Department of Computer Science & 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: Dream of where you want.	**Mission:** Means to achieve Vision

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
Bhushan V. Tayade
10-08-2025

Department of Computer Science & 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	**2025-26 (ODD)**	**Course Name**	**Operating System Lab**
Semester	**5**	**Course Code**	**23IOT1504**
Roll Number	**035**	**Name of Student**	**Bhushan V. Tayade**

Practical Number	2
Course Outcome	1. Understand Computer System Configuration and Simulate system resources efficiently using Linux Commands (CO1) 2. Analyse operating system functionalities utilizing system calls, thread programming and process scheduling algorithms (CO2) 3. Apply Synchronization primitives to implement a Deadlock-free solution(CO3) 4. Simulate Disk scheduling, Memory allocation, File allocation, page replacement algorithms (CO4)
Aim	Implement programs of process, memory, and device management using appropriate Linux system calls.
Problem Definition	In operating systems, effective management of processes, memory, and devices is essential to ensure optimal system performance and resource utilization. The problem focuses on implementing and analyzing the core functionalities of an operating system through Linux system calls. The objective is to write programs that demonstrate how the kernel handles process creation and termination, memory allocation and deallocation, and device communication and control. By using system calls such as fork(), exec(), wait(), malloc(), free(), and file-handling operations, the experiment aims to provide practical insights into how these mechanisms interact within the Linux environment. This implementation helps in understanding low-level resource management and the coordination between hardware and software components.
Theory (100 words)	A system call is a programmatic interface that allows a user-level program to request services directly from the kernel of an operating system. It acts as a bridge between user processes and the core functionalities of the OS, enabling controlled access to hardware and system resources. System calls provide essential mechanisms for process control, memory

management, file handling, and device communication. They are the only entry points into the kernel and are executed in kernel mode, ensuring system security and stability. Without system calls, each program would need to implement its own low-level operations for hardware access, leading to inefficiency and inconsistency across applications.

System calls can be written in high-level languages such as C, C++, or Pascal, as well as in assembly language. When a program invokes a system call, it triggers a trap instruction that switches the processor from user mode to kernel mode. The kernel then executes the requested operation and returns the result to the user program.

Common System Calls in Linux

- open():
  The open() system call is used to access files within the file system. It allocates the required file resources and returns a file descriptor, which acts as a handle for subsequent operations. Files may be opened by one or multiple processes simultaneously, depending on the file system's access control mechanisms.

- wait():
  The wait() system call is used for process synchronization. When a parent process creates a child process, the parent may need to pause its execution until the child completes. The wait() call suspends the parent process, allowing the kernel to resume it only after the child process has terminated.

- fork():
  The fork() system call is used to create a new process, known as the child process, which is a duplicate of the parent. This mechanism is fundamental to process creation in Linux. After invoking fork(), both the parent and child execute concurrently, though they may perform different tasks based on process identifiers (PIDs).

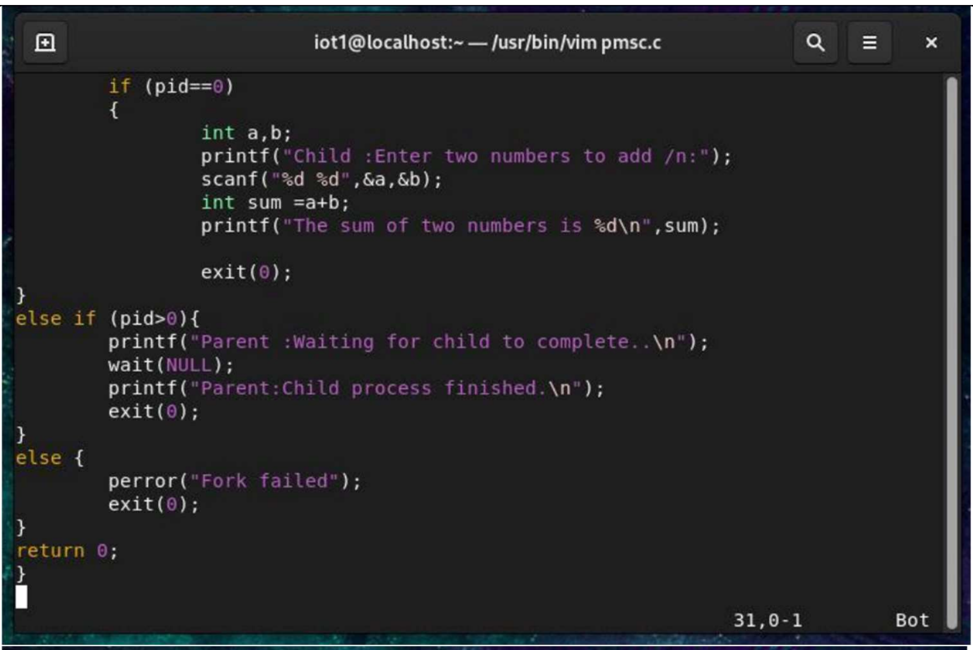
- exit():
  The exit() system call is used to terminate a process or thread. It signals the operating system that the process has completed its execution, prompting the kernel to reclaim the resources (memory, open files, CPU time) previously allocated to it.

In summary, system calls serve as the foundation of user-kernel interaction,

	enabling secure, efficient, and standardized access to operating system services. They are essential for process management, file manipulation, and resource allocation within the Linux environment.
Procedure and Execution (100 Words)	Step for Implementation: -
	Code:

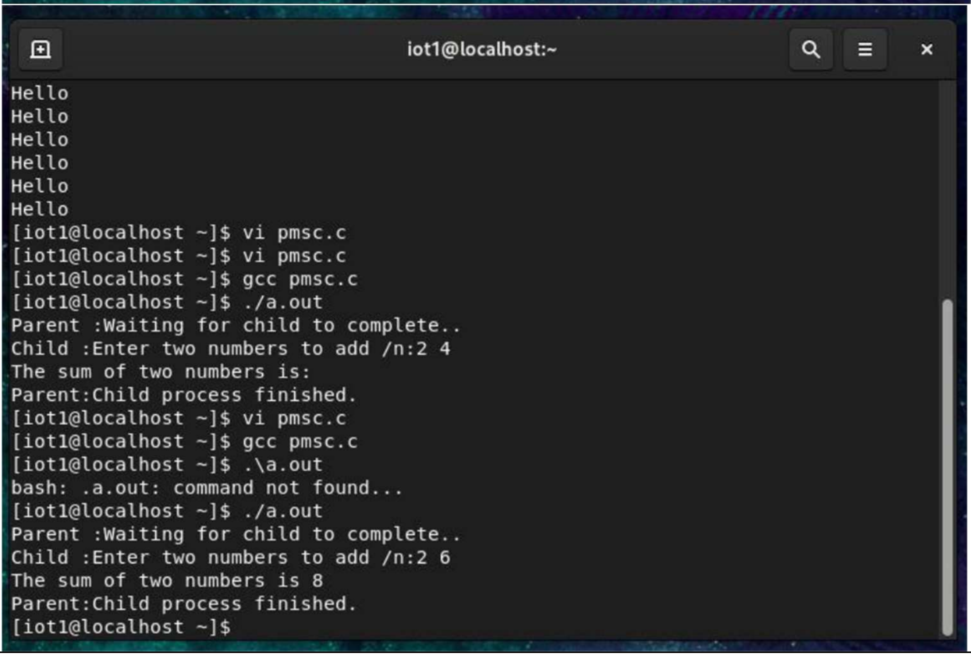
```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

#define MAX 1024

int main() {
    int fd;
    char filename[100];
    char content[MAX];
    char buffer[MAX];
    ssize_t bytesRead;

    // 1. Ask user for filename
    printf("Enter the filename: ");
    scanf("%s", filename);

    // 2. Ask user for content to write
    printf("Enter content to write to the file:\n");
    getchar(); // Consume leftover newline from scanf
    fgets(content, MAX, stdin);

    // 3. Open or create the file for writing
    fd = open(filename, O_CREAT | O_WRONLY | O_TRUNC, 0644);
    if (fd == -1) {
        perror("open for write");
        return 1;
    }

    // 4. Write content to the file
    if (write(fd, content, strlen(content)) == -1) {
        perror("write");
        close(fd);
        return 1;
    }
    close(fd);

    // 5. Open the same file for reading
    fd = open(filename, O_RDONLY);
    if (fd == -1) {
        perror("open for read");
        return 1;
    }

    // 6. Read the content back and display it
    printf("\nReading back the file contents:\n");
    while ((bytesRead = read(fd, buffer, sizeof(buffer) - 1)) > 0) {
        buffer[bytesRead] = '\0'; // Null-terminate the buffer
        printf("%s", buffer);
    }

    if (bytesRead == -1) {
        perror("read");
    }

    close(fd);
    return 0;
}
```

Output:



iot1@localhost:~ — /usr/bin/vim pmsc.c

```
        if (pid==0)
        {
                int a,b;
                printf("Child :Enter two numbers to add /n:");
                scanf("%d %d",&a,&b);
                int sum =a+b;
                printf("The sum of two numbers is %d\n",sum);

                exit(0);
}
else if (pid>0){
        printf("Parent :Waiting for child to complete..\n");
        wait(NULL);
        printf("Parent:Child process finished.\n");
        exit(0);
}
else {
        perror("Fork failed");
        exit(0);
}
return 0;
}

                                                    31,0-1        Bot
```



iot1@localhost:~

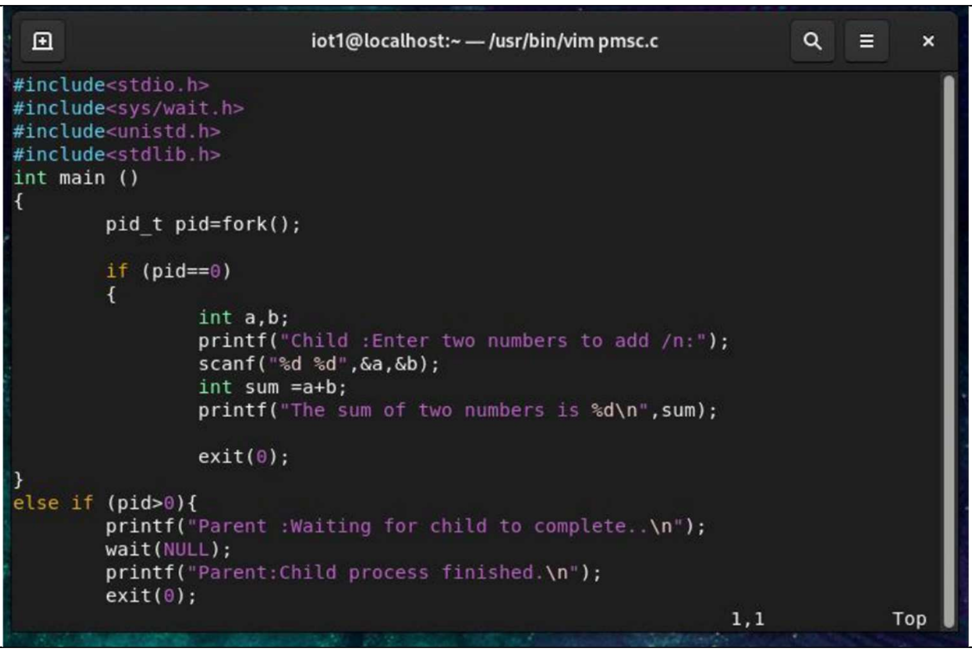
```
Hello
Hello
Hello
Hello
Hello
Hello
[iot1@localhost ~]$ vi pmsc.c
[iot1@localhost ~]$ vi pmsc.c
[iot1@localhost ~]$ gcc pmsc.c
[iot1@localhost ~]$ ./a.out
Parent :Waiting for child to complete..
Child :Enter two numbers to add /n:2 4
The sum of two numbers is:
Parent:Child process finished.
[iot1@localhost ~]$ vi pmsc.c
[iot1@localhost ~]$ gcc pmsc.c
[iot1@localhost ~]$ .\a.out
bash: .a.out: command not found...
[iot1@localhost ~]$ ./a.out
Parent :Waiting for child to complete..
Child :Enter two numbers to add /n:2 6
The sum of two numbers is 8
Parent:Child process finished.
[iot1@localhost ~]$
```

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

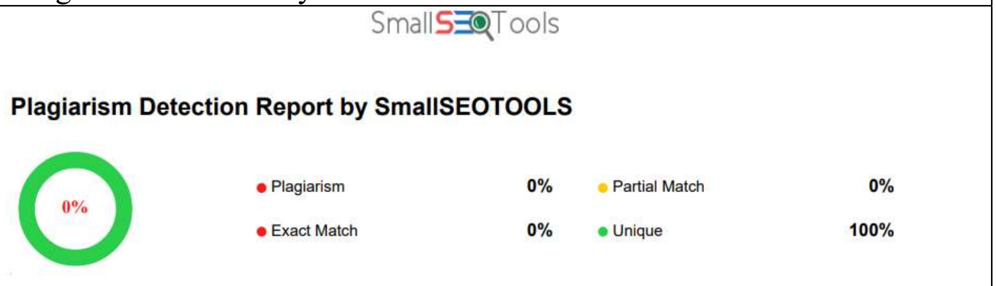
To establish learning ambience for ushering in computer engineering professionals in core and multidisciplinary area by developing Problem-solving skills through emerging technologies.

iot1@localhost:~ — /usr/bin/vim pmsc.c

```c
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdlib.h>
int main ()
{
        pid_t pid=fork();

        if (pid==0)
        {
                int a,b;
                printf("Child :Enter two numbers to add /n:");
                scanf("%d %d",&a,&b);
                int sum =a+b;
                printf("The sum of two numbers is %d\n",sum);

                exit(0);
}
else if (pid>0){
        printf("Parent :Waiting for child to complete..\n");
        wait(NULL);
        printf("Parent:Child process finished.\n");
        exit(0);
                                                              1,1           Top
```

Output Analysis	The implementation of Linux system calls was successfully carried out to manage processes, memory, and devices. • In process management, system calls such as fork() and exec() were used to create child processes, while wait() ensured proper synchronization between parent and child processes. The outputs verified successful process creation and termination, reflecting the hierarchical structure of process control in Linux. • In memory management, the use of dynamic allocation (malloc(), calloc(), realloc(), and free()) demonstrated efficient handling of runtime memory requirements. The system accurately allocated and released memory blocks without leaks. • In device management, file descriptors and system calls like open(), read(), write(), and close() were employed to facilitate communication between the kernel and I/O devices. The results confirmed successful data transfer and resource handling at the device level. Overall, the outputs validated the correct functioning of each system call and showcased the layered interaction between the user space and kernel space in Linux.
Link of student Github profile where lab	"https://github.com/Bhushan-Tayade/YCCN-23071391.git"

assignment has been uploaded	
Conclusion	The experiment effectively demonstrated the implementation of process, memory, and device management in Linux using appropriate system calls. Through hands-on programming, the core concepts of process hierarchy, memory allocation, and device communication were explored in depth. The successful execution of all programs confirmed that Linux provides a robust and flexible environment for system-level operations through its rich set of system calls. This practical exercise enhanced the understanding of internal operating system mechanisms, including how resources are allocated, synchronized, and managed efficiently. In conclusion, this experiment bridged theoretical knowledge with real-time implementation, reinforcing fundamental concepts of operating system design and functionality.
Plag Report (Similarity index < 12%)	 SmallSEOTools **Plagiarism Detection Report by SmallSEOTOOLS** 0% ● Plagiarism 0% ● Partial Match 0% ● Exact Match 0% ● Unique 100%
Date	10-08-2025