# Distributed Operating System

Rakesh Kumar Rai
Assistant Professor(1)
School of Computer Engineering
KIIT Deemed to be University,
Bhubaneswar

# Contents

- Process and Threads
- Introduction
- Usage
- Implementing Thread in User Space and Kernel Space

# 1. Introduction

**Processes**

- **Definition**: A process is an instance of a program in execution. It is an active entity, with a program counter specifying the next instruction to execute and a set of associated resources (such as memory, files, and I/O devices).
- **Characteristics**:
    - o Has its own memory space.
    - o Can communicate with other processes via Inter-Process Communication (IPC) mechanisms like pipes, shared memory, message queues, etc.
    - o More overhead due to context switching.

## Threads

- **Definition**: A thread is the smallest unit of execution within a process. Multiple threads can exist within the same process, sharing the same memory space.
- **Characteristics**:
  - o Share the same memory space and resources of the process.
  - o More efficient than processes due to lower context-switching overhead.
  - o Easier and more efficient for communication within the same process.

**Why Use Processes?**

- **Isolation**: Processes are isolated from each other, providing better fault tolerance and security.
- **Resource Management**: Suitable for tasks that require a dedicated amount of resources.

**Why Use Threads?**

- **Efficiency**: Threads are lightweight compared to processes, allowing faster creation, termination, and context switching.
- **Shared Memory**: Easier communication and data sharing since threads within the same process share memory.
- **Concurrency**: Useful for performing multiple operations concurrently within the same application.

# Implementing Threads in User Space and Kernel Space

## User Space Threads

User space threads are managed entirely by a user-level library, without kernel awareness. Below are steps to implement user space threads.

**Steps to Implement User Space Threads:**

1. **Thread Library**:Use a threading library like POSIX Threads (Pthreads) for user space thread management.

2. **Thread Creation**:Use the library's API to create threads. For example, in Pthreads, use pthread_create.

**3.Thread Management**:The library handles scheduling, context switching, and synchronization between threads.

**4.Synchronization**:Use mutexes, condition variables, or other synchronization primitives provided by the threading library to manage access to shared resources.

## Code for User Space Threads:

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

void* thread_function(void* arg) {
    printf("Thread ID: %ld\n", pthread_self());
    return NULL;
}

int main() {
    pthread_t thread1, thread2;

    if (pthread_create(&thread1, NULL, thread_function, NULL)) {
        fprintf(stderr, "Error creating thread\n");
        return 1;
    }
```

# Code for User Space Threads (Cont'd...)

```
    if (pthread_create(&thread2, NULL, thread_function, NULL)) {
        fprintf(stderr, "Error creating thread\n");
        return 1;
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

**Kernel Space Threads**

Kernel space threads are managed directly by the operating system kernel. Below are steps to implement kernel space threads.

**Steps to Implement Kernel Space Threads:-**

1. **Thread Creation**:Use system calls or native threading APIs provided by the operating system.
   **Thread Management**:The kernel manages scheduling, context switching, and synchronization between threads.
   **Synchronization**:Use OS-provided synchronization primitives like mutexes, semaphores, or condition variables.

# Code for Kernel Space Threads:

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

void* thread_function(void* arg) {
    printf("Thread ID: %ld\n", pthread_self());
    return NULL;
}

int main() {
    pthread_t thread1, thread2;

    if (pthread_create(&thread1, NULL, thread_function, NULL)) {
        fprintf(stderr, "Error creating thread\n");
        return 1;
    }
```

## Code for Kernel Space Threads (Cont’d...):

```
    if (pthread_create(&thread2, NULL, thread_function, NULL)) {
        fprintf(stderr, "Error creating thread\n");
        return 1;
    }

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

# Thank You