



بِسْمِ اللَّهِ وَنُصَلِّهِ وَنُسَلِّمُ عَلَيْهِ رَسُوْلَهُ الْكَرِيْمِ ، وَعَلَيْهِ اٰلِهٖ وَصَلَّيْهِ اٰجْمَعِيْنَ ، اَمَّا بَعْدُ :



Operating System Concept

Threads

Week 04

Lecturer: Mashal Rokhan

Master of Science in Computer Science

Email: m.rokhan@kardan.edu.af / mashalrokhan@gmail.com

Week 4 Course Outline

- Threads and Multithreaded Programming
- Process Synchronization
- Multicore programming
- Multithread Models
- Implicit Threading
- Thread Issues



Learning Objectives

Thread

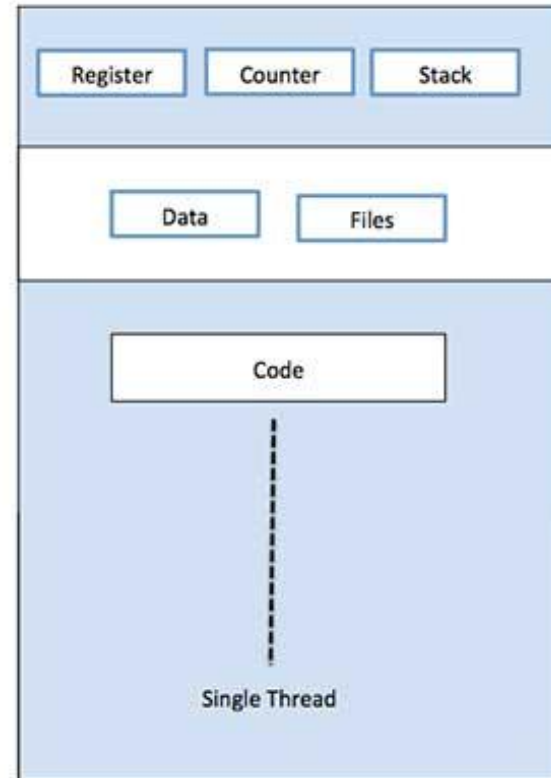
- A **thread** is a single flow of execution within a process.
- Each thread has its own **Program Counter (PC)** that tracks the next instruction to execute.
- Each thread maintains its own **CPU registers** and **stack** for execution history and local variables.
- Threads within the same process **share code, data, and open files**.
- A thread is often called a **Lightweight Process (LWP)** because it requires fewer resources than a process.
- Threads improve system and application performance through **parallelism and concurrency**.
- Creating and managing threads involves **less overhead** than creating full processes.
- Every thread belongs to **exactly one process** and cannot exist independently.

Thread

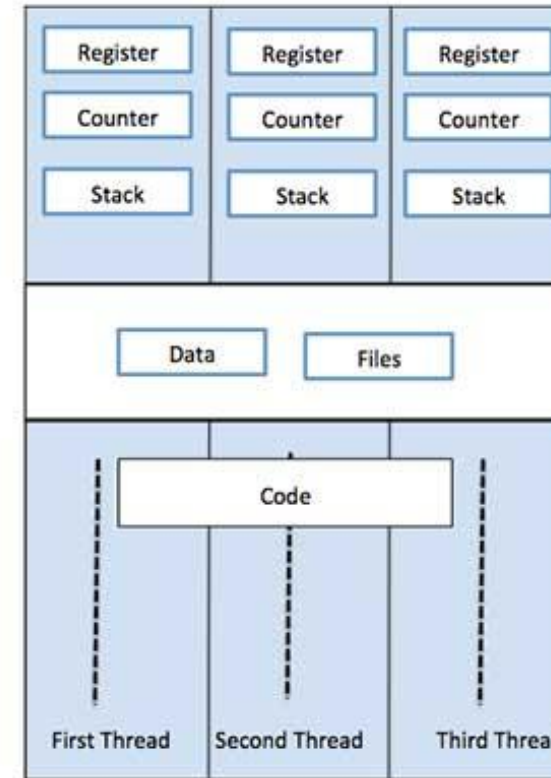
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads.
- **Update display**
- **Fetch data**
- **Spell checking**
- **Answer a network request**
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

Thread

- The following figure illustrates a multithreaded process.



Single Process P with single thread



Single Process P with three threads

headed and a

Difference between Process and Thread

Process	Thread
Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

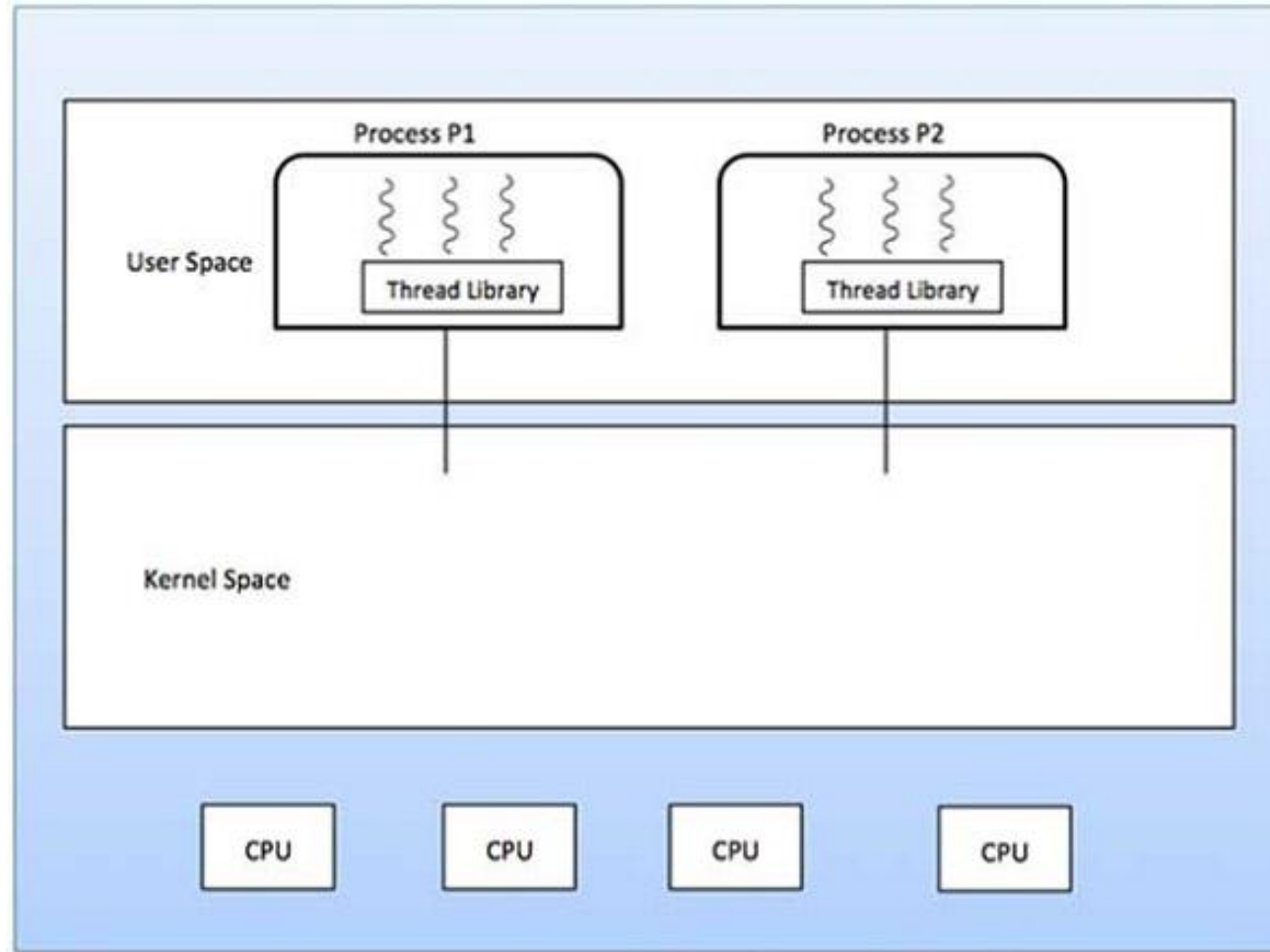
Types of Thread

1. **User Level Threads** – User managed threads.
2. **Kernel Level Threads** – Operating System managed threads acting on kernel, an operating system core.

User Level Threads

- In **User-Level Threads (ULTs)**, the operating system kernel is **not aware** of thread existence.
- Thread management is handled entirely by a **user-level thread library**.
- The thread library provides functions for **creating and terminating threads**.
- It manages **communication and data sharing** between threads.
- The library is responsible for **thread scheduling** and execution control.
- It saves and restores **thread contexts** during thread switches.
- Applications typically begin execution with **a single thread**.

User Level Threads



User Level Threads

Advantages

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Disadvantages

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

Kernel Level Threads

- In **Kernel-Level Threads (KLTs)**, thread management is performed directly by the **operating system kernel**.
- There is **no thread management code** in the user application area.
- The operating system has **full knowledge of all threads** within a process.
- All threads of an application execute within the **same process**.
- **Thread scheduling** is performed by the kernel on a thread-by-thread basis.
- The kernel handles **thread creation, execution, synchronization, and termination**.
- If one thread blocks, **other threads of the same process can continue executing**.
- Creating and managing kernel threads requires **system calls**, resulting in higher overhead.
- Kernel threads are generally **slower to create and switch** compared to user-level threads.