



Department of Computer Science

Subject: Operating System

Submitted to: Sir Nasir Mahmood

Submitted by: Eisha Muzaffar

Reg. number: 23-NTU-CS-1147

Assignment

semester:5th

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.

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#include <time.h>

#define NUM_THREADS 5

// Thread function

void* thread_function(void* arg) {

    int thread_num = *(int*)arg;

    pthread_t tid = pthread_self();
```

```

int sleep_time = (rand() % 3) + 1;

printf("Thread %d started | Thread ID: %lu | Sleeping for %d seconds\n",
      thread_num, tid, sleep_time);

sleep(sleep_time);

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

pthread_exit(NULL);
}

int main() {
    pthread_t threads[NUM_THREADS];
    int thread_args[NUM_THREADS];
    srand(time(NULL));

    printf("Creating %d Threads\n\n", NUM_THREADS);
    for (int i = 0; i < NUM_THREADS; i++) {
        thread_args[i] = i + 1;

        pthread_create(&threads[i], NULL, thread_function, &thread_args[i]);
    }

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

    printf("\n All Threads Completed\n");

    return 0;
}

```

Terminal:

```

eisha123@DESKTOP-4BG4SFL:~/OS-labs/Assignment$ gcc Task1.c -o Task1.out -lpthread
eisha123@DESKTOP-4BG4SFL:~/OS-labs/Assignment$ ./Task1.out
Creating 5 Threads

Thread 1 started | Thread ID: 139677013112512 | Sleeping for 1 seconds
Thread 2 started | Thread ID: 139677004719808 | Sleeping for 3 seconds
Thread 3 started | Thread ID: 139676996327104 | Sleeping for 3 seconds
Thread 4 started | Thread ID: 139676987934400 | Sleeping for 1 seconds
Thread 5 started | Thread ID: 139676979541696 | Sleeping for 1 seconds
Thread 1 (ID: 139677013112512) finished after 1 seconds
Thread 4 (ID: 139676987934400) finished after 1 seconds
Thread 5 (ID: 139676979541696) finished after 1 seconds
Thread 2 (ID: 139677004719808) finished after 3 seconds
Thread 3 (ID: 139676996327104) finished after 3 seconds

All Threads Completed
eisha123@DESKTOP-4BG4SFL:~/OS-labs/Assignment$

```

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.

```

/*
 * Task 2 – Personalized Greeting Thread
 * Name: Eisha Muzaffar
 * Reg No: 23-NTU-CS-1147
 */

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

// Thread function to print greeting

```

```

void* greet(void* arg) {

    char* name = (char*)arg; // Cast the argument to string

    printf("Thread says: Hello, %s! Welcome to the world of
threads.\n", name);

    return NULL;
}

int main() {

    pthread_t thread;

    char name[50];

    printf("Enter your name: ");

    scanf("%s", name); // Take user input

    printf("Main thread: Waiting for greeting...\n");

    // Create a thread and pass name as argument
    pthread_create(&thread, NULL, greet, (void*)name);

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

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

    return 0;
}

```

Terminal:

```

eisha123@DESKTOP-4BG4SFL:~/OS-labs/Assignment$ gcc Task2.c -o Task2.out -lpthread
eisha123@DESKTOP-4BG4SFL:~/OS-labs/Assignment$ ./Task2.out
Enter your name: Eisha Muzaffar
Main thread: Waiting for greeting...
Thread says: Hello, Eisha! Welcome to the world of threads.
Main thread: Greeting completed.
eisha123@DESKTOP-4BG4SFL:~/OS-labs/Assignment$ █

```

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."

```
/*  
Name: Eisha Muzaffar  
Reg. No: 23-NTU-CS-1147  
Task 3 – Number Info Thread  
*/  
  
#include <stdio.h>  
#include <pthread.h>  
  
// Function executed by the thread  
void* numberInfo(void* arg) {  
    int num = *(int*)arg; // Dereference pointer to get the number  
  
    printf("Thread: The number is %d\n", num);  
    printf("Thread: Square = %d\n", num * num);  
    printf("Thread: Cube = %d\n", num * num * num);  
  
    pthread_exit(NULL); // Exit thread cleanly  
}  
  
int main() {  
    pthread_t thread;  
    int number;  
  
    // Take user input  
    printf("Enter an integer: ");  
    scanf("%d", &number);  
  
    // Create a thread and pass number as argument
```

```

pthread_create(&thread, NULL, numberInfo, &number);

// Wait for thread to complete

pthread_join(thread, NULL);

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

return 0;
}

```

Terminal:

```

eisha123@DESKTOP-4BG4SFL:~/OS-labs/Assignment$ touch Task3.c
eisha123@DESKTOP-4BG4SFL:~/OS-labs/Assignment$ code .
eisha123@DESKTOP-4BG4SFL:~/OS-labs/Assignment$ gcc Task3.c -o Task3.out -lpthread
eisha123@DESKTOP-4BG4SFL:~/OS-labs/Assignment$ ./Task3.out
Enter an integer: 3
Thread: The number is 3
Thread: Square = 9
Thread: Cube = 27
Main thread: Work completed.

```

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.

```

/*
Name: Eisha Muzaffar
Reg. No: 23-NTU-CS-1147
Task 4 – Thread Return Values
*/

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

// Function executed by the thread
void* computeFactorial(void* arg) {
    int n = *(int*)arg; // Get the number

    unsigned long long *fact = malloc(sizeof(unsigned long long)); // allocate memory for result

    *fact = 1;

```

```

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

        *fact *= i;

    }

    pthread_exit((void*)fact); // Return pointer to result
}

int main() {

    pthread_t thread;

    int num;

    unsigned long long *result;

    // Take input

    printf("Enter a positive integer: ");

    scanf("%d", &num);

    // Create thread

    pthread_create(&thread, NULL, computeFactorial, &num);

    // Wait for thread to complete and get return value

    pthread_join(thread, (void**)&result);

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

    // Free allocated memory

    free(result);

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

    return 0;

}

```

Terminal:

```

eisha123@DESKTOP-4BG4SFL:~/OS-labs/Assignment$ gcc Task4.c -o Task4.out -lpthread
eisha123@DESKTOP-4BG4SFL:~/OS-labs/Assignment$ ./Task4.out
Enter a positive integer: 5
Factorial of 5 is: 120
Main thread: Work completed.
eisha123@DESKTOP-4BG4SFL:~/OS-labs/Assignment$

```

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 ($GPA \geq 3.5$).
- The main thread counts how many students made the Dean's List.

/*

Name: Eisha Muzaffar

Reg. No: 23-NTU-CS-1147

Task 5 – Student Database Thread Simulation

*/

#include <stdio.h>

#include <pthread.h>

// Define Student structure

typedef struct {

int student_id;

char name[50];

float gpa;

} Student;

// Shared counter for Dean's List students

int deanCount = 0;

pthread_mutex_t lock; // To protect shared variable

// Thread function

void* checkDeanList(void* arg) {

Student* s = (Student*)arg; // Cast argument to Student pointer

printf("\nStudent ID: %d\n", s->student_id);

printf("Name: %s\n", s->name);

printf("GPA: %.2f\n", s->gpa);

// Check Dean's List eligibility

```

if (s->gpa >= 3.5) {

    printf("%s is on the Deans List!\n", s->name);

    pthread_mutex_lock(&lock);

    deanCount++;

    pthread_mutex_unlock(&lock);

} else {

    printf("%s is not on the Deans List.\n", s->name);

}

return NULL;

}

```

```

int main() {

    pthread_t t1, t2, t3;

    // Initialize mutex

    pthread_mutex_init(&lock, NULL);

    // Create 3 students

    Student s1 = {101, "Amina", 3.8};

    Student s2 = {102, "maha", 3.2};

    Student s3 = {103, "Eman", 3.6};

    // Create 3 threads for 3 students

    pthread_create(&t1, NULL, checkDeanList, &s1);

    pthread_create(&t2, NULL, checkDeanList, &s2);

    pthread_create(&t3, NULL, checkDeanList, &s3);

    // Wait for all threads to finish

    pthread_join(t1, NULL);

    pthread_join(t2, NULL);

    pthread_join(t3, NULL);
}

```

```

pthread_join(t3, NULL);

// Print total Dean's List count

printf("\n-----\n");

printf("Total Students on Deans List: %d\n", deanCount);

printf("-----\n");

// Destroy mutex

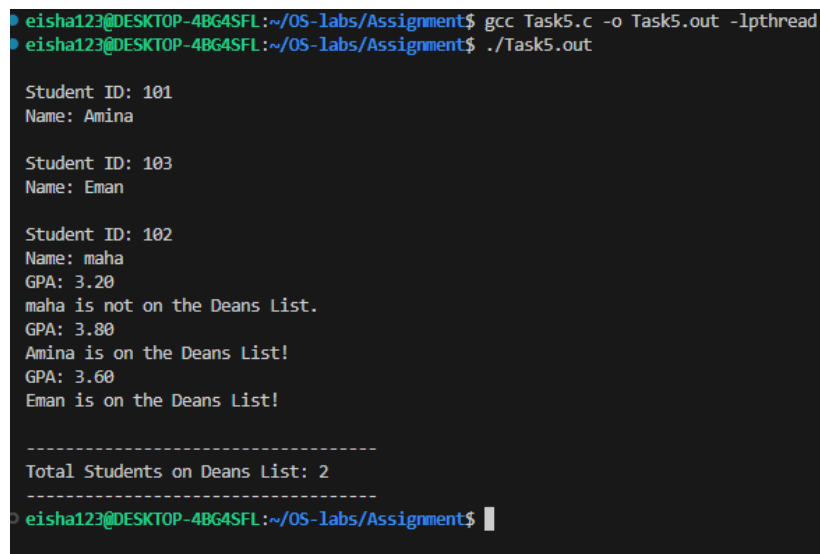
pthread_mutex_destroy(&lock);

return 0;

}

```

Terminal:



```

eisha123@DESKTOP-4BG4SFL:~/OS-labs/Assignment$ gcc Task5.c -o Task5.out -lpthread
eisha123@DESKTOP-4BG4SFL:~/OS-labs/Assignment$ ./Task5.out

Student ID: 101
Name: Amina

Student ID: 103
Name: Eman

Student ID: 102
Name: maha
GPA: 3.20
maha is not on the Deans List.
GPA: 3.80
Amina is on the Deans List!
GPA: 3.60
Eman is on the Deans List!

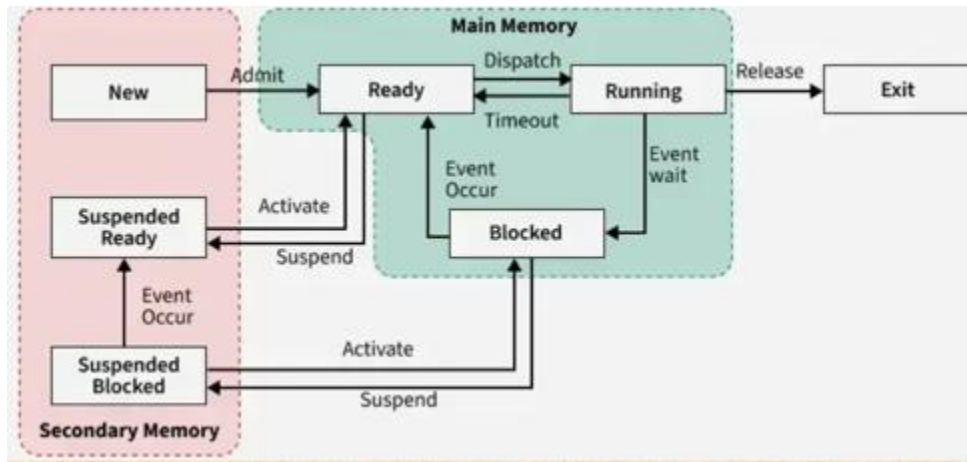
-----
Total Students on Deans List: 2
-----
eisha123@DESKTOP-4BG4SFL:~/OS-labs/Assignment$

```

Section-B: Short Questions

1. **Answer all questions briefly and clearly.**
2. **Define an Operating System in a single line.**
Acts as intermediary between user and hardware, provides environment for program execution.
3. **What is the primary function of the CPU scheduler?**
The CPU scheduler decides which process from the ready queue should be given the CPU next for execution.
4. **List any three states of a process.**
The three states of processes are:
 - **Ready:** waiting to be assigned to the CPU.
 - **Running:** currently being executed by the CPU.

- **Waiting (or Blocked):** waiting for some event like I/O to complete.



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

A PCB is a data structure that stores all information(meta data) about a process, such as its process ID, current state, program counter, and CPU registers, so that the operating system can manage and switch between processes properly

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

Aspect	Program	Process
Definition	A program is a set of instructions written to perform a specific task.	A process is a program in execution
Nature	It is passive , just stored on disk (like .exe or .c file).	It is active , has its own CPU state, memory, and resources while running.
Existence	Exists permanently on storage until deleted.	Exists temporarily in main memory while executing.
Resources	Does not require system resources.	Requires CPU time, memory, files, and I/O devices to execute.

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

Context switching is the process of saving the current state of a running process and loading the state of another process so the CPU can switch between them.

Thus,

CPU doesn't stay idle

Helps with multitasking

And every process gets a fair share of CPU time

8. **Define CPU utilization and throughput.**

CPU Utilization:

It measures how busy the CPU is; the goal is to keep the CPU as active as possible.

Throughput:

It refers to the number of processes completed in a given time.

9. What is the turnaround time of a process?

Turnaround time is the total time taken from when a process is submitted to when it is completely finished.

10. How is waiting time calculated in process scheduling?

The waiting time can be calculated as follows,

Waiting time = Turnaround time – Execution Time

It's the total time a process spends waiting in the ready queue.

11. Define response time in CPU scheduling.

Response time is the time from when a request is submitted until the first response is produced (not the final output)

12. What is preemptive scheduling?

Preemptive scheduling allows the CPU to take control from a running process and give it to another process with higher priority or shorter execution time.

Example:

Round Robin

Shortest Remaining Time First (SRTF)

13. What is non-preemptive scheduling?

In non-preemptive scheduling, once a process starts executing, it continues until it finishes or voluntarily releases the CPU.

Example:

First Come, First Served (FCFS)

Shortest Job First (SJF)

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

Two advantages of round robin scheduling Algorithm are:

- Every process gets an equal share of CPU time, so it's fair.
- It gives quick responses, making it good for time-sharing systems.

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

One major drawback of Shortest Job First is,

It may cause **starvation**, where long processes keep waiting if short ones keep arriving.

16. Define CPU idle time.

CPU idle time is the time when the CPU is not executing any process and is sitting idle.

17. State two common goals of CPU scheduling algorithms.

The Two common goals of CPU scheduling algorithm are:

- To make the CPU work efficiently (high utilization).
- To minimize waiting and turnaround time for processes.

18. List two possible reasons for process termination.

The two possible reasons for process termination are:

- The process completes its execution successfully.
- The process is killed due to an error or by another process (like the parent).

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

wait():

Used by a parent process to wait until its child process finishes.

exit():

Used by a process to end its execution and return control to the operating system.

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

Shared Memory Model	Message Passing Model
Processes communicate by sharing a common memory space .	Processes send and receive messages using system calls.
Faster , because data is exchanged directly through memory.	Slower , since messages must go through the operating system.
Processes need to coordinate (synchronize) access to avoid conflicts.	No shared data, so synchronization is handled automatically by message passing.
Used when processes are on the same system .	Used when processes are on different systems or computers .

21. Differentiate between a thread and a process.

Process	Thread
A process is an independent program in execution with its own memory and resources.	A thread is a smaller unit of a process that shares the same memory and resources with other threads.
Each process has its own separate memory space .	All threads of a process share the same memory space .
Communication between processes is slower and complex (needs IPC).	Communication between threads is faster , since they share data easily.
Processes are independent of each other.	Threads are dependent , if one crashes, it can affect the whole process.

23. Define multithreading.

Multithreading is when a single process has multiple threads running at the same time, allowing different parts of the program to execute simultaneously.

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

CPU-bound process:

Spends most of its time doing calculations and using the CPU.

I/O-bound process:

Spends most of its time waiting for input/output operations like reading or writing data.

25. What are the main responsibilities of the dispatcher?

The **dispatcher** is a part of the operating system that gives control of the CPU to the process selected by the scheduler.

Its main responsibilities are:

1. Saving the state of the old process and loading the state of the new one.
2. Switching context between processes.
3. Jumping to the correct program location to start the next process.
4. Ensuring proper CPU control and timing.

26. Define starvation and aging in process scheduling.

Starvation: When a process waits for a very long time (or never gets CPU time) because other higher-priority processes keep getting executed first.

Aging: A technique used to **prevent starvation** by **gradually increasing the priority** of a waiting process over time.

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

A **time quantum** is the **fixed amount of CPU time** given to each process in the **Round Robin** scheduling algorithm before the CPU moves to the next process.

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

If **too large**: system behaves like **FCFS (First Come, First Served)**, leading to **poor response time**.

If **too small**: **too many context switches** happen, wasting CPU time and causing overhead.

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

The **Turnaround Ratio (TR/TS)** is the ratio of **Turnaround Time (TR)** to **Service (or Burst) Time (TS)**.

It shows how efficiently a process is completed compared to the actual CPU time it needed.

30. What is the purpose of a ready queue?

The **ready queue** holds all the **processes that are ready to run** but are waiting for the CPU.

It helps the **scheduler** pick the next process to execute.

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

Aspect	CPU Burst	I/OBurst
Definition	Time when a process is executing instructions on the CPU.	Time when a process is waiting for I/O operations (like reading or writing).
Resource Used	Uses the CPU .	Uses I/O devices .
Speed	Very fast .	Slower than CPU operations.
Example	Performing calculations.	Reading a file or printing output.

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

Round Robin (RR) is starvation-free because **each process gets an equal time slice (quantum)** to run, so no process is left waiting forever.

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

The main steps in process creation in UNIX:

- The parent process calls **fork()**, which creates a **child process**.
- The child gets a **copy of the parent's data and code**.
- The child can replace its program by using **exec()** to run a new program.
- The parent can use **wait()** to wait for the child to finish.
- After completion, the child calls **exit()** to end execution

34. Define zombie and orphan processes.

Zombie Process:

A process that has finished execution but **still has an entry in the process table** because its parent hasn't read its exit status.

Orphan Process:

A process whose **parent has terminated** while it's still running. The **init process (PID 1)** later adopts it.

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

Aspect	Priority Scheduling	Shortest Job First (SJF)
Basis of Selection	Based on priority value (higher priority first).	Based on shortest CPU burst time

Starvation	Can cause starvation for low-priority processes.	Can cause starvation for long processes.
Preemption	Can be preemptive or non-preemptive .	Can also be preemptive (SRTF) or non-preemptive .
Use Case	Used when some tasks are more important.	Used when average waiting time must be minimized

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

Context switch time is the time the CPU takes to **save the current process state** and **load another process's state** during a context switch.

It's considered **overhead** because **no actual process work is done** during this time, the CPU only manages switching.

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

Long-Term Scheduler (Job Scheduler):

Decides which processes are admitted to the system for execution

Short-Term Scheduler (CPU Scheduler):

Chooses which ready process should run next on the CPU

Medium-Term Scheduling:

Temporarily removes processes from memory (suspends) and later resumes them to control multiprogramming.

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

Aspect	User Mode	Kernel Mode
Access Level	Has limited access to system resources	Has full access to all hardware and system resources.
Purpose	Used for running user applications .	Used by the operating system to perform critical tasks.
Instructions Allowed	Used for running user applications .	All instructions , including privileged ones, are allowed.
Example	Running MS Word or a web browser.	Executing system calls or device management.

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

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

Definition:

A **process life cycle** shows how a process changes its state from creation to termination.

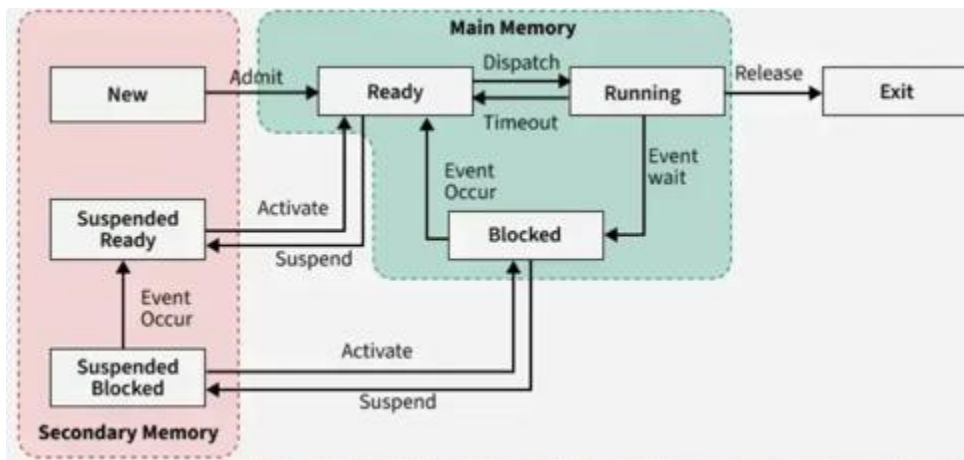
States of a Process:

1. **New:** The process is being created.
2. **Ready:** The process is loaded into memory and waiting for CPU time.
3. **Running:** The process is currently being executed by the CPU.
4. **Waiting (Blocked):** The process is waiting for some event (like I/O) to complete.
5. **Terminated:** The process has finished execution and is removed from memory.

State Transitions:

- **New → Ready:** After creation, the process is put into the ready queue.
- **Ready → Running:** The CPU scheduler selects it for execution.
- **Running → Waiting:** The process needs I/O or waits for an event.
- **Waiting → Ready:** The event finishes, and the process becomes ready again.
- **Running → Terminated:** The process finishes execution.

Diagram:



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

A **context switch** happens when the CPU changes from one process to another. During this switch, the system must **save the current process's state** and **load the next process's state**.

Context Switch Overhead:

- It's called overhead because **no useful work** is done during this time — the CPU is only busy switching.
- Too many context switches reduce overall system performance.

Information Saved and Restored:

1. **Program Counter (PC)** – to know where to resume execution.
2. **CPU Registers** – to restore the process's working data.
3. **Process State** – ready, waiting, running, etc.
4. **Memory Management Info** – pointers, page tables, etc.
5. **I/O Status** – details about open files or ongoing I/O.

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

A **Process Control Block (PCB)** is a data structure that stores **all information about a process**, so the operating system can manage it.

Main Components:

1. **Process ID (PID):**

Unique number to identify each process.

2. **Process State:**

Shows if the process is new, ready, running, waiting, or terminated.

3. **Program Counter:**

Holds the address of the next instruction to execute.

4. **CPU Registers:**

Store current working values used by the CPU.

5. **Memory Management Information:**

Includes base and limit registers, page tables, etc.

6. **Accounting Information:**

Records CPU usage, time limits, and process priority.

7. **I/O Information:**

Keeps track of I/O devices, files, and resources used.

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

Type of Scheduler	Function	Frequency	Example
Long-Term Scheduler	Decides which processes are admitted into the system for	Runs less frequently .	Admitting a new program

(Job Scheduler)	execution. Controls the degree of multiprogramming.		from disk into memory.
Medium-Term Scheduler	Temporarily removes (suspends) processes from memory to reduce load, and later resumes them.	Runs occasionally .	Swapping processes between RAM and disk.
Short-Term Scheduler (CPU Scheduler)	Chooses which process from the ready queue should run next on the CPU.	Runs very frequently (every few milliseconds).	Round Robin or Priority scheduling.

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

Criterion	Meaning	Optimization Goal
CPU Utilization	How busy the CPU is (percentage of time it is active).	Maximize CPU utilization (keep it busy).
Throughput	Number of processes completed in a given time.	Maximize throughput (finish more work).
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 first response shown.	Minimize response time (for interactive systems).

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-B

Process	Arrival Time	Service Time
P1	0	3
P2	1	5

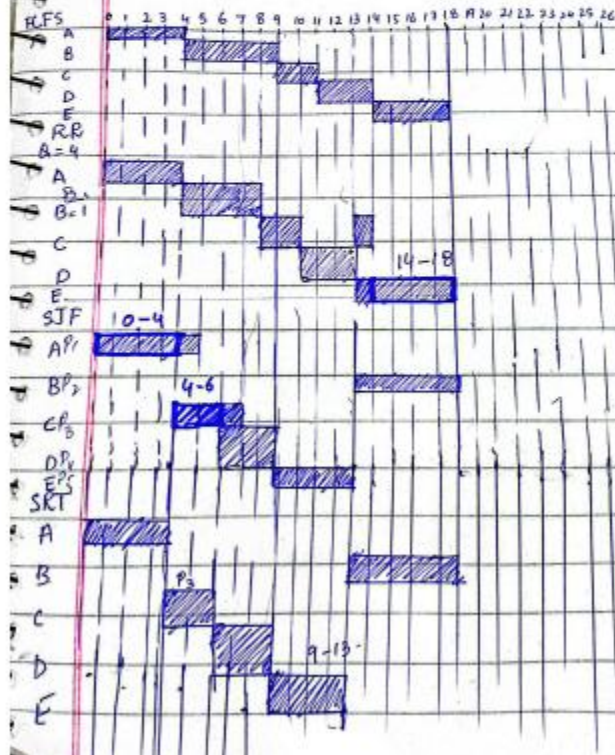
P3	3	2
P4	9	6
P5	10	4

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

Process	Arrival Time	Service Time
P1	-	-
P2	-	-
P3	-	-
P4	-	-
P5	-	-

Section D: CPU scheduling Calculations Part A:

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



FCFS:

CPU idle time: 0

Avg. waiting time: 3.40

Avg. turn around time: 7.00

Ratio,

$P_1: w=0, T=4, TR/TS=1.00$

$P_2: w=2, T=7, TR/TS=1.40$

$P_3: w=5, T=7, TR/TS=3.50$

$P_4: w=5, T=8, TR/TS=2.67$

$P_5: w=5, T=9, TR/TS=2.25$

RR:

CPU idle time: 0

Avg. waiting time: 3.00

Avg. turn around time: 7.60

SJF:

Avg. waiting time: 2.20

Avg. turn around time: 5.80

SRTF:

Avg. waiting time: 2.20

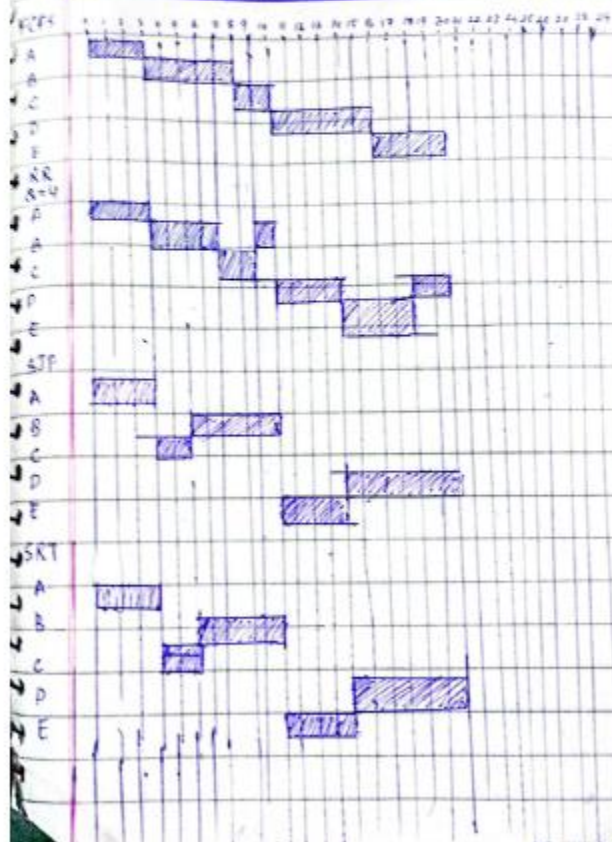
Avg. turn around: 5.80

Best,

SJF / SRTF (lowest average waiting = 2.20 and lowest avg turn around = 5.80), shortest jobs are scheduled earlier, reducing waiting turn around time.

Part B :-

Process	Arrival Time	Service time
P ₁	0	3
P ₂	1	5
P ₃	3	2
P ₄	9	6
P ₅	10	4



Part B

FCFS:

Avg waiting: 2.80

Avg turnaround: 6.80

Idle CPU time: 0

RR:

Idle: 0

Avg waiting: 2.20

Avg turnaround: 7.40

SJF:

Avg waiting: 1.80

Avg turnaround: 5.80

SRTF:

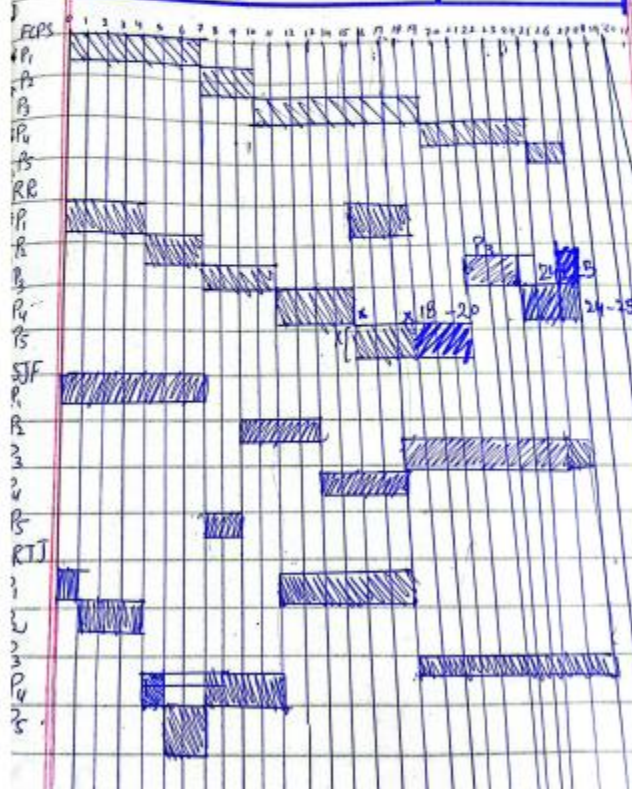
Avg waiting: 1.80

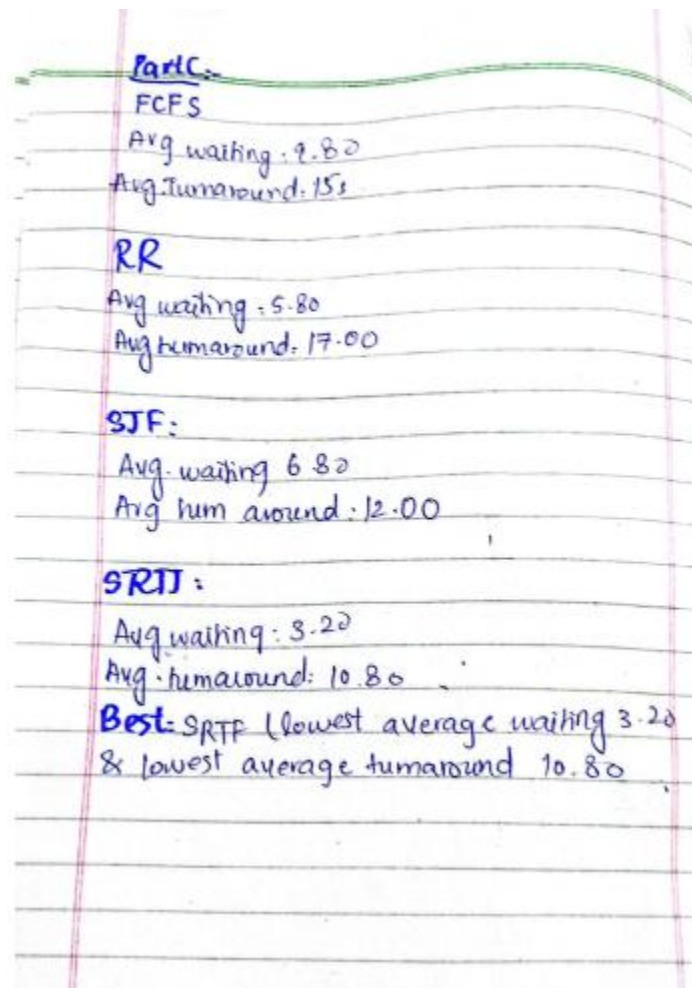
Avg turnaround: 5.80

Best: SJF/SRTF (avg waiting = 1.80, avg turnaround = 5.80)

Part C:-

Process	Arrival time	Service time
P1	0	7
P2	1	3
P3	2	9
P4	3	5
P5	5	2





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.**