Department of Computer Technology B. Tech in Computer Science and Engineering (IOT)

Vision of the Department

To be a well-known centre for pursuing computer education through innovative pedagogy, value-based education and industry collaboration.

Mission of the Department

To establish learning ambience for ushering in computer engineering professionals in core and multidisciplinary area by developing Problem-solving skills through emerging technologies.

Session 2025-2026

Vision To develop competent professionals with strong technical skills and ethical values in the field of Computer Science and Engineering	**Mission:** students with strong computing knowledge and problem-solving skills while fostering innovation, research, and practical learning. We aim to develop teamwork, leadership, and a mindset for lifelong learning, all grounded in social responsibility and professional ethics.

Program Educational Objectives of the program (PEO): (broad statements that describe the professional and career accomplishments)

PEO1	**Preparation**	**P: Preparation**	**Pep-CL abbreviation pronounce as Pep-si-lL easy to recall**
PEO2	**Core Competence**	**E: Environment (Learning Environment)**	
PEO3	**Breadth**	**P: Professionalism**	
PEO4	**Professionalism**	**C: Core Competence**	
PEO5	**Learning Environment**	**L: Breadth (Learning in diverse areas)**	

Program Outcomes (PO): (statements that describe what a student should be able to do and know by the end of a program)

Keywords of POs:
Engineering knowledge, Problem analysis, Design/development of solutions, Conduct Investigations of Complex Problems, Engineering Tool Usage, The Engineer and The World, Ethics, Individual and Collaborative Team work, Communication, Project Management and Finance, Life-Long Learning

PSO Keywords: Cutting edge technologies, Research

"I am an engineer, and I know how to apply engineering knowledge to investigate, analyse and design solutions to complex problems using tools for entire world following all ethics in a collaborative way with proper management skills throughout my life." *to contribute to the development of cutting-edge technologies and Research*.

Integrity: I will adhere to the Laboratory Code of Conduct and ethics in its entirety.

Name and Signature of Student and Date
(Signature and Date in Handwritten)

Session	**2025-26 (ODD)**	**Course Name**	**Operating System**
Semester	**5**	**Course Code**	**23IOT1504**
Roll No	**42**	**Name of Student**	**Karan F. Chopkar**

Practical Number	**3**
Course Outcome	• CO1: Use Linux toolchain and commands to compile, run, and observe thread behavior and system resource usage during the simulation, showing efficient use of the environment for OS experiments. • CO2: Analyze operating system functionalities by implementing threads, coordinating them with appropriate synchronization, and reasoning about scheduling/ordering of events in the order–prepare pipeline. • CO3: Apply synchronization primitives (e.g., mutexes/semaphores/condition variables) to ensure no deadlock or race conditions while both threads terminate cleanly after handling 5 orders. .
Aim	1. Library 2. Wait and exit System Calls 3. CPU Scheduling
Problem Definition	1. Several student threads try to borrow books from a small library (limited copies). A student must wait if the book is not available and borrow it once returned. 2. Write a program that creates a child process using fork(). The child process should print a message and terminate using the exit() system call. The parent process should wait for the child to finish using the wait() system call and then print a message. 3. CPU Scheduling
Theory (100 words)	**1. Library (Process Synchronization Problem)** The library problem represents a classical synchronization scenario in operating systems. Several students (threads) try to borrow books from a library with

	limited copies. If a book is available, a student can borrow it; otherwise, the student must wait until another student returns it. This models the producer-consumer or semaphore problem, where resources (books) are finite and access must be synchronized to prevent race conditions. Semaphores or monitors are generally used to ensure mutual exclusion and avoid conflicts. This problem helps us understand how operating systems handle limited resources among multiple processes. **2. Wait and Exit System Calls** In UNIX/Linux systems, process creation and termination are managed using system calls. The fork() system call creates a new child process. The child executes separately and can terminate using the exit() system call, which returns a status to the parent. The parent process can wait for the child's termination using the wait() system call. This prevents zombie processes and ensures proper synchronization. In practice, the child might perform a task (e.g., printing a message), and after its completion, the parent acknowledges its termination. This demonstrates parent-child relationships in process management. **3. CPU Scheduling** CPU scheduling is a fundamental concept in operating systems where the CPU decides which process to execute when multiple processes are ready. The main objective is to maximize CPU utilization, throughput, and fairness while minimizing waiting and response time. Popular scheduling algorithms include **FCFS (First-Come, First-Served)**, **SJF (Shortest Job First)**, **Priority Scheduling**, and **Round Robin**. Each has its advantages and drawbacks depending on workload type. For example, Round Robin ensures fairness in time-sharing systems, while SJF reduces average waiting time. CPU scheduling directly affects system performance and user experience.
Procedure and Execution (100 Words)	**1. Library (Synchronization)** **Procedure:** • Initialize semaphores for book availability. • Create multiple student threads. • Each student checks if a book is available. • If available → borrow, else → wait. • After reading, return the book and signal availability. **Execution:** Run the program → multiple students request books. If copies are limited, extra students wait until a book is returned, then proceed. **2. Wait and Exit System Calls**

Procedure:

- Use fork() to create a child process.
- Child prints a message and calls exit().
- Parent uses wait() to wait for child.
- Parent prints completion message.

Execution:

On running, child message appears first, then parent waits and prints after child exits.

3. CPU Scheduling

Procedure:

- Input process IDs, burst times, and priorities.
- Apply scheduling algorithm (FCFS, SJF, RR, or Priority).
- Calculate waiting and turnaround time.
- Display results in tabular form.

Execution:

Run program → processes scheduled as per algorithm. Output shows order, waiting time, and turnaround time.

Code:

1. **Threads :**

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define STUDENTS 5
#define BOOKS 2   // number of available copies

sem_t books;

void* student(void* id) {
   int sid = *(int*)id;
   printf("Student %d wants a book\n", sid);

   sem_wait(&books); // wait if no book
   printf("Student %d borrowed a book\n", sid);
   sleep(2); // reading time

   printf("Student %d returned a book\n", sid);
   sem_post(&books); // return book
   return NULL;
}
```

```
int main() {
    pthread_t tid[STUDENTS];
    int ids[STUDENTS];

    sem_init(&books, 0, BOOKS);

    for(int i=0;i<STUDENTS;i++) {
        ids[i] = i+1;
        pthread_create(&tid[i], NULL, student, &ids[i]);
    }

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

    sem_destroy(&books);
    return 0;
}
```

2. Fork :-

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid = fork();

    if(pid < 0) {
        printf("Fork failed!\n");
        return 1;
    }
    else if(pid == 0) {
        // Child process
        printf("Child process: Hello, I am the child!\n");
        exit(0);
    }
    else {
        // Parent process
        wait(NULL); // wait for child to finish
        printf("Parent process: Child finished, now parent exits.\n");
    }
    return 0;
}
```

3. CPU Sheduling :-

```
#include <stdio.h>

int main() {
    int n, i, qt = 2, count = 0, y, wt = 0, tat = 0;
    int bt[10], temp[10], at[10], tat_arr[10], wt_arr[10], completed[10];
    float avg_wt, avg_tat;

    printf("Enter total number of processes: ");
    scanf("%d", &n);
    y = n;

    for(i=0;i<n;i++) {
        printf("Enter Arrival Time for process %d: ", i+1);
        scanf("%d", &at[i]);
        printf("Enter Burst Time for process %d: ", i+1);
        scanf("%d", &bt[i]);
        temp[i] = bt[i];
        completed[i] = 0; // mark not completed
    }

    printf("\nProcess\tAT\tBT\tTAT\tWT\n");

    int t = 0;
    while(y > 0) {
        int executed = 0;
        for(i=0;i<n;i++) {
            if(at[i] <= t && temp[i] > 0) { // process has arrived
                executed = 1;
                if(temp[i] <= qt) {
                    t += temp[i];
                    temp[i] = 0;
                    tat_arr[i] = t - at[i];
                    wt_arr[i] = tat_arr[i] - bt[i];
                    printf("P%d\t%d\t%d\t%d\t%d\n", i+1, at[i], bt[i], tat_arr[i],
wt_arr[i]);
                    wt += wt_arr[i];
                tat += tat_arr[i];
                completed[i] = 1;
                y--;
              } else {
                temp[i] -= qt;
                t += qt;
              }
            }
          }
          // If no process was executed (CPU idle), advance time
          if(!executed) {
            t++;
          }
        }

        avg_wt = (float)wt/n;
        avg_tat = (float)tat/n;
        printf("\nAverage Waiting Time: %.2f", avg_wt);
        printf("\nAverage Turnaround Time: %.2f\n", avg_tat);
        return 0;
      }
```

Output:

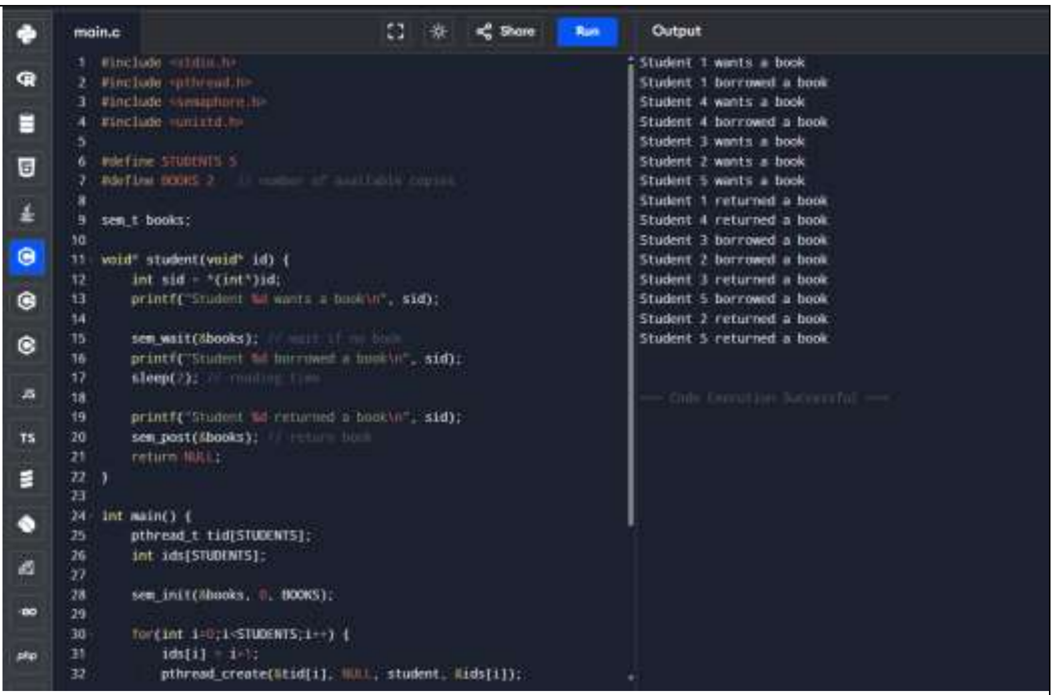
1. **Threads :-**

Vision of the Department

To be a well-known centre for pursuing computer education through innovative pedagogy, value-based education and industry collaboration.

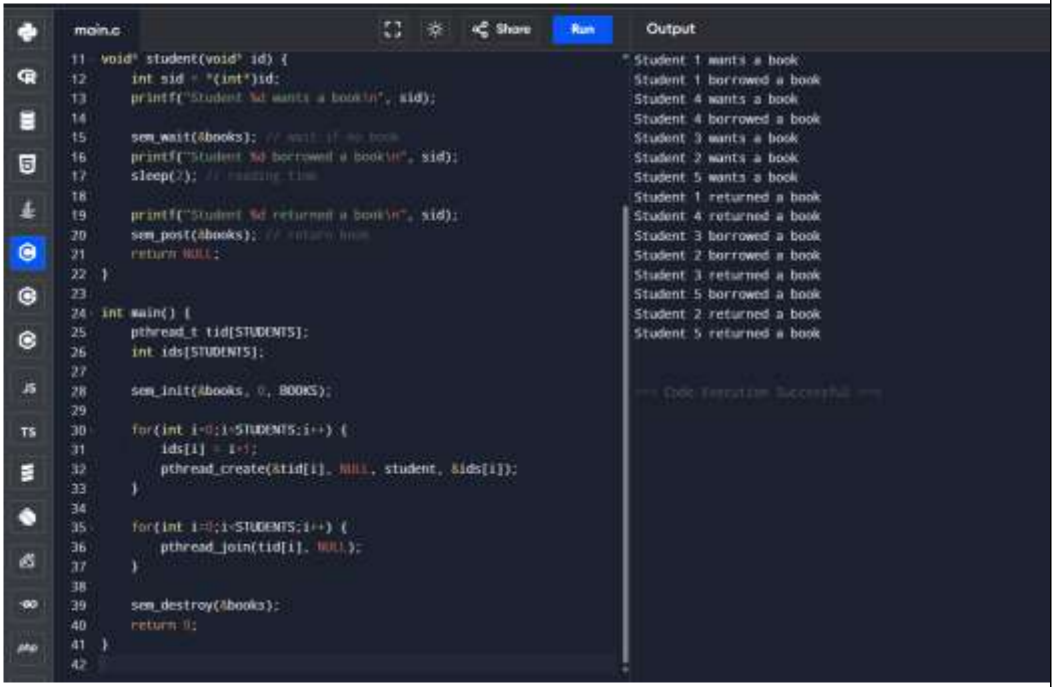
Mission of the Department

To establish learning ambience for ushering in computer engineering professionals in core and multidisciplinary area by developing Problem-solving skills through emerging technologies.



```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define STUDENTS 5
#define BOOKS 2   // number of available copies

sem_t books;

void* student(void* id) {
    int sid = *(int*)id;
    printf("Student %d wants a book\n", sid);

    sem_wait(&books); // wait if no book
    printf("Student %d borrowed a book\n", sid);
    sleep(2); // reading time

    printf("Student %d returned a book\n", sid);
    sem_post(&books); // return book
    return NULL;
}

int main() {
    pthread_t tid[STUDENTS];
    int ids[STUDENTS];

    sem_init(&books, 0, BOOKS);

    for(int i=0;i<STUDENTS;i++) {
        ids[i] = i+1;
        pthread_create(&tid[i], NULL, student, &ids[i]);
```

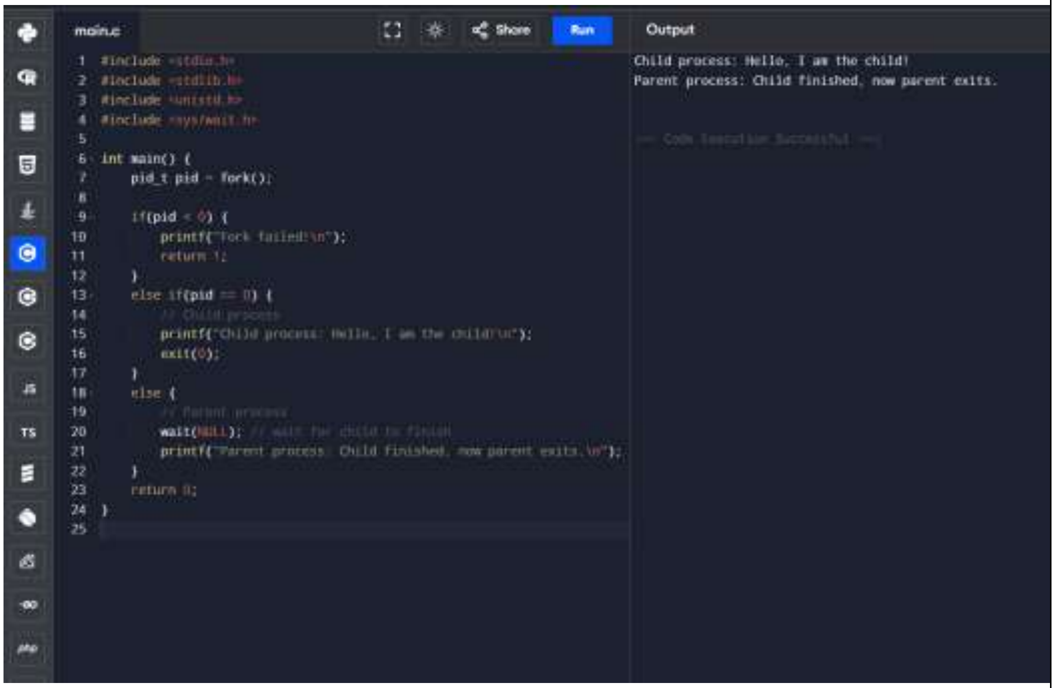
```
Student 1 wants a book
Student 1 borrowed a book
Student 4 wants a book
Student 4 borrowed a book
Student 3 wants a book
Student 2 wants a book
Student 5 wants a book
Student 1 returned a book
Student 4 returned a book
Student 3 borrowed a book
Student 2 borrowed a book
Student 3 returned a book
Student 5 borrowed a book
Student 2 returned a book
Student 5 returned a book


=== Code Execution Successful ===
```



```c
void* student(void* id) {
    int sid = *(int*)id;
    printf("Student %d wants a book\n", sid);

    sem_wait(&books); // wait if no book
    printf("Student %d borrowed a book\n", sid);
    sleep(2); // reading time

    printf("Student %d returned a book\n", sid);
    sem_post(&books); // return book
    return NULL;
}

int main() {
    pthread_t tid[STUDENTS];
    int ids[STUDENTS];

    sem_init(&books, 0, BOOKS);

    for(int i=0;i<STUDENTS;i++) {
        ids[i] = i+1;
        pthread_create(&tid[i], NULL, student, &ids[i]);
    }

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

    sem_destroy(&books);
    return 0;
}

```

```
Student 1 wants a book
Student 1 borrowed a book
Student 4 wants a book
Student 4 borrowed a book
Student 3 wants a book
Student 2 wants a book
Student 5 wants a book
Student 1 returned a book
Student 4 returned a book
Student 3 borrowed a book
Student 2 borrowed a book
Student 3 returned a book
Student 5 borrowed a book
Student 2 returned a book
Student 5 returned a book


=== Code Execution Successful ===
```

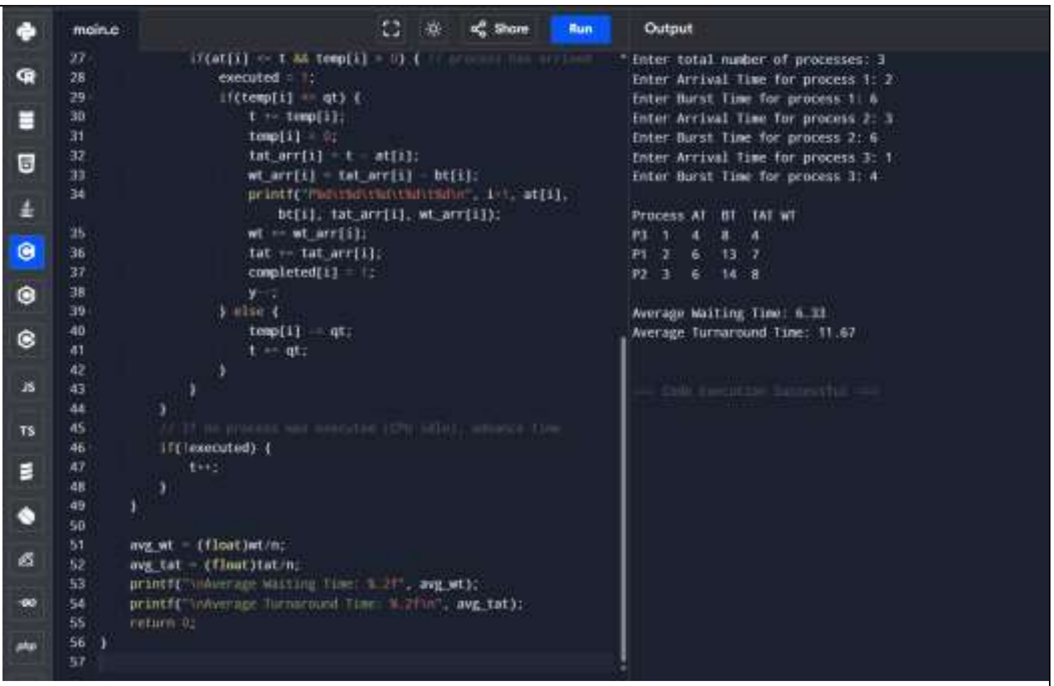
2. Fork :-



```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid = fork();

    if(pid < 0) {
        printf("Fork failed!\n");
        return 1;
    }
    else if(pid == 0) {
        // Child process
        printf("Child process: Hello, I am the child!\n");
        exit(0);
    }
    else {
        // Parent process
        wait(NULL); // wait for child to finish
        printf("Parent process: Child finished, now parent exits.\n");
    }
    return 0;
}
```

```
Child process: Hello, I am the child!
Parent process: Child finished, now parent exits.

=== Code Execution Successful ===
```

3. CPU Scheduling :-



```c
            if(at[i] <= t && temp[i] > 0) { // process has arrived
                executed = 1;
                if(temp[i] <= qt) {
                    t += temp[i];
                    temp[i] = 0;
                    tat_arr[i] = t - at[i];
                    wt_arr[i] = tat_arr[i] - bt[i];
                    printf("P%d\t%d\t%d\t%d\t%d\n", i+1, at[i],
                        bt[i], tat_arr[i], wt_arr[i]);
                    wt += wt_arr[i];
                    tat += tat_arr[i];
                    completed[i] = 1;
                    y--;
                } else {
                    temp[i] -= qt;
                    t += qt;
                }
            }
        }
        // If no process was executed (CPU idle), advance time
        if(!executed) {
            t++;
        }
    }

    avg_wt = (float)wt/n;
    avg_tat = (float)tat/n;
    printf("\nAverage Waiting Time: %.2f", avg_wt);
    printf("\nAverage Turnaround Time: %.2f\n", avg_tat);
    return 0;
}
```

```
Enter total number of processes: 3
Enter Arrival Time for process 1: 2
Enter Burst Time for process 1: 6
Enter Arrival Time for process 2: 3
Enter Burst Time for process 2: 6
Enter Arrival Time for process 3: 1
Enter Burst Time for process 3: 4

Process AT  BT  TAT WT
P3  1   4   8   4
P1  2   6   13  7
P2  3   6   14  8

Average Waiting Time: 6.33
Average Turnaround Time: 11.67


=== Code Execution Successful ===
```

Output Analysis

1. Library (Synchronization)
The output shows students requesting, borrowing, and returning books. At any time, only as many students as available book copies can borrow. If all books are borrowed, other students wait until one is returned. This confirms that semaphores correctly control access to limited resources.

2. Wait & Exit (System Calls)
The output first displays the child's message, then after the child exits, the parent prints its message. This shows that wait() ensures the parent waits for the child's completion, preventing zombie processes and maintaining proper process synchronization.

3. CPU Scheduling (Round Robin, Quantum = 2)
The output prints a table with Arrival Time, Burst Time, Turnaround Time, and Waiting Time for each process, along with average values. Processes execute in cyclic order, each getting equal time slices until completion. The analysis shows fairness, reduced starvation, and correct calculation of waiting and turnaround times.

Nagar Yuwak Shikshan Sanstha’s

Yeshwantrao Chavan College of Engineering

(An Autonomous Institution affiliated to Rashtrasant Tukadoji Maharaj Nagpur University)

Hingna Road, Wanadongri, Nagpur - 441 110

NAAC A++

Ph.: 07104-237919, 234623, 329249, 329250 Fax: 07104-232376, Website: www.ycce.edu

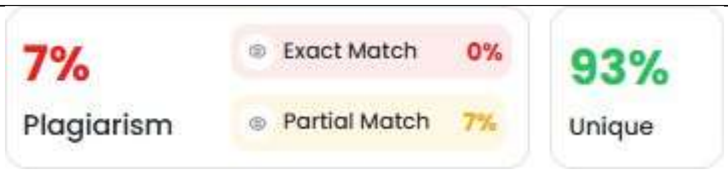
Department of Computer Technology B. Tech in Computer Science and Engineering (IOT)

Vision of the Department

To be a well-known centre for pursuing computer education through innovative pedagogy, value-based education and industry collaboration.

Mission of the Department

To establish learning ambience for ushering in computer engineering professionals in core and multidisciplinary area by developing Problem-solving skills through emerging technologies.

Link of student Github profile where lab assignment has been uploaded	https://github.com/karan-0123/OS_LAB
Conclusion	The three programs demonstrate key operating system concepts. The Library problem shows synchronization and resource sharing using semaphores. The Wait–Exit program illustrates parent–child process handling with system calls. The Round Robin scheduling program ensures fair CPU allocation and calculates turnaround and waiting times. Together, these enhance understanding of process management and scheduling in OS.
Plag Report (Similarity index < 12%)	 7% Plagiarism; Exact Match 0%; Partial Match 7%; 93% Unique
Date	29/09/2025