National Textile University

Department of Computer Science

Subject:

Operating System

Submitted to:

Sir Nasir

Submitted by:

Abdul Rehman

Reg number:

1122

Assignment No:

01

Semester:

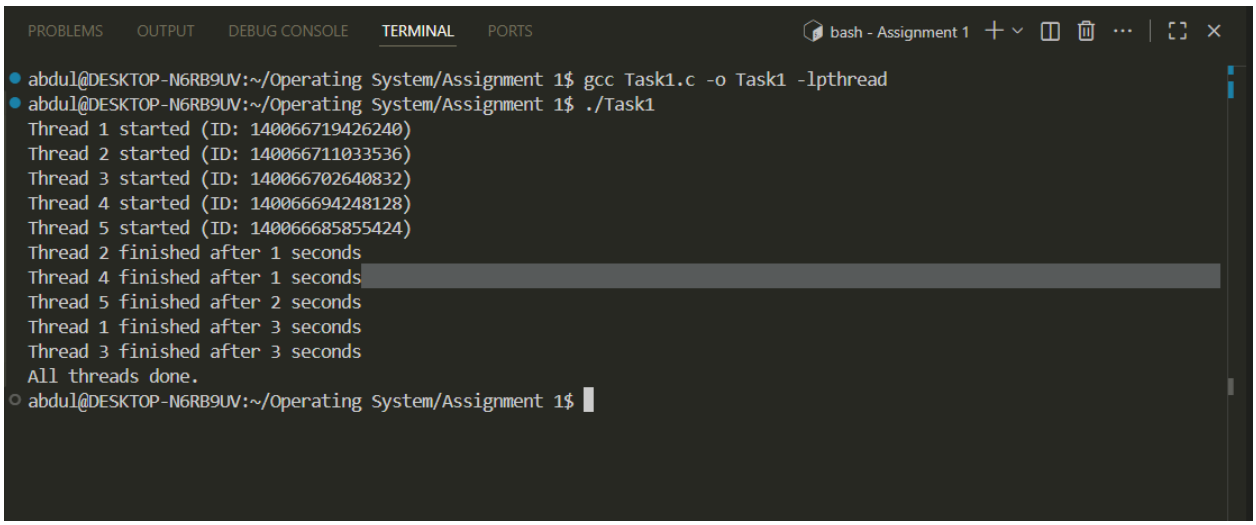
Section-A: Programming Tasks

Task 1:

Code:

```c
//Abdul Rehman
//23-ntu-cs-1122
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>

void* work(void* arg) {
    int num = *(int*)arg;               // Thread number
    printf("Thread %d started (ID: %lu)\n", num, pthread_self());

    int t = rand() % 3 + 1;             // Random sleep time (1-3 sec)
    sleep(t);

    printf("Thread %d finished after %d seconds\n", num, t);
    return NULL;
}

int main() {
    pthread_t t[5];
    int i, num[5];
    srand(time(0));                     // Seed random number

    for(i = 0; i < 5; i++) {
        num[i] = i + 1;
        pthread_create(&t[i], NULL, work, &num[i]);
    }

    for(i = 0; i < 5; i++) {
        pthread_join(t[i], NULL);
    }

    printf("All threads done.\n");
    return 0;
}

```

Output:

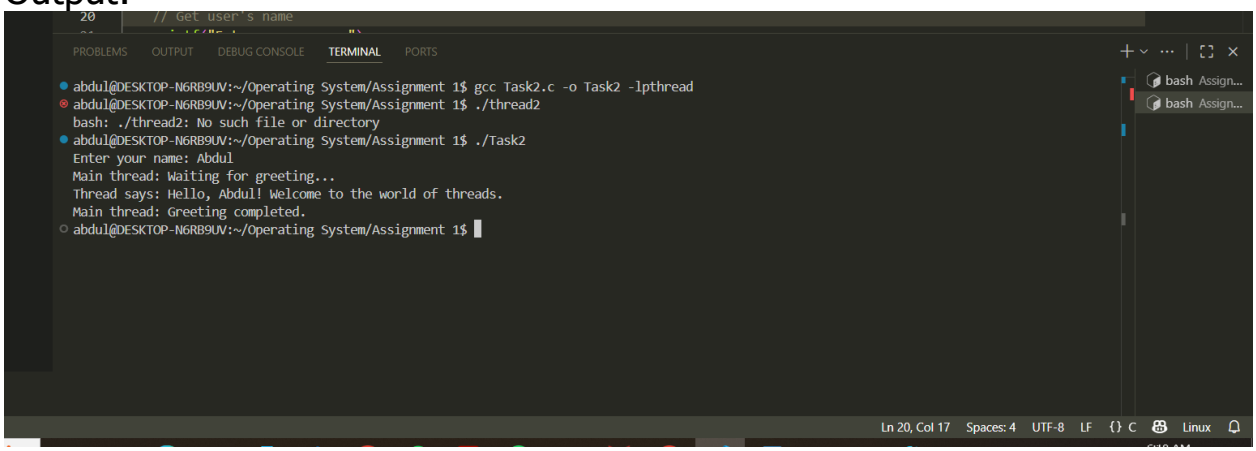

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
bash - Assignment 1
abdul@DESKTOP-N6RB9UV:~/Operating System/Assignment 1$ gcc Task1.c -o Task1 -lpthread
abdul@DESKTOP-N6RB9UV:~/Operating System/Assignment 1$ ./Task1
Thread 1 started (ID: 140066719426240)
Thread 2 started (ID: 140066711033536)
Thread 3 started (ID: 140066702640832)
Thread 4 started (ID: 140066694248128)
Thread 5 started (ID: 140066685855424)
Thread 2 finished after 1 seconds
Thread 4 finished after 1 seconds
Thread 5 finished after 2 seconds
Thread 1 finished after 3 seconds
Thread 3 finished after 3 seconds
All threads done.
abdul@DESKTOP-N6RB9UV:~/Operating System/Assignment 1$

Task 2:

Code:

```c
//Abdul Rehman
//23-ntu-cs-1122


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

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

int main() {
    pthread_t thread;
    char name[50];

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

    // Create thread
    printf("Main thread: Waiting for greeting...\n");
    if (pthread_create(&thread, NULL, greet, (void*)name) != 0) {
        perror("Failed to create thread");
        return 1;
    }

    // Wait for the created thread to complete
    pthread_join(thread, NULL);

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

```

Output:

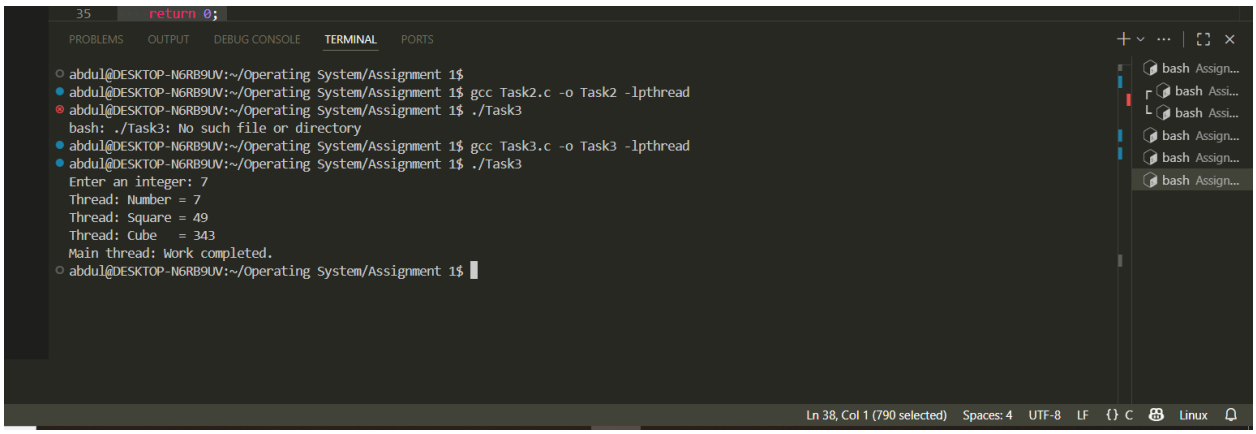


Task 3:

Code:

```c
//Abdul Rehman
//23-ntu-cs-1122
#include <stdio.h>
#include <pthread.h>

void* calculate(void* arg) {
    int num = *(int*)arg;  // Get the integer value passed to the thread

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

    pthread_exit(NULL);  // end
}

int main() {
    pthread_t thread;
    int number;

    // Input from user
    printf("Enter an integer: ");
    scanf("%d", &number);

    // Create thread
    if (pthread_create(&thread, NULL, calculate, (void*)&number) != 0) {
        perror("Failed to create thread");
        return 1;
    }

    // Wait for the thread to complete
    pthread_join(thread, NULL);

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

    return 0;
}
```

Output:



Task 4:

Code:

```c
// Abdul rehman
// 23-ntu-cs-1122
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>

void* factorial(void* arg) {
    int n = *(int*)arg;
    long long* result = malloc(sizeof(long long));  // Allocate memory to store result
    *result = 1;

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

    return result;  // Return the pointer to main thread
}

int main() {
    pthread_t thread_id;
    int number;
    long long* fact_result;

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

    pthread_create(&thread_id, NULL, factorial, &number);
    pthread_join(thread_id, (void**)&fact_result);

    printf("Factorial of %d = %lld\n", number, *fact_result);
    printf("Main thread: Work completed.\n");

    free(fact_result);  // Free allocated memory
    return 0;
}
```

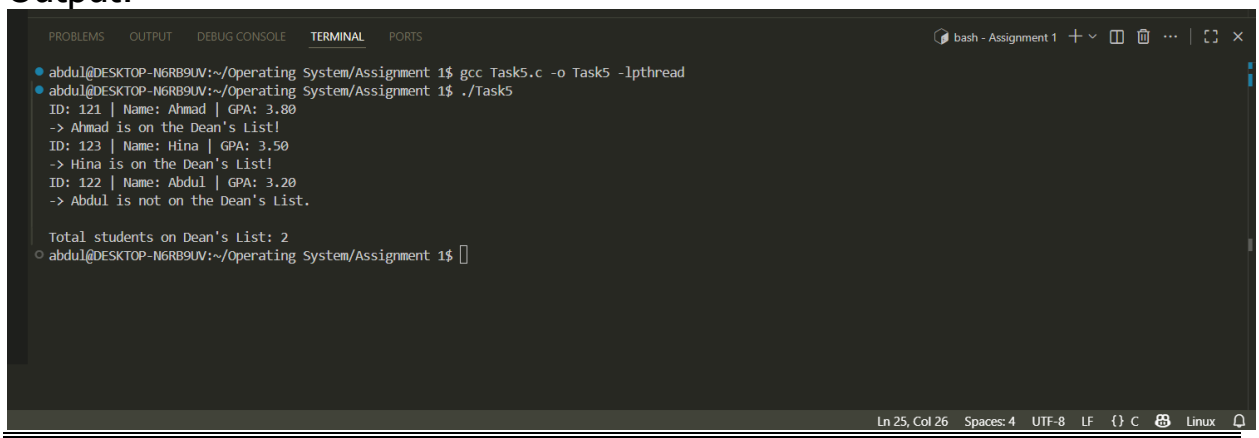
Output:

Task 5:

Code:

```c
// Abdul rehman
// 23-ntu-cs-1122
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>

void* factorial(void* arg) {
    int n = *(int*)arg;
    long long* result = malloc(sizeof(long long));  // Allocate memory to store result
    *result = 1;

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

    return result;  // Return the pointer to main thread
}

int main() {
    pthread_t thread_id;
    int number;
    long long* fact_result;

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

    pthread_create(&thread_id, NULL, factorial, &number);
    pthread_join(thread_id, (void**)&fact_result);

    printf("Factorial of %d = %lld\n", number, *fact_result);
    printf("Main thread: Work completed.\n");

    free(fact_result);  // Free allocated memory
    return 0;
}
```

Output:



Section-B: Short Questions

Q1. Define an Operating System in a single line

An Operating System is a bridge between the user and the hardware. It ensures that each

program runs smoothly and manages both computer hardware and software resources.

Q2. What is the primary function of the CPU scheduler?

The primary function of the CPU scheduler is to decide which process should run next on the CPU based on the chosen scheduling algorithm.

Q3. List any three states of a process.

The three main states of a process are:

1. **Ready:** The process is ready to run but waiting for CPU time.
2. **Running:** The process is currently executing on the CPU.
3. **Waiting:** The process is blocked and waiting for an I/O operation to complete.

Q4. What is meant by a Process Control Block (PCB)?

A PCB is a data structure in the OS that stores information about a process such as its process ID (PID), state, registers, and memory details. It helps the OS manage context switching between processes.

Q5. Differentiate between a process and a program.

A **program** is a set of instructions stored on disk, while a **process** is the execution of those instructions in memory. When a program starts executing, it becomes a process.

Q6. What do you understand by context switching?

Context switching occurs when the CPU changes from one process to another. It involves saving the state of the current process and loading the state of the next one.

Q7. Define CPU utilization and throughput.

CPU utilization measures how much time the CPU remains busy executing tasks.

Throughput refers to the number of processes completed by the CPU in a given period.

Q8. What is the turnaround time of a process?

Turnaround time is the total time taken by a process from submission to completion, including waiting, execution, and I/O time.

Q9. How is waiting time calculated in process scheduling?

Waiting time is the total time a process spends in the ready queue waiting for CPU execution.

Formula: Waiting Time = Turnaround Time – Burst Time

Q10. Define response time in CPU scheduling.

Response time is the time from when a process is submitted until it first gets CPU attention.

Q11. What is preemptive scheduling?

Preemptive scheduling allows the CPU to be taken away from a running process and given to another process with higher priority or shorter remaining time.

Q12. What is non-preemptive scheduling?

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

Q13. State any two advantages of the Round Robin scheduling algorithm.

1. Every process gets an equal share of CPU time.
2. It is suitable for time-sharing systems and provides quick responses for small tasks.

Q14. Mention one major drawback of the Shortest Job First (SJF) algorithm.

SJF may cause **starvation**, as longer processes might never get CPU time if shorter ones keep arriving.

Q15. Define CPU idle time.

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

Q16. State two common goals of CPU scheduling algorithms.

1. Maximize CPU utilization by keeping it busy as much as possible.
2. Minimize waiting and turnaround time to improve overall system performance.

Q17. List two possible reasons for process termination.

1. The process completes its execution successfully.
2. The process terminates due to an error (e.g., memory overflow or system kill).

Q18. Explain the purpose of the wait() and exit() system calls.

- **exit():** Terminates a process and returns its status to the OS.
- **wait():** Allows a parent process to pause until its child process completes execution.

Q19. Differentiate between shared memory and message-passing models of IPC.

Shared Memory	**Message Passing**
Processes communicate by sharing a common	Processes communicate by sending and

Shared Memory	Message Passing
memory space.	receiving messages.
Faster communication.	Slower due to OS/kernel mediation.
Requires synchronization by processes.	Easier to synchronize; handled by the OS.

Q20. Differentiate between a thread and a process.

Process	Thread
Independent program with its own memory space.	A smaller unit within a process sharing the same memory.
Heavyweight; creation and switching are slower.	Lightweight; faster to create and switch.
Inter-process communication is slower.	Threads communicate easily through shared memory.

Q21. Define multithreading.

Multithreading means running multiple threads within a single process to perform different tasks simultaneously, improving CPU utilization and performance.

Q22. Explain the difference between a CPU-bound process and an I/O-bound process.

A **CPU-bound process** mainly performs computations and uses the CPU heavily.

An **I/O-bound process** spends most of its time waiting for input/output operations.

Q23. What are the main responsibilities of the dispatcher?

The dispatcher gives CPU control to the process selected by the scheduler by performing context switching and restoring the process state.

Q24. Define starvation and aging in process scheduling.

Starvation: When a process never gets CPU time due to lower priority.

Aging: A technique to gradually increase the priority of waiting processes to prevent starvation.

Q25. What is a time quantum (or time slice)?

A fixed amount of CPU time allocated to each process before the CPU switches to the next

one. Used in Round Robin scheduling.

Q26. What happens when the time quantum is too large or too small?

- **Too large:** System behaves like FCFS, increasing waiting time.
- **Too small:** Too many context switches occur, wasting CPU time.

Q27. Define the turnaround ratio (TR/TS).

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

Q28. What is the purpose of a ready queue?

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

Q29. Differentiate between a CPU burst and an I/O burst.

- **CPU Burst:** The period when a process is actively using the CPU.
- **I/O Burst:** The period when a process waits for or performs input/output operations.

Q30. Which scheduling algorithm is starvation-free, and why?

Round Robin is starvation-free because every process receives an equal share of CPU time in a cyclic manner.

Q31. Outline the main steps involved in process creation in UNIX.

1. **fork():** Creates a new child process.
2. **exec():** Replaces the child's memory with a new program.
3. **wait():** Parent waits for the child to complete.
4. **exit():** Child terminates and releases its resources.

Q32. Define zombie and orphan processes.

- **Zombie Process:** A process that has finished execution but still remains in the process table.
- **Orphan Process:** A process whose parent has finished before it, leaving it without a parent process.

Q33. Differentiate between Priority Scheduling and Shortest Job First (SJF).

- **Priority Scheduling:** Selects the process with the highest priority.
- **SJF:** Selects the process with the shortest CPU burst time.

Both aim to increase system efficiency but use different selection criteria.

Q34. Define context switch time and explain why it is considered overhead.

Context switch time is the time taken to save and load process states when switching between processes. It is considered **overhead** because no useful computation occurs during this time.

Q35. List and briefly describe the three levels of schedulers in an OS.

- **Long-Term Scheduler:** Decides which processes are admitted into the ready queue.
- **Short-Term Scheduler:** Chooses which ready process runs next on the CPU.
- **Medium-Term Scheduler:** Suspends or resumes processes to control multitasking and manage memory.

Q36. Differentiate between User Mode and Kernel Mode in an OS.

- **User Mode:** Normal user programs run here with limited access to system resources.
- **Kernel Mode:** The OS runs here with full privileges to control hardware and execute critical operations.

- ## Section-C: Technical / Analytical Question:

1. Life Cycle of a Process

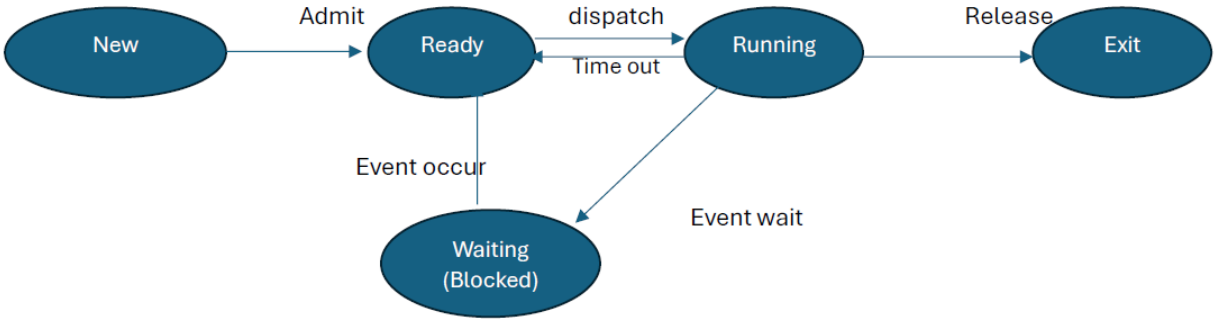
A process life cycle represents the different stages a process passes through from creation to termination.

States:

1. New: The process is being created by the OS.
2. Ready: The process is loaded in memory and waiting for CPU allocation.
3. Running: The process is currently being executed by the CPU.
4. Waiting (Blocked): The process is waiting for an I/O operation or an event to complete.
5. Terminated: The process has finished execution and is removed from the system.

State Transitions:

- New → Ready: When admitted by the OS.
- Ready → Running: When the CPU is assigned.
- Running → Waiting: When waiting for I/O.
- Waiting → Ready: When I/O completes.
- Running → Terminated: When execution finishes.



2. Context Switch Overhead

A context switch occurs when the CPU changes from one process (or thread) to another. During a switch, the OS must save the state of the current process and load the state of the next one from its PCB.

Overhead:

It is considered overhead because the CPU performs administrative work instead of executing actual instructions. Frequent switching can reduce CPU efficiency.

Information Saved and Restored:

- CPU registers and program counter
- Stack pointer and memory management information
- Process state and scheduling info

Efficient scheduling aims to minimize context switches to reduce time loss and maintain smooth performance.

3. Components of a Process Control Block (PCB)

A Process Control Block (PCB) is a data structure maintained by the OS for every process. It stores all essential information to manage and control the process.

Main Components:

1. Process Identification Data: Process ID (PID), parent process ID.
2. Process State: Indicates if the process is new, ready, running, waiting, or terminated.
3. CPU Registers & Program Counter: Used to resume execution after a context switch.
4. Memory Management Information: Base and limit registers, page tables, or segment tables.
5. Scheduling Information: Process priority, CPU burst time, and scheduling queue pointers.
6. Accounting Information: CPU usage, process time, and owner details.

The PCB ensures the OS can pause and restart processes correctly without losing data or progress.

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

Schedulers manage process selection at different levels:

Scheduler	**Function**	**Example / Description**
Long-Term Scheduler (Job	Decides which processes are admitted into the system for processing. Controls the degree of	Example: Choosing which batch jobs to load

Scheduler	Function	Example / Description
Scheduler)	multiprogramming.	into memory.
Medium-Term Scheduler (Swapper)	Temporarily removes processes from memory (swapping out) and brings them back later (swapping in) to manage memory efficiently.	Example: Suspends inactive processes to free memory.
Short-Term Scheduler (CPU Scheduler)	Selects which ready process should run next on the CPU. Works frequently and very fast.	Example: Round Robin or FCFS selection for CPU execution.

5. CPU Scheduling Criteria and Optimization Goals

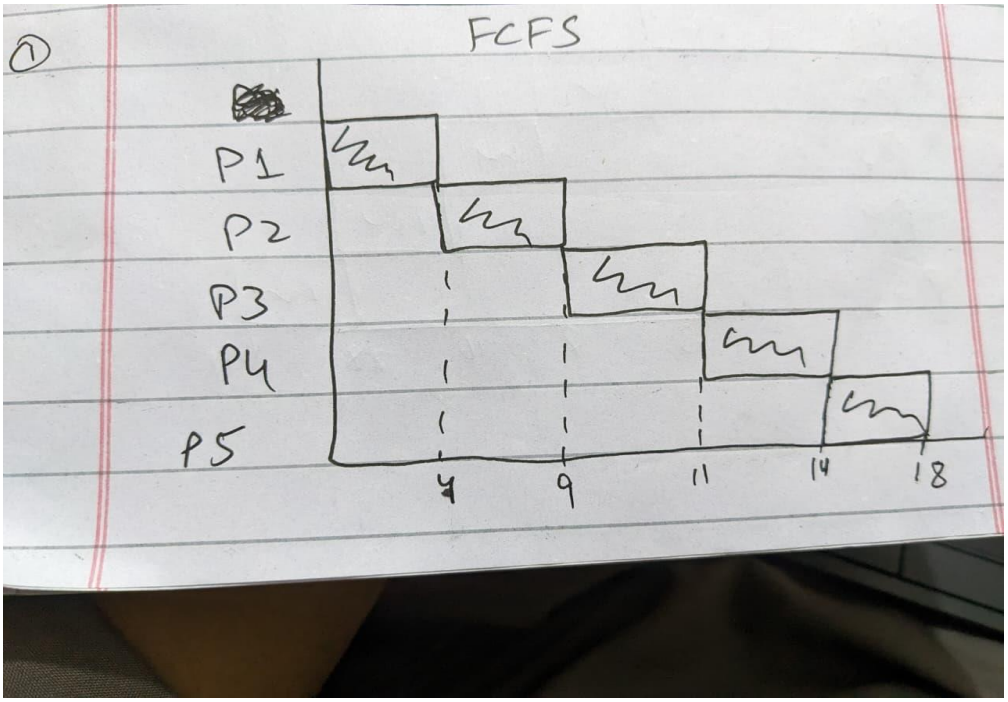
CPU scheduling is evaluated using several performance criteria:

Criterion	Meaning	Optimization Goal
CPU Utilization	The percentage of time the CPU is actively executing processes.	Maximize utilization (keep CPU busy).
Throughput	Number of processes completed per unit time.	Maximize throughput.
Turnaround Time	Total time from process submission to completion.	Minimize turnaround time.
Waiting Time	Total time a process spends waiting in the ready queue.	Minimize waiting time.
Response Time	Time from request submission to the first response shown to the user.	Minimize response time (important in interactive systems).

Part-A :

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

Gant chart:

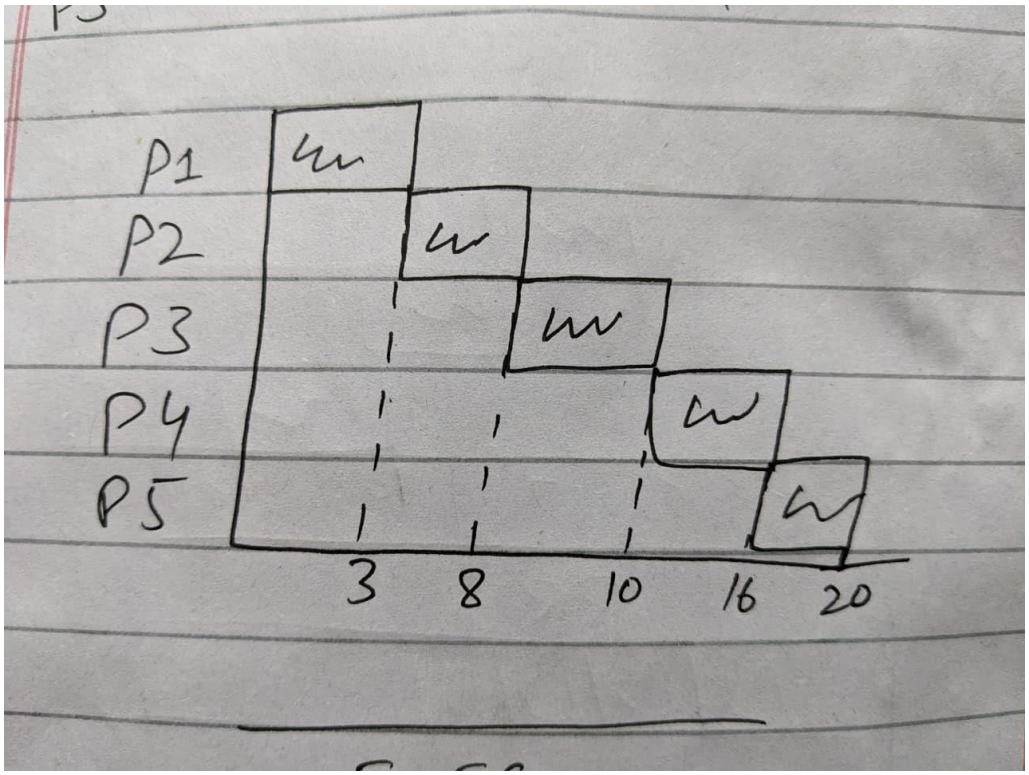


Completion, Turn Around, Waiting:
P1: Finish=4, Turn Around=4, Waiting=0
P2: Finish=9, Turn Around=7, Waiting=2
P3: Finish=11, Turn Around=7, Waiting=5
P4: Finish=14, Turn Around=8, Waiting=5
P5: Finish=18, Turn Around=9, Waiting=5

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

Round Robin:

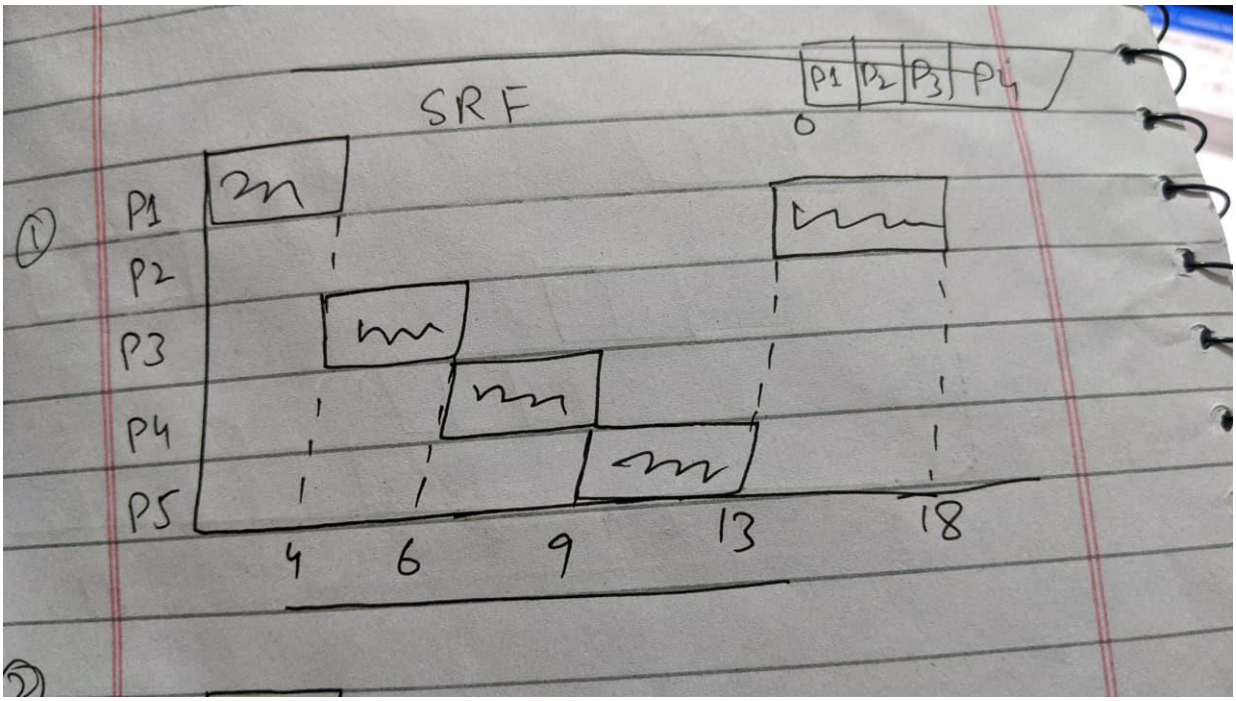
Gant Chart:


P1
P2
P3
P4
P5
3
8
10
16
20

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

Shortest Job First:

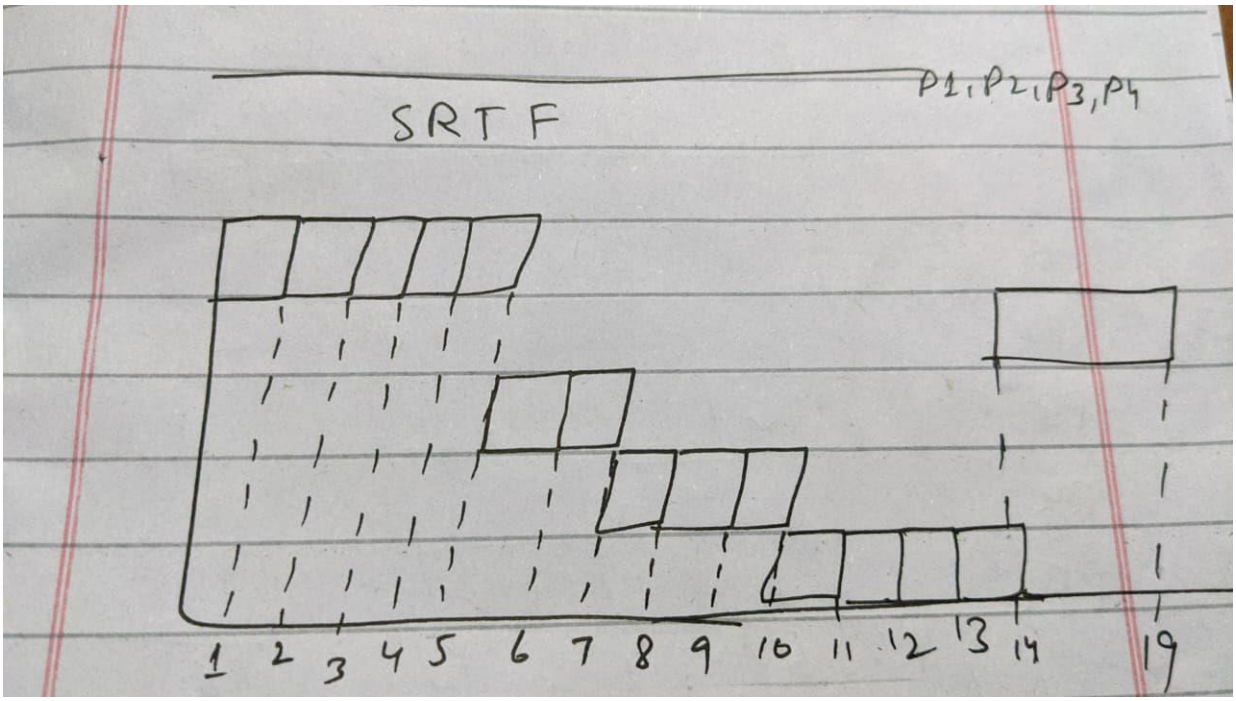


Completion, Turn Around, Waiting:
P1: Finish=4, Turn Around=4, Waiting=0
P2: Finish=18, Turn Around=16, Waiting=11
P3: Finish=6, Turn Around=2, Waiting=0
P4: Finish=9, Turn Around=3, Waiting=0
P5: Finish=13, Turn Around=4, Waiting=0

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

SRTF:

Gant chart:



Conclusion (Part-A):

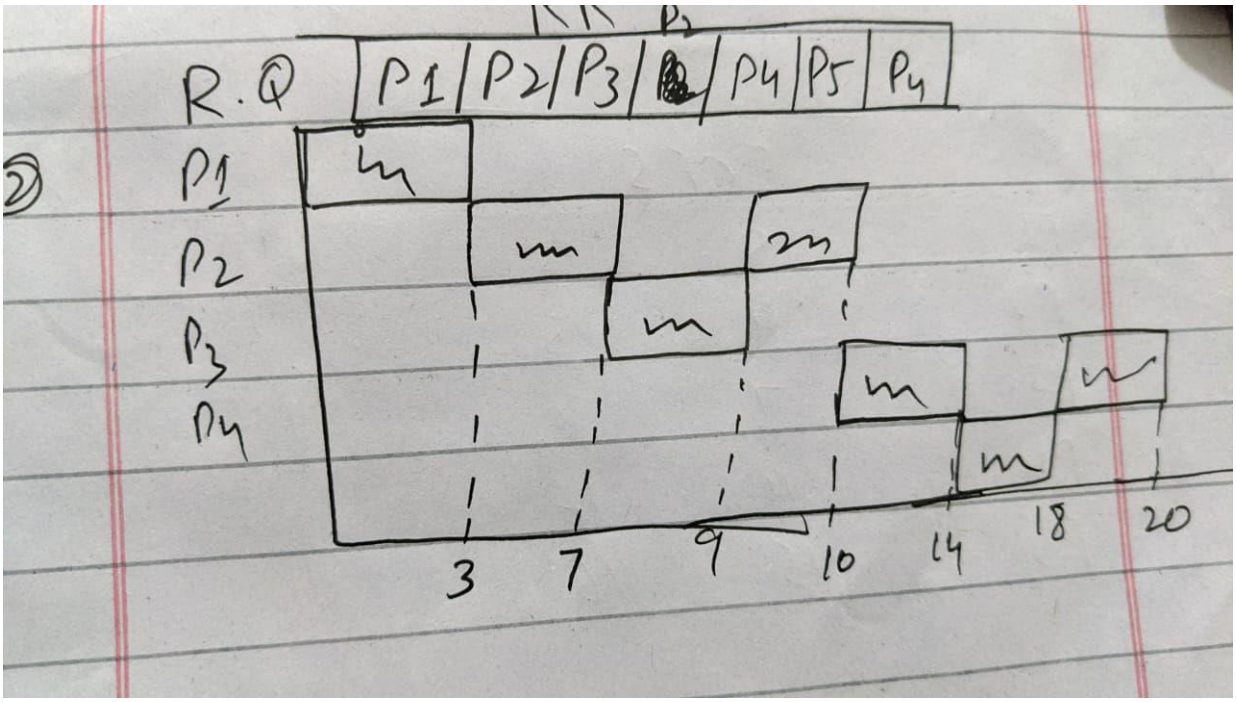
SJF/SRTF gives best average turnaround and smallest average waiting time for these arrivals. RR improves response but increases average turnaround vs SJF. FCFS is

Part B:

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

FCFS:

Gant chart:

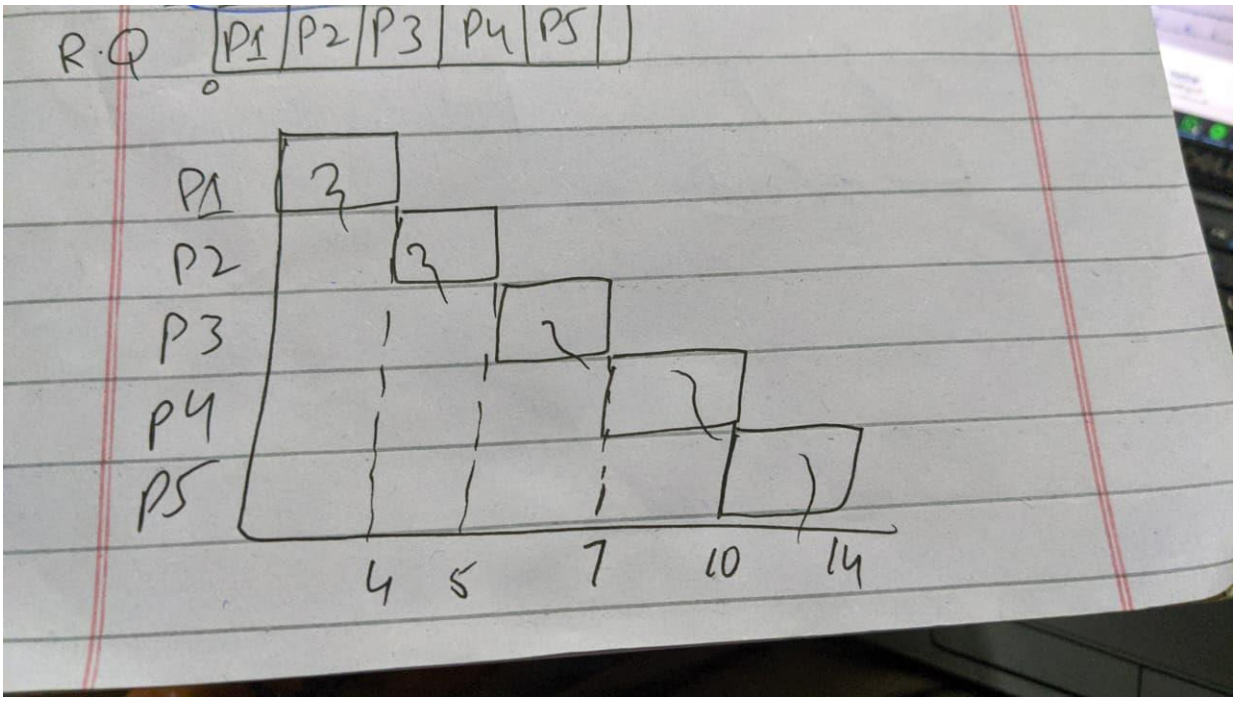


Completion, Turn Around, Waiting:
P1: Finish=3, Turn Around=3, Waiting=0
P2: Finish=8, Turn Around=7, Waiting=2
P3: Finish=10, Turn Around=7, Waiting=5
P4: Finish=16, Turn Around=7, Waiting=11
P5: Finish=20, Turn Around=10, Waiting=6

Averages:
Avg Wait Time= (0+2+5+11+6)/5 = 4.8
Avg Turn Around= (3+7+7+7+10)/5 = 6.8
Avg TR/TS = 1.91
CPU Idle = 0

Round Robin:

Gant chart:

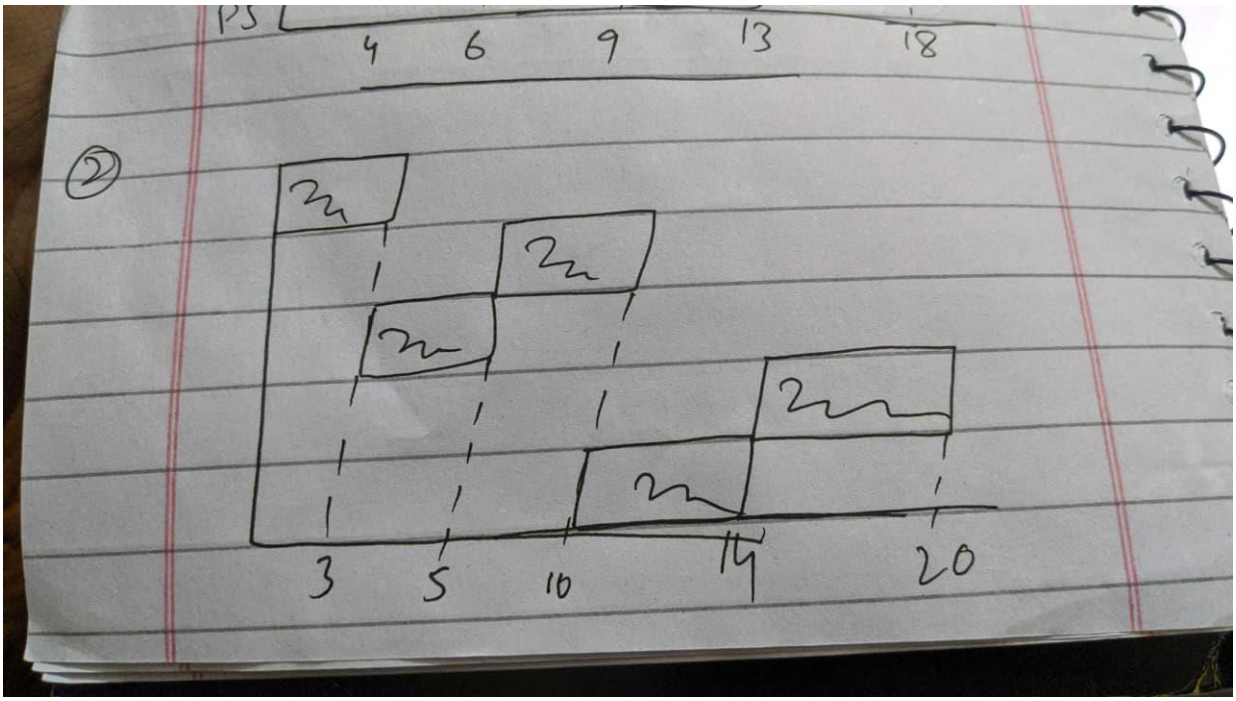


Completion, Turn Around, Waiting:
P1: Finish=3, Turn Around=3, Waiting=0
P2: Finish=14, Turn Around=13, Waiting=8
P3: Finish=9, Turn Around=6, Waiting=4
P4: Finish=16, Turn Around=7, Waiting=1
P5: Finish=20, Turn Around=10, Waiting=6

Averages:
Avg Wait Time= (0+8+4+1+6)/5 = 3.8
Avg Turn Around= (3+13+6+7+10)/5 = 7.8
Avg TR/TS = 2.05
CPU Idle = 0

Shoertest job First:

Gant chart:

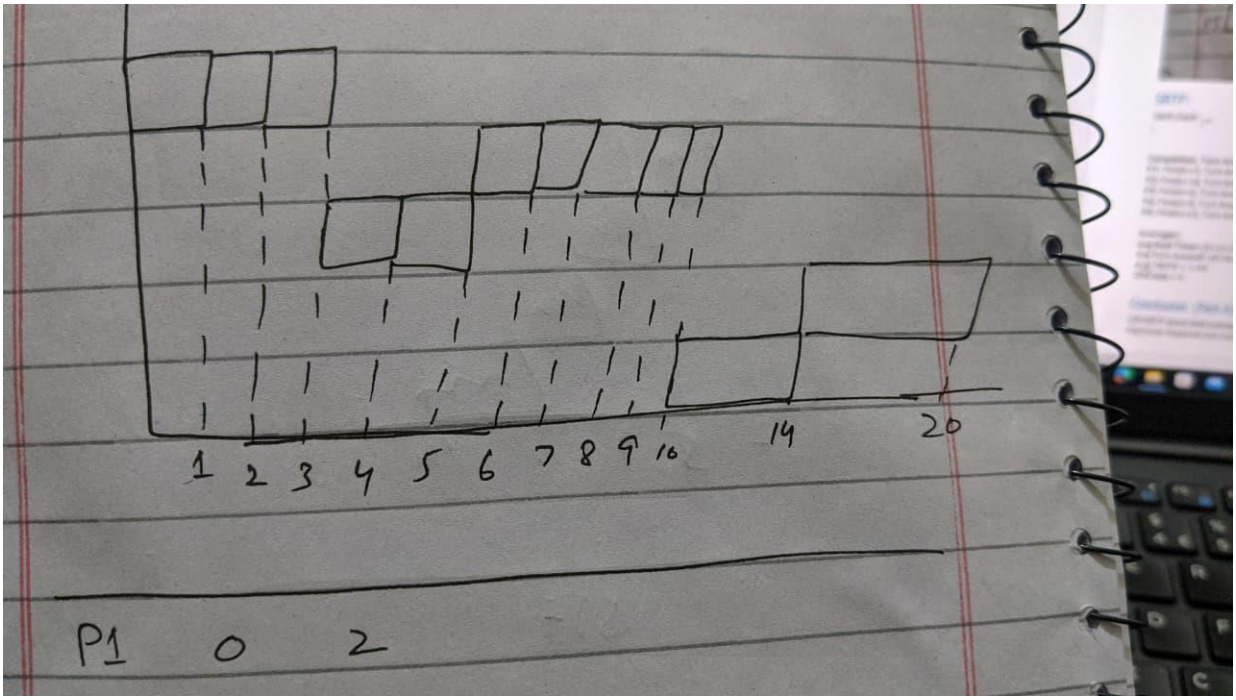


Completion, Turn Around, Waiting:
P1: Finish=3, Turn Around=3, Waiting=0
P2: Finish=10, Turn Around=9, Waiting=4
P3: Finish=5, Turn Around=2, Waiting=0
P4: Finish=20, Turn Around=11, Waiting=5
P5: Finish=14, Turn Around=4, Waiting=6

Averages:
Avg Wait Time= (0+4+0+5+0)/5 = 1.8
Avg Turn Around= (3+G+2+11+4)/5 = 5.8
Avg TR/TS = 1.33
CPU Idle = 0

SRTF:

Gant chart:



Conclusion:

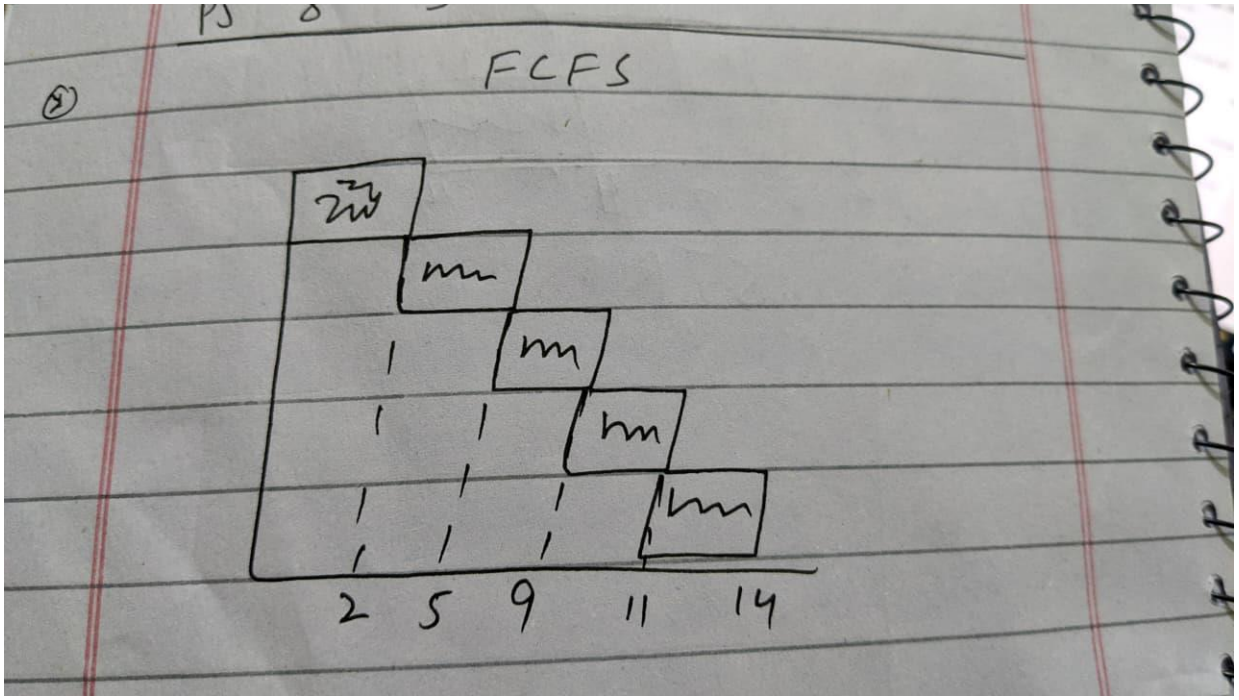
SJF gives best average waiting C turnaround; RR gives fairness but higher average turnaround than SJF.

Part C

Process	Arrival Time	Service Time (Burst)
P1	0	2
P2	1	3
P3	2	4
P4	4	2
P5	8	3

FCFS:

Gant chart:



Completion, Turn Around, Waiting:

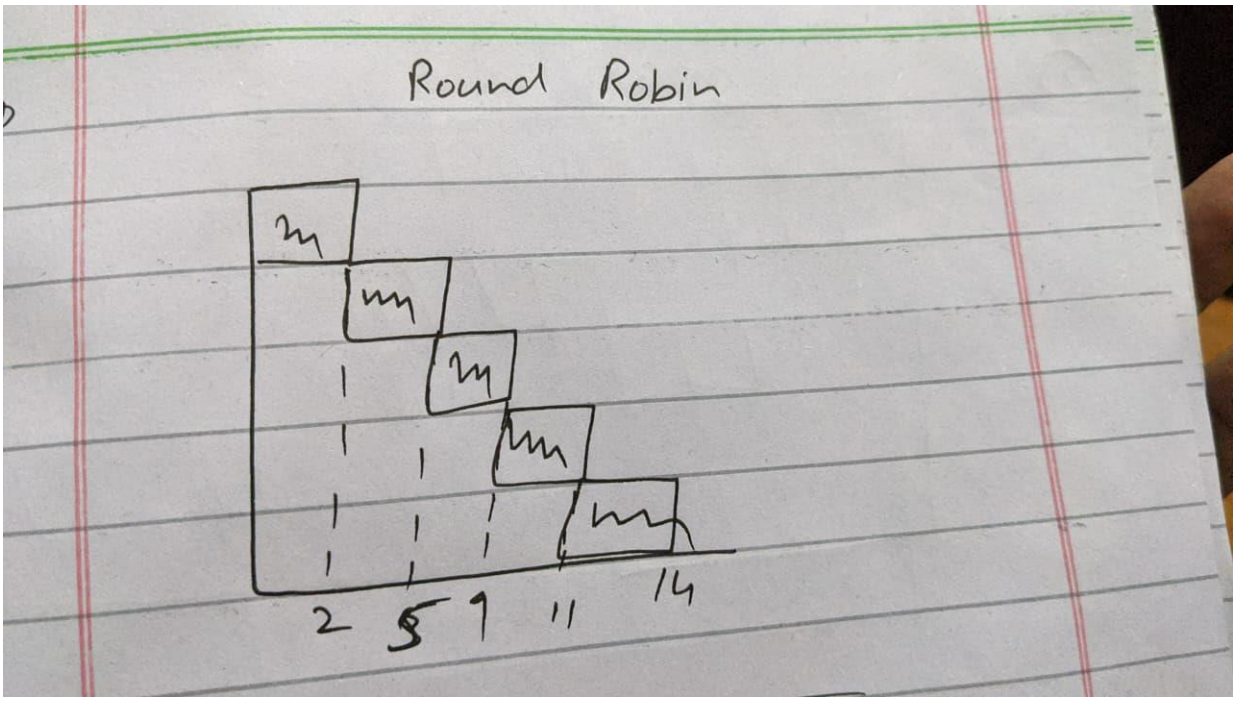
P1: Finish=2, Turn Around=2, Waiting=0
P2: Finish=5, Turn Around=4, Waiting=1
P3: Finish=9, Turn Around=7, Waiting=3
P4: Finish=11, Turn Around=7, Waiting=5
P5: Finish=14, Turn Around=6, Waiting=3

Averages:

Avg Waiting = (0+1+3+5+3)/5 = 2.4
Avg Turnaround = (2+4+7+7+6)/5 = 5.2
CPU Idle = 0

Round Robin :

Gant chart:

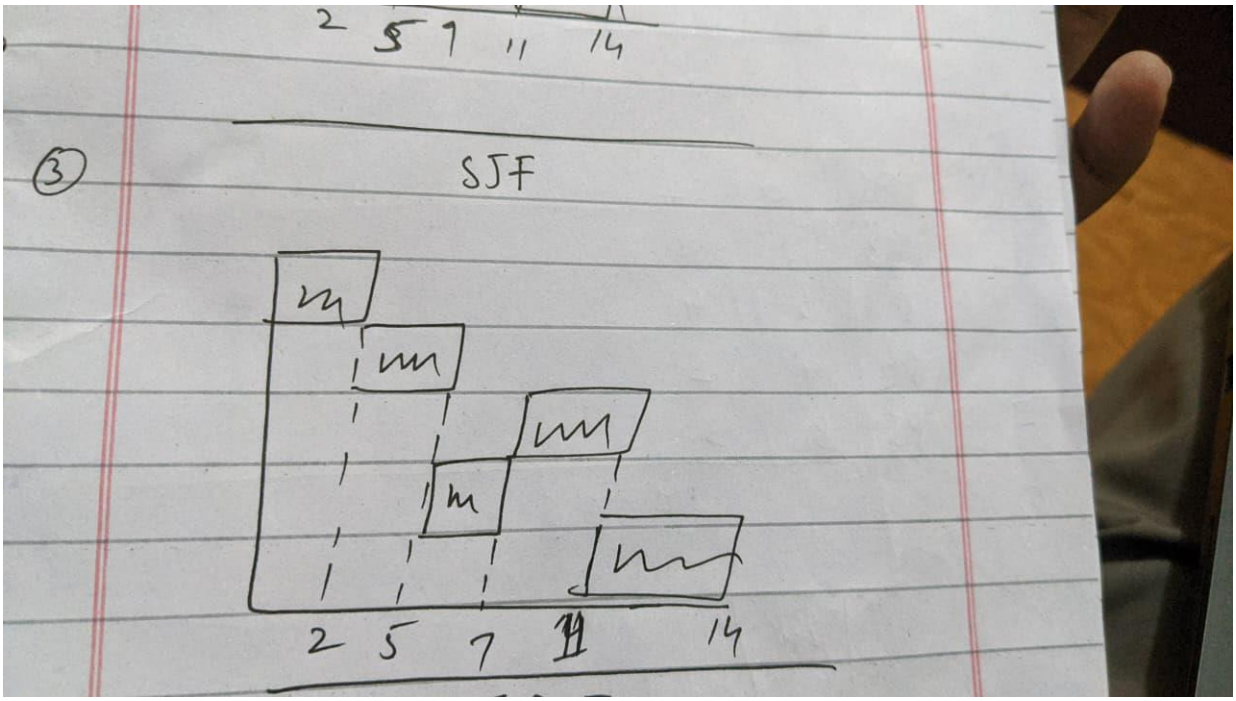


Completion, Turn Around, Waiting:
P1: Finish=2, Turn Around=2, Waiting=0
P2: Finish=5, Turn Around=4, Waiting=1
P3: Finish=9, Turn Around=7, Waiting=3
P4: Finish=11, Turn Around=7, Waiting=5
P5: Finish=14, Turn Around=6, Waiting=3

Averages:
Avg Waiting = (0+1+3+5+3)/5 =2.4
Avg Turnaround = (2+4+7+7+6)/5 =
5.2
CPU Idle = 0

SJF:

Gant chart:

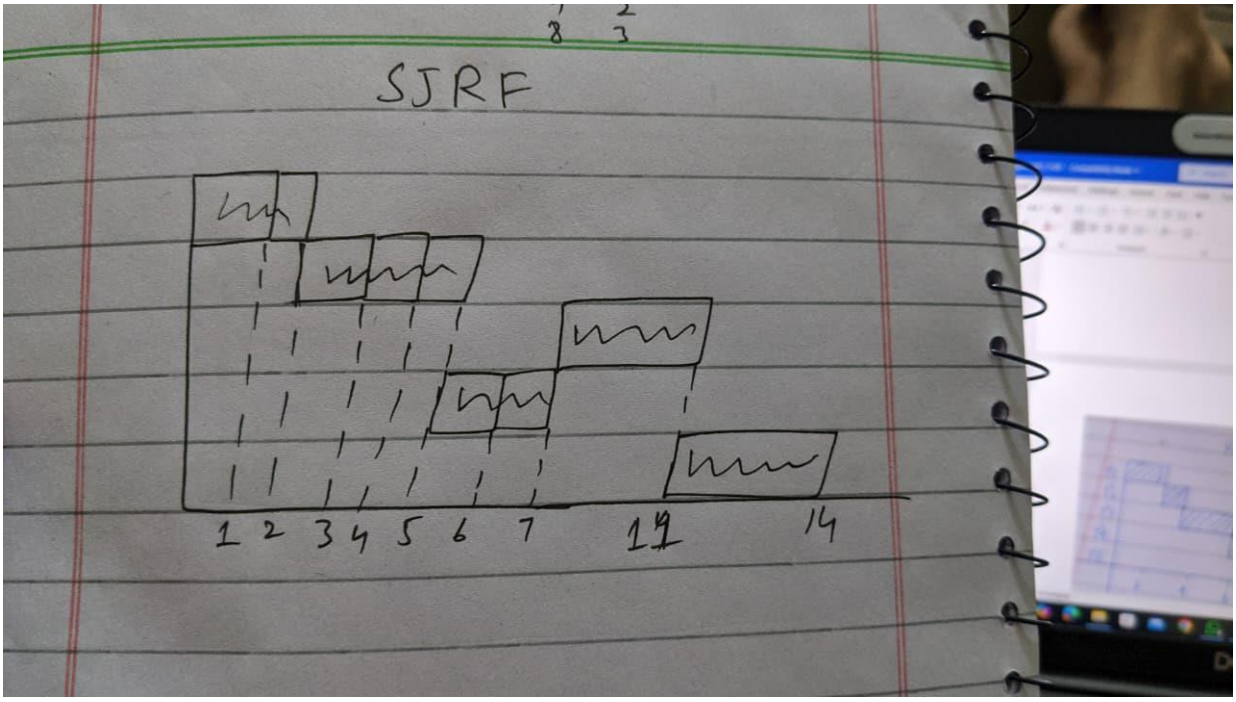


Completion, Turn Around, Waiting:
P1: Finish=2, Turn Around=2, Waiting=0
P2: Finish=5, Turn Around=4, Waiting=1
P3: Finish=7, Turn Around=3, Waiting=1
P4: Finish=11, TurnAround=9, Waiting=5
P5: Finish=14 TurnAround=6, Waiting=3

Averages:
Avg Waiting = (0+1+1+5+3)/5 = 2.0
Avg Turnaround = (2+4+3+9+6)/5 = 4.8
CPU Idle = 0

SRTF:

Gant chart:



Completion, Turn Around, Waiting:

P1: Finish=2, Turn Around=2, Waiting=0
P2: Finish=5, Turn Around=4, Waiting=1
P3: Finish=7, Turn Around=3, Waiting=1
P4: Finish=11, TurnAround=9, Waiting=5
P5: Finish=14 TurnAround=6, Waiting=3

Averages:
Avg Waiting = (0+1+1+5+3)/5 = 2.0
Avg Turnaround = (2+4+3+9+6)/5 = 4.8
CPU Idle = 0

Conclusion:

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