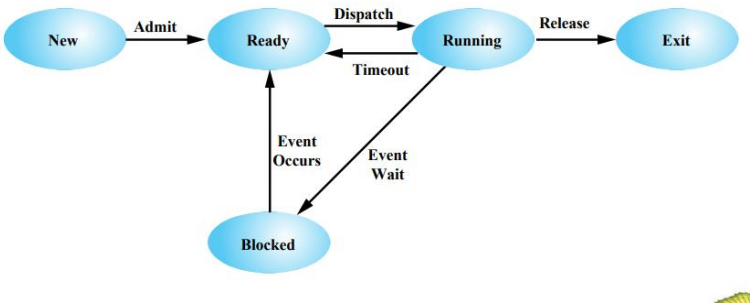
Answer:

The CPU scheduler decides which process from the ready queue should be executed next by the CPU.

4. List any three states of a process.

Answer:

- **Ready** – waiting to be assigned to the CPU.
- **Running** – currently being executed.
- **Waiting/Blocked** – waiting for I/O or event completion.



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

Answer:

PCB is a data structure that stores all information about a process, like process ID, state, registers, memory and CPU scheduling information.

6. Differentiate between a process and a program.

Answer:

Aspect	Program	Process
Definition	A passive set of instructions stored on disk.	An active instance of a program in execution.
State	Static (does not change).	Dynamic (changes as it executes).

Aspect	Program	Process
Stored In	Secondary memory (disk).	Main memory (RAM).
Example	chrome.exe file	Chrome running on your P

7. What do you understand by context switching?

Answer:

It's the process of saving the current state of a process and loading another process's state to resume its execution.

8. Define CPU utilization and throughput.

Answer:

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

9. What is the turnaround time of a process?

Answer:

Turnaround Time = Completion Time − Arrival Time.
It shows total time taken from process submission to completion.

10. How is waiting time calculated in process scheduling?

Answer:

Waiting Time = Turnaround Time – Execution Time.
It represents the time a process spends waiting in the ready queue.

11. Define response time in CPU scheduling.

Answer:

Response Time is the time between process submission and the first response from the system.

12. What is preemptive scheduling?

Answer:

Preemptive scheduling allows a process to be interrupted and moved to the ready queue so another process can run.

13. What is non-preemptive scheduling?

Answer:

Once a process starts executing, it cannot be interrupted until it finishes.

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

Answer:

- Ensures fairness — every process gets CPU time.
- Suitable for time-sharing systems.

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

Answer:

SJF may cause **starvation** for longer processes.

16. Define CPU idle time.

Answer:

The period when the CPU is not executing any process.

17. State two common goals of CPU scheduling algorithms.

Answer:

1. Maximize CPU utilization and throughput.
2. Minimize waiting and turnaround times.

18. List two possible reasons for process termination.

Answer:

1. Normal completion.
2. Runtime error or resource failure.

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

Answer:

- **wait():** Used by the parent to wait for child process completion.
- **exit():** Terminates a process and releases resources.

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

Answer:

Aspect	Shared Memory Model	Message Passing Model
Communication	Processes share a common memory space.	Processes communicate by sending messages.
Speed	Faster (no kernel involvement).	Slower (involves system calls).
Synchronization	Must be managed by programmer.	Handled by OS.
Used In	Systems with tightly coupled processes.	Distributed systems.

21. Differentiate between a thread and a process.

Answer:

Aspect	Thread	Process
Definition	Smallest unit of execution within a process.	Independent program in execution.
Memory	Shares memory and resources with other threads.	Has its own memory space.
Overhead	Low.	High.

Communication	Easier (shared memory).	Requires IPC mechanisms.

22. Define multithreading.

Answer:

Multithreading allows multiple threads within the same process to execute concurrently.

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

Answer:

Aspect	CPU-Bound Process	I/O-Bound Process
Focus	Performs heavy computation.	Waits more for I/O operations.
CPU Usage	High.	Low.
I/O Usage	Low.	High.
Example	Mathematical calculations.	File download or printing.

24. What are the main responsibilities of the dispatcher?

Answer:

Dispatcher gives CPU control to the selected process by performing context switching, changing modes, and jumping to the program counter.

25. Define starvation and aging in process scheduling.

Answer:

Aspect	Starvation	Aging
Definition	A process never gets CPU time.	Priority of waiting process increases with time.
Cause	Low priority or long waiting.	Introduced to prevent starvation.

Aspect	Starvation	Aging
Effect	Some processes may never execute.	Ensures fairness.
Example	In Priority Scheduling.	Used in improved scheduling.

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

Answer:

A fixed time period for which a process can execute before being preempted in Round Robin scheduling.

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

Answer:

- **Too large:** Becomes like FCFS (less responsive).
- **Too small:** Increases context switch overhead.

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

Answer:

Turnaround Ratio = Turnaround Time / Service Time.
It indicates how efficiently processes are handled.

29. What is the purpose of a ready queue?

Answer:

It holds all processes that are ready and waiting to execute on the CPU.

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

Answer:

 CPU Burst: Process executes instructions using CPU.

 I/O Burst: Process waits for input/output operations.

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

Answer:

Round Robin because every process gets equal CPU time in rotation.

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

Answer:

- Parent process calls **fork()** to create a child.
- Child process gets a copy of parent's address space.
- **exec()** loads a new program into child's space.

33. Define zombie and orphan processes.

Answer:

- **Zombie:** Process finished execution but parent hasn't read its exit status.
- **Orphan:** Process whose parent has terminated.

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

Answer:

Aspect	Priority Scheduling	Shortest Job First (SJF)
Basis	Based on process priority.	Based on shortest CPU burst.
Starvation	Possible for low-priority processes.	Possible for long processes.
Type	Can be preemptive or non-preemptive.	Usually non-preemptive.
Example Use	Real-time systems.	Batch systems.

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

Answer:

Context switch time is the duration to save and load process states.
It’s overhead because no useful work is done during this time.

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

Answer:

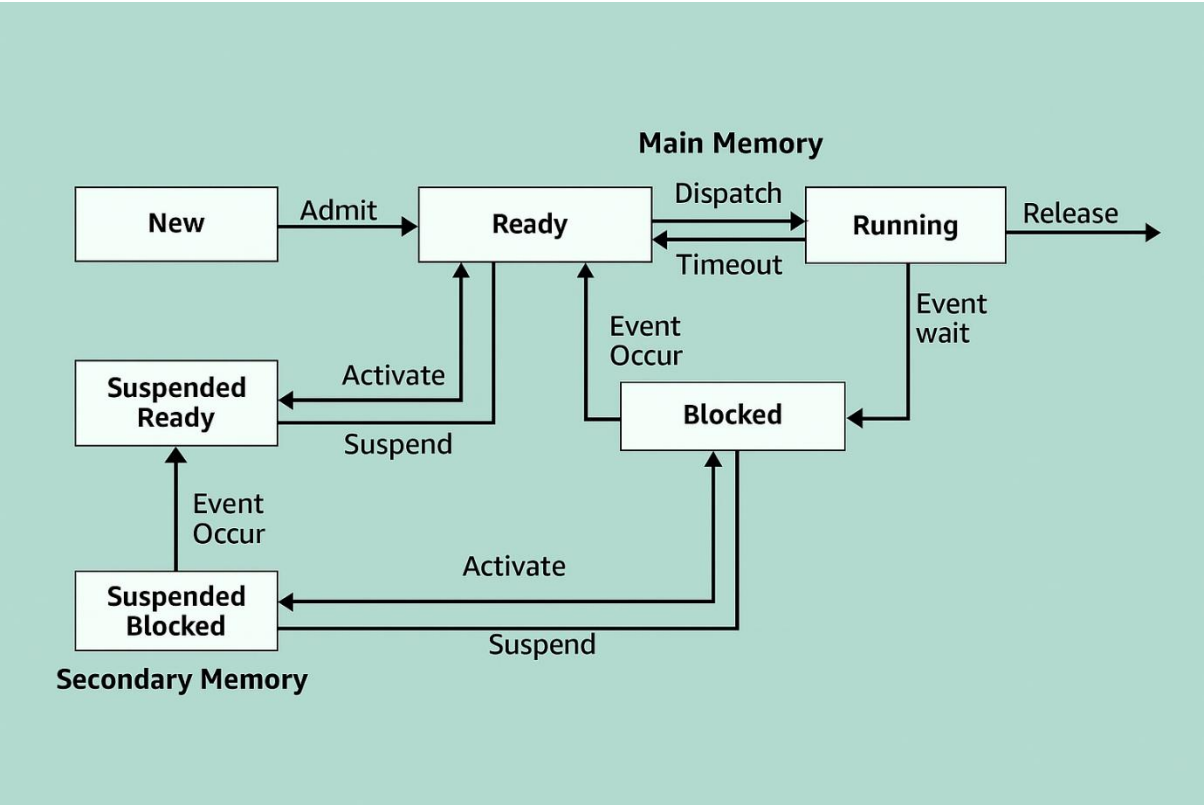
1. **Long-Term:** Selects jobs from the job pool for admission.
2. **Medium-Term:** Suspends/resumes processes for memory management.
3. **Short-Term:** Chooses ready process for CPU execution.

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

Aspect	User Mode	Kernel Mode
Access	Limited to user applications.	Full access to hardware and memory.
Example	Running a word processor.	Executing OS instructions.
Privilege Level	Low.	High.
System Calls	Must request kernel for operations.	Executes system-level operations directly.

1. Describe the complete life cycle of a process with a neat diagram showing transitions between New, Ready, Running, Waiting, and Terminated states.

Answer:



Explanation of States:

State	Description
New	Process is being created and loaded into memory.
Ready	Process is ready to execute but waiting for CPU allocation.
Running	CPU is executing process instructions.
Waiting (Blocked)	Process is waiting for an event (e.g., I/O completion).
Terminated	Process has finished execution and is removed from memory.

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

Definition:

A **context switch** occurs when the CPU switches from executing one process to another.

It saves the **current process's state** and loads the **next process's state**.

Overhead Explanation:

- Context switching takes **CPU time**.
- Frequent switches reduce performance.
- The time spent saving/restoring states is **called** context switch overhead.

Information Saved & Restored:

Saved Information	Description
Program Counter (PC)	Address of next instruction to execute.
CPU Registers	General-purpose and status registers.
Stack Pointer	Points to the top of the stack.
Memory Management Info	Page tables, base & limit registers.
I/O Status Info	Pending I/O requests.
Accounting Info	CPU usage, scheduling info, etc.

When Restored:
When the process is rescheduled, all saved data is reloaded from the PCB, and execution resumes as if it was never interrupted.

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

Definition:

A **PCB** is a data structure maintained by the operating system for every process. It stores **process-related information** required for management.

Components:

Component	Description
Process ID (PID)	Unique identifier for the process.
Process State	New, Ready, Running, Waiting, or Terminated.
Program Counter	Next instruction address.
CPU Registers	Stored register contents for context switching.
Memory Management Info	Page tables, segment tables, base/limit registers.
Accounting Info	CPU usage, time limits, process priority.
I/O Status Info	Devices allocated, open file list.
Parent/Child Info	Links to parent and child processes.

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

Answer:

Types of Schedulers

Type	Function	Example / Role	Frequency
Long-Term Scheduler (Job Scheduler)	Decides which processes are admitted to the ready queue. Controls **degree of multiprogramming**.	Example: Batch system deciding which jobs to load from disk to memory.	Infrequent
Medium-Term Scheduler	Handles **suspension and resumption** of processes. Moves processes between main memory and secondary storage.	Example: Swapping process out to disk when memory is full.	Occasional
Short-Term Scheduler (CPU Scheduler)	Selects which ready process will run next on CPU.	Example: Round Robin, Priority, or FCFS scheduling.	Very Frequent

5. Explain CPU Scheduling Criteria (Utilization, Throughput, Turnaround, Waiting, and

Response) and their optimization goals.
Answer:

Criterion	Definition	Optimization Goal
CPU Utilization	Percentage of time CPU is busy doing useful work.	**Maximize** (ideal: 100%)
Throughput	Number of processes completed per unit time.	**Maximize**
Turnaround Time	Total time from process submission to completion.	**Minimize**
Waiting Time	Total time a process spends in the ready queue.	**Minimize**
Response Time	Time from process submission until first output (for interactive systems).	**Minimize**

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.

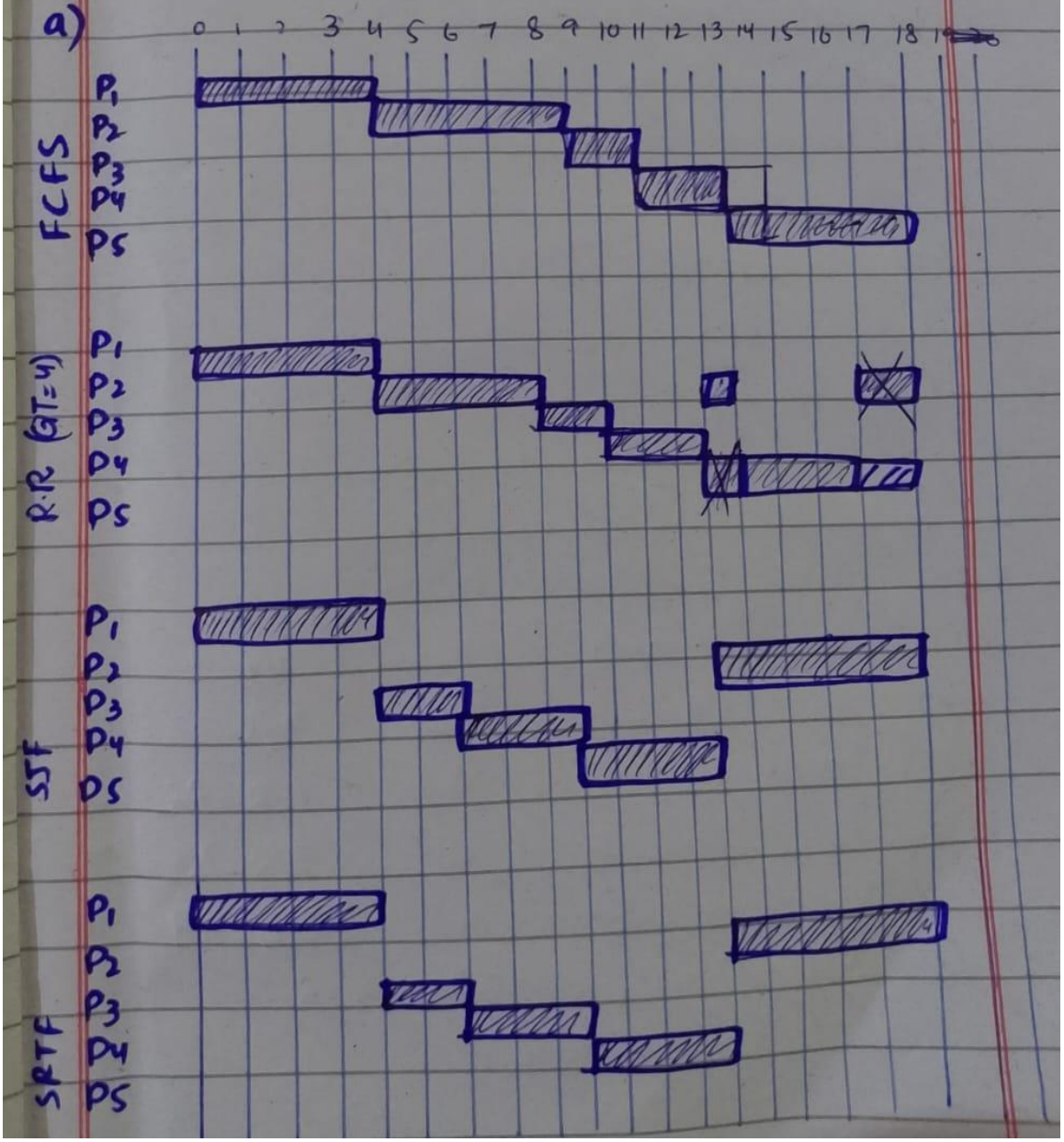
Part-A

Section D

Part - A

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

a)



FCFS

Processes	Finish	Turnaround (Tr) (Finish - Arrival)	Ts	waiting Time	Tr/Ts
P_1	4	4-0 = 4	4	0	1
P_2	9	9-2 = 7	5	2	1.4
P_3	11	11-4 = 7	2	5	3.6
P_4	14	14-6 = 8	3	5	2.67
P_5	18	18-9 = 9	4	5	2.25

Avg Waiting Time = 0+2+5+5+5/5

= 3.4

Avg Turnaround Time = 4+7+7+8+9/5

= 7

Avg Tr/Ts Ratio = 1+1.4+3.6+2.67+2.25/5

= 2.18

CPU idle Time = 0

Roond Robin (Q=4)

P	Finish	Turnaround	Waiting T	Tr/Ts
P_1	4	4-0=4	0	4/4=1
P_2	18	18-2=16	11	16/5=3.2
P_3	10	10-4=6	4	6/2=3.2
P_4	13	13-6=7	4	7/3=2.4
P_5	17	17-9=8	4	8/4=1.9

Avg Waiting Time = 0+11+4+4+4/5

= 4.6

Avg Turnaround Time: = 4+16+6+7+8/5

= 8.2

Avg Tr/Ts = 1+3.2+3+2.4+1.9/5

= 2.51

CPU idle = 0

STF

P	Finish	Turnaround	Waiting	Tr/Ts
P_1	4	4	0	1
P_2	18	16	11	3.2
P_3	6	2	0	1
P_4	9	3	0	1
P_5	13	4	0	1

Avg Waiting Time = 2.2

Avg Turnaround Time = 5.8

Avg Tr/Ts = 1.44

CPU Idle = 0

SRTF

P	Finish	Turnaround Time	Waiting T	Tr/Ts
P_1	4	4	0	1
P_2	18	16	11	3.2
P_3	6	2	0	1
P_4	9	3	0	1
P_5	13	4	0	1

Avg Waiting Time = 2.2

Avg Turnaround Time = 5.8

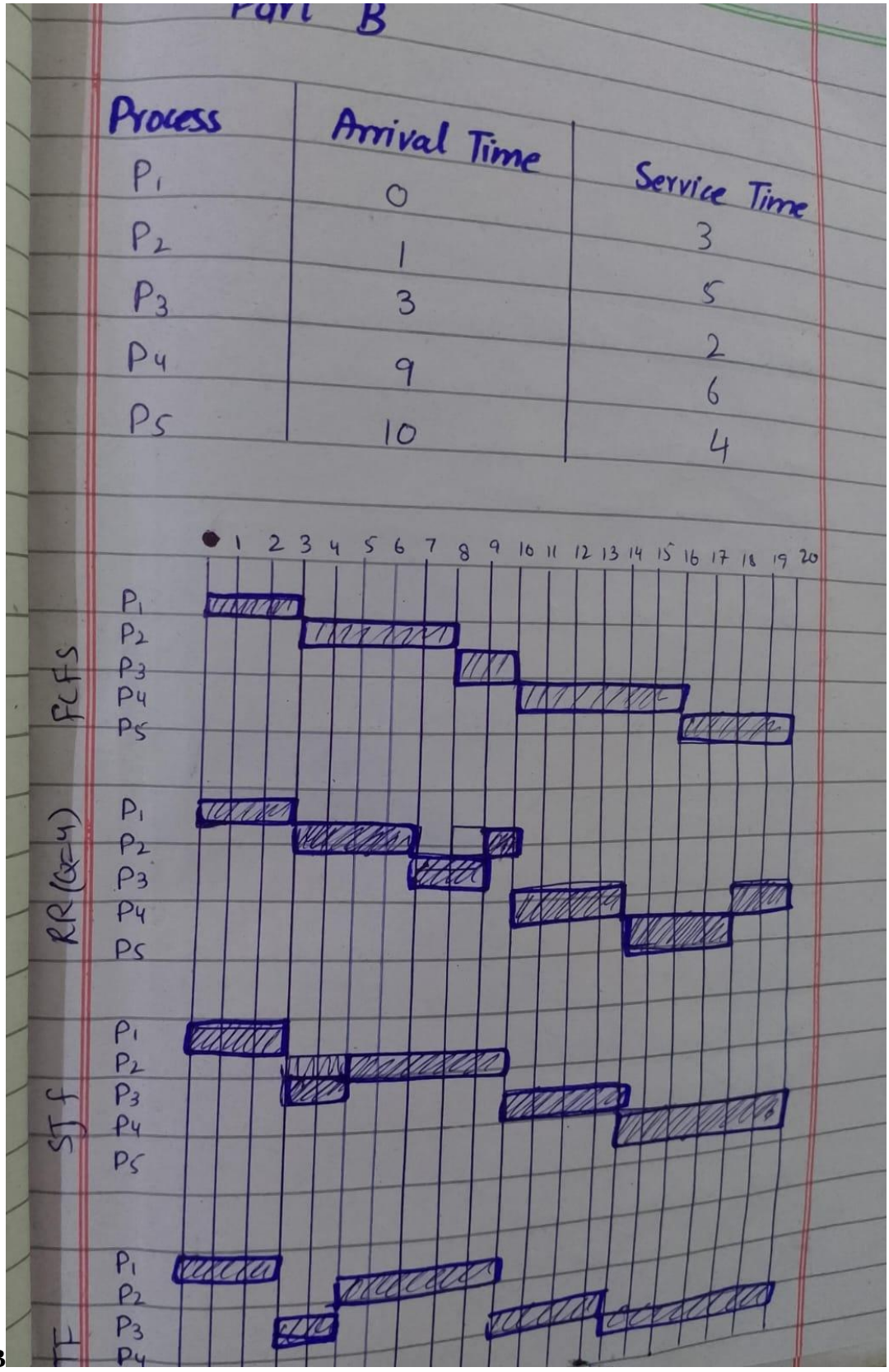
Avg Tr/Ts = 1.44

CPU idle = 0

Best Algorithm

- SRTf and SJf having shortest Avg Waiting and Turnaround Time
- Having less Tr/Ts than others
- SRTf and SJf are best than other

Part-B


Part B
Process
Arrival Time
Service Time
P1
0
3
P2
1
5
P3
3
2
P4
9
6
P5
10
4
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
FCFS
RR (Q=4)
SJF
TF
P1
P2
P3
P4
P5

FCFS

P	Finish	Tr	Waiting	Tr/Ts
P_1	3	3	0	1
P_2	8	7	2	1.4
P_3	10	7	5	3.5
P_4	16	7	1	1.17
P_5	20	10	6	2.5

Avg Waiting Time = 2.6
Avg Turnarond Time = 6.8
Avg Tr/Ts = 1.91
Idle CPU Time = 0

RR

P	Finish	Tr	Waiting	Tr/Ts
P_1	3	3	0	1
P_2	16	9	4	1.8
P_3	9	6	4	3.0
P_4	20	11	5	1.8
P_5	18	8	4	

Avg Waiting Time
= 0+4+4+5+4 /5
= 3.4

Avg
Avg
CPU

SJF

P
P_1
P_2
P_3
P_4
P_5

Avg
Avg
Avg
CPU

SRT

P
P_1
P_2
P_3
P_4
P_5

Avg

Avg Turnaround Time = 7.4
Avg Tr/Ts = 1.93
CPU Idle Time = 0

SJF

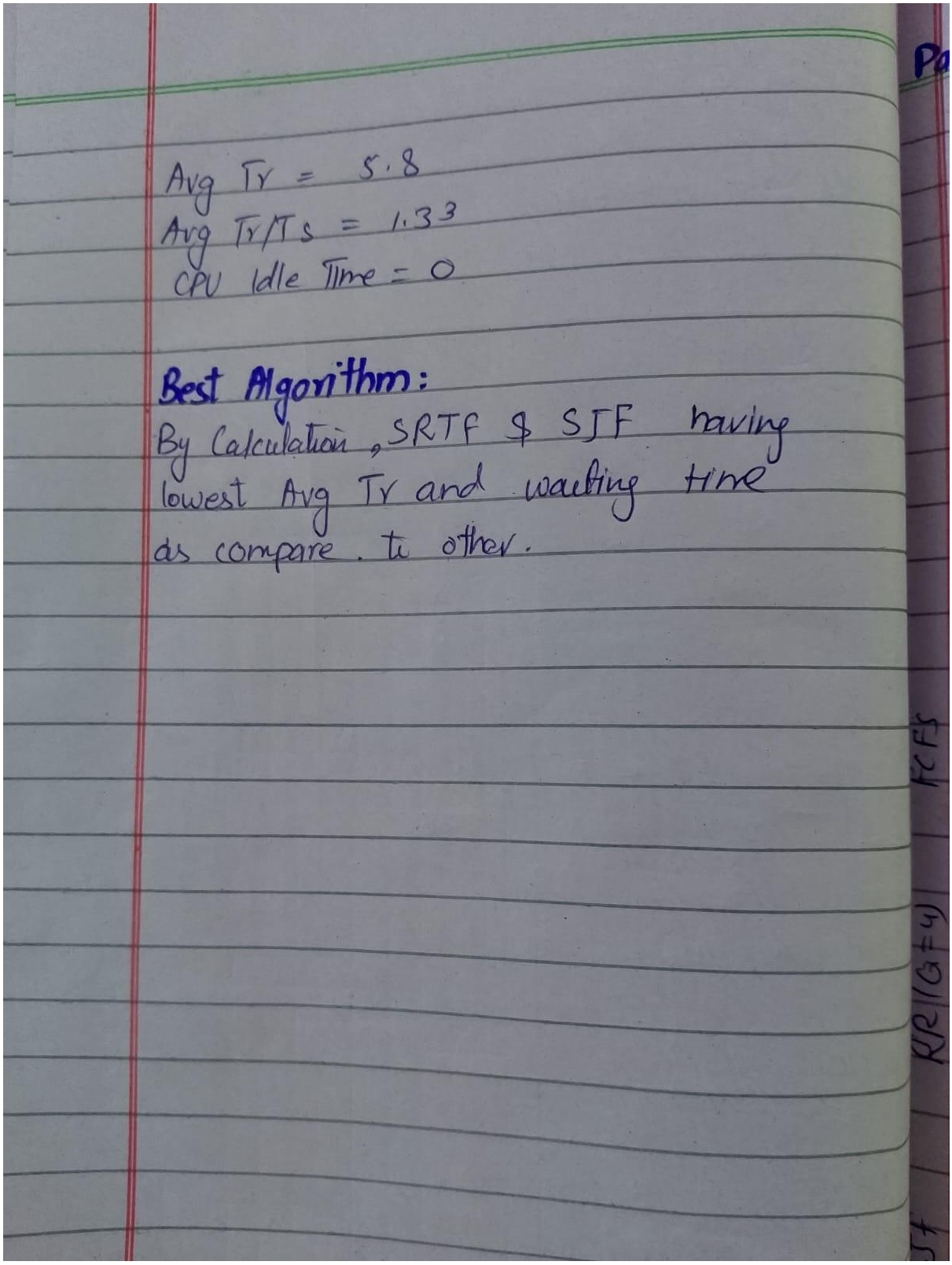
P	finish	Tr	Waiting Time	Tr/Ts
P_1	3	3	0	1
P_2	10	9	4	1.8
P_3	5	2	0	1
P_4	20	11	5	1.83
P_5	14	4	0	1

Avg Waiting time = 1.8
Avg Tr = 5.8
Avg Tr/Ts = 1.33
CPU idle time = 0

SRTF

P	Finish	Tr	Waiting Time	Tr/Ts
P_1	3	3	0	1
P_2	10	9	4	1.8
P_3	5	2	0	1
P_4	20	11	5	1.83
P_5	14	4	0	1

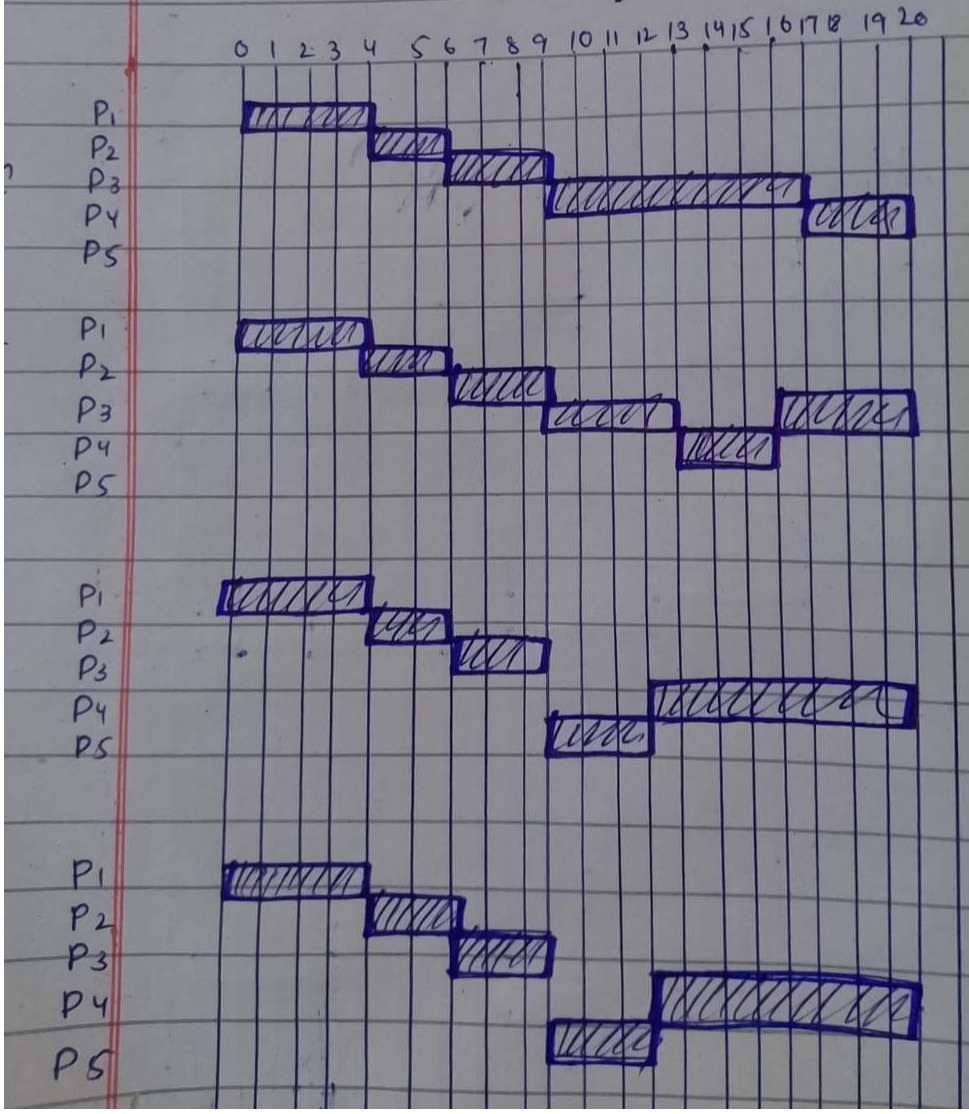
Avg Waiting Time = 1.8


Avg Tr = 5.8
Avg Tr/Ts = 1.33
CPU Idle Time = 0
Best Algorithm:
By Calculation, SRTF & SJF having lowest Avg Tr and waiting time as compare to other.

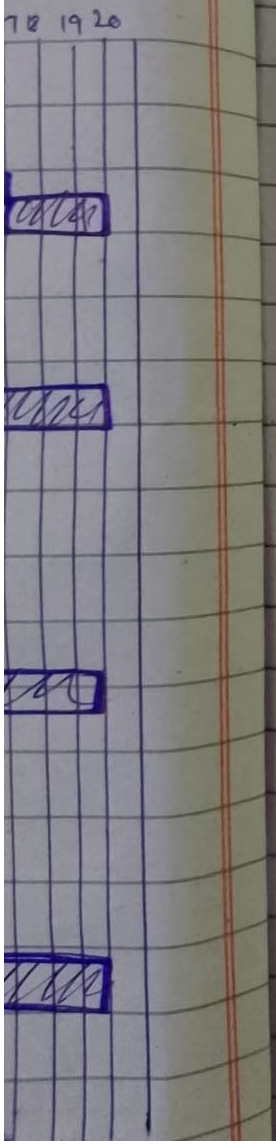
Part-C

Part-C

Process	Arrival Time	Service Time
P_1	0	4
P_2	2	2
P_3	3	3
P_4	4	8
P_5	8	3



Service Time
4
2
3
8
3



FCFS

P	Finish	Turnaround Time (F-AT)	Waiting time (Tr-Ts)	Tr/Ts
A	4	4	0	1
B	[illegible]	4	2	2
P_3	9	6	3	2
P_4	17	13	5	1.62
P_5	20	12	9	4
				1

Avg Turn Around time = 7.8
Avg Waiting time = 3.8
Avg Tr/Ts = 2.1
CPU idle = 0

RR (Q=4)

P	Turnaround Time	waitiyTme	Tr/Ts	Finish
P_1	4	0	1	4
P_2	4	2	2	6
P_3	6	3	2	9
P_4	16	8	2	20
P_5	8	5	2.6	16

Avg Trnaraund time = 7.6
Avg waeity tive = 3.6
Avg Tr/Ts = 9.6
CPU bdle = 0

SJF

Finish	P	Turnaround Time	Waiting Time	Tr/Ts
4	P_1	4	0	1
6	P_2	4	2	2
9	P_3	6	3	2
20	P_4	16	8	2
12	P_5	4	1	1.33

Avg Turnaround Time = 8.5
Avg Waiting time = 2.8
Avg Tr/Ts = 1.66
Idle CPU = 0

SRTF

Finish	P	Turnaround Time	Waiting Time	Tr/Ts
4	P_1	4	0	1
6	P_2	4	2	2
9	P_3	6	3	2
20	P_4	16	8	2
12	P_5	4	1	1.33

Avg Turnaround Time = 8.5
Avg Waiting Time = 2.8
Avg Tr/Ts = 1.66
CPU idle = 0

Best Algorithm

By analysis SJF and SRTF are optimal Algorithm because their avg turnaround Time are Avg Waiting time is shorter than others