COLLEGE LOGO

Project Report on

"Study of threads in linux"

Submitted in partial fulfilment for the award of the degree
Of

MASTER OF COMPUTRE APPLICATIION

Branch- MCA

DEPARTMENT OF SCHOOL OF COMPUTING

COLLEGE NAME:DIT UNIVERSITY DEHRADUN UTTARAKHAND ,(INDIA)

COLLEGE ADDRESS: MUSSOORIE DIVERSION RAOD DEGRADUN,UTTARAKHAND,(INDIA),248009

Submitted To:

TEACHER NAME:MR.GOURAV DIHMAN

Submitted by: (SHASHANK BHARDWAJ),

SAP ID:(1000024713)

COLLEGE NAME:DIT UNIVERSITY

DEHRADUN,UTTARAKHAND,(INDIA)

COLLEGE ADDRESS:MUSSOORIE DIVERSION ROAD DEHRADUN, UTTARAKHAND,(INDIA),248009

DECLARATION (CERTIFICATE):-

We declare that minor project entitled **“Study of threads in linux”** is our own work conducted under the supervision of **“(Mr .GOURAV DIHMAN)”**, Department of Information Technology, **DIT UNIVERSITY MUSSOORIE DIVERSION ROAD DEHRADUN,UTTARAKHAND,(India),248009.** We further declare that, to the best of our knowledge the project does not contain any work which has been submitted for the award of the degree either in the University or in any other University/Deemed University without proper citations.

ACKNOWLEDGEMENT

I wish to express our sincere gratitude to God for his protection, providence, guidance and above all, for sustaining us.

I would like to express our special thanks of gratitude to our professor 'Professor Name' (Mr .GOURAV DIHMAN) as well as Our' HOD'(Mrs.BHARTI MAM) who gave me the golden opportunity to do this wonderful
Project on the topic "Study of threads in linux" which also helped me in doing
A lot of Research and gaining some precious knowledge.

Finally I wish to express our appreciation to our parents for their love and support.

Certificate of Completion

This is to certify that Shashank Bhardwaj(1000024713), of Department of computer science, School of Computing has successfully completed a project on the topic Study of threads in linux under the guidance of Mr. GOURAV DIHMAN for the academic year and session 2025-26 in partial fulfilment of the requirement for the award of the Degree Master in Computer Application Acknowledgement This acknowledgment section recognizes the contributions of individuals and groups who supported you during your project work, from your supervisor to peers and family. Feel free to tailor it with specific names or additional details relevant to your project!. Supervisor Signature Head of Department

Supervisor	**Head of department**
Signature	**Signature&seal**

CONTENTS

Mini Project Report: Study of Threads on Linux

1. Introduction

In the field of modern computing, threads are a fundamental concept that helps in achieving multitasking and efficient utilization of system resources. Threads are lightweight processes that enable a program to execute multiple tasks concurrently. In Linux, threads are implemented using the pthreads library, which stands for POSIX threads. The study of threads on Linux helps to understand the underlying mechanism of multithreading, synchronization techniques, and their impact on system performance. This project aims to explore the concept of threads in Linux, their creation, management, synchronization, and the challenges associated with multithreading.

2. Objectives

The main objectives of this mini project are:

- **To understand the concept of threads and how they differ from processes.**
- **To learn how threads are created and managed in Linux using the pthreads library.**
- **To investigate thread synchronization techniques and their importance in multithreaded applications.**
- **To analyze the impact of multithreading on system performance.**
- **To examine common challenges related to thread management, such as deadlock and race conditions.**

3. System Requirements

- **Operating System: Linux (Ubuntu or any other distribution)**
- **Programming Language: C**
- **Tools: GCC compiler, GDB debugger**
- **Libraries: POSIX threads library (pthread.h)**

4. Theory

4.1 Threads vs Processes

- **Process: A process is an independent program in execution, with its own memory space, system resources, and execution state.**
- **Thread: A thread is a smaller unit of execution within a process. Multiple threads can exist within the same process and share resources like memory, file descriptors, and the program counter.**

4.2 POSIX Threads (pthreads)

POSIX threads (pthreads) is a standard set of APIs for thread management in Unix-like operating systems. The pthreads library provides functionality for creating, managing, and synchronizing threads.

4.3 Thread Creation and Management

In Linux, threads are created using the `pthread_create()` function, which takes the following parameters:

- **A pointer to the thread identifier.**
- **A set of thread attributes (or `NULL` for default settings).**
- **The function to be executed by the thread.**
- **A single argument passed to the function.**

4.4 Thread Synchronization

Multithreaded programs often face challenges such as race conditions, where multiple threads try to access shared data simultaneously. To prevent this, synchronization mechanisms like mutexes, semaphores, and condition variables are used to ensure data consistency and avoid conflicts.

- **Mutexes: A mutex is a locking mechanism that ensures only one thread can access a critical section at a time.**
- **Semaphores: A semaphore controls access to a shared resource by multiple threads.**
- **Condition Variables: Condition variables allow threads to wait for certain conditions to be met before proceeding.**

5. Implementation

In this project, we implemented a multithreaded program to demonstrate the creation of threads, thread synchronization, and handling of shared data.

5.1 Thread Creation Example

The following C code demonstrates the creation of multiple threads in Linux using the `pthread_create()` function.

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

void* thread_function(void* arg) {
    printf("Hello from thread! Thread ID: %ld\n",
pthread_self());
```

```
    return NULL;
}

int main() {
    pthread_t threads[5];
    int i;

    for(i = 0; i < 5; i++) {
        if(pthread_create(&threads[i], NULL, thread_function,
NULL)) {
            printf("Error creating thread %d\n", i);
            return 1;
        }
    }

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

    printf("Main thread exiting.\n");
    return 0;
}
```

5.2 Thread Synchronization Example

The following C code demonstrates the use of mutexes for thread synchronization.

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

pthread_mutex_t lock;

void* thread_function(void* arg) {
    pthread_mutex_lock(&lock);  // Lock the mutex

    // Critical section
    printf("Thread ID: %ld has entered the critical section.\n",
pthread_self());

    pthread_mutex_unlock(&lock);  // Unlock the mutex
    return NULL;
}

int main() {
    pthread_t threads[3];
    int i;
    pthread_mutex_init(&lock, NULL);  // Initialize the mutex

    for(i = 0; i < 3; i++) {
        if(pthread_create(&threads[i], NULL, thread_function,
NULL)) {
            printf("Error creating thread %d\n", i);
            return 1;
        }
    }

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

    pthread_mutex_destroy(&lock);  // Destroy the mutex

    printf("Main thread exiting.\n");
    return 0;
}
```

5.3 Deadlock Example

A deadlock occurs when two or more threads are blocked forever, each waiting on the other to release resources.

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

pthread_mutex_t lock1, lock2;

void* thread1_function(void* arg) {
    pthread_mutex_lock(&lock1);
    printf("Thread 1 acquired lock1\n");

    pthread_mutex_lock(&lock2);  // This will cause a deadlock
    printf("Thread 1 acquired lock2\n");

    pthread_mutex_unlock(&lock2);
    pthread_mutex_unlock(&lock1);

    return NULL;
}

void* thread2_function(void* arg) {
    pthread_mutex_lock(&lock2);
    printf("Thread 2 acquired lock2\n");
    pthread_mutex_lock(&lock1);  // This will cause a deadlock
    printf("Thread 2 acquired lock1\n");

    pthread_mutex_unlock(&lock1);
    pthread_mutex_unlock(&lock2);

    return NULL;
}

int main() {
    pthread_t t1, t2;

    pthread_mutex_init(&lock1, NULL);
    pthread_mutex_init(&lock2, NULL);

    pthread_create(&t1, NULL, thread1_function, NULL);
    pthread_create(&t2, NULL, thread2_function, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    pthread_mutex_destroy(&lock1);
    pthread_mutex_destroy(&lock2);

    return 0;
}
```

6. Results and Discussion

- **Thread Creation: The threads were successfully created and executed concurrently. Each thread displayed its own ID and executed the `thread_function()`.**
- **Thread Synchronization: The use of mutexes allowed threads to enter the critical section one at a time, ensuring that no race conditions occurred. The program correctly demonstrated the locking and unlocking of the mutex to prevent simultaneous access to shared resources.**
- **Deadlock: In the deadlock example, two threads waited for each other's resources indefinitely, causing the program to hang. This is a classic example of a deadlock, demonstrating the importance of avoiding circular dependencies in multithreaded programs.**

7. Conclusion

In this mini project, we explored the concept of threads in Linux, focusing on their creation, management, and synchronization using the POSIX threads library. We demonstrated practical examples of multithreading, including thread creation, mutex-based synchronization, and potential issues like deadlock. Understanding thread management in Linux is crucial for developing efficient, concurrent applications that make

full use of system resources. The project highlighted the importance of careful synchronization to prevent race conditions and deadlocks in multithreaded programs.

8. Future Work

- **Exploring advanced synchronization techniques such as read-write locks and semaphores.**
- **Implementing thread pools for better resource management in large-scale applications.**
- **Analyzing thread performance in terms of scalability and efficiency in a multi-core system.**

References

1. **Linux Pthreads Programming: A POSIX Standard for Better Threads (Book)**
2. **POSIX Threads Programming Manual**
3. **Manual Pages (pthread_create, pthread_mutex, etc.)**