Threading in Operating Systems

What is a Thread?

A thread is the smallest unit of CPU execution within a process. It is sometimes called a lightweight process because it exists within a process and shares some resources of the process.

- A process can have one or more threads.
- Threads share process resources (memory, code, data, files) but have their own execution context (registers, program counter, stack).

Key Components of a Thread:

1. Thread ID – Unique identifier for the thread.
2. Program Counter (PC) – Keeps track of the next instruction.
3. Registers – Store CPU state.
4. Stack – For function calls and local variables.
5. Shared resources – Threads of the same process share code, data, and OS resources.

Thread vs Process

Features	Process	Thread
Resource Ownership	Own memory, files, I/O	Shares memory & files
Creation Overhead	High	Low
Context Switching	Expensive	Cheap
Communication	Inter-process communication	Can communicate directly (shared memory)
Execution Unit	Independent	Part of Process

Types of Threads

A. User-Level Threads (ULT)

- Managed by a thread library at user level (e.g., POSIX threads).
- OS kernel does not know about them.
- Advantages: Fast creation, low overhead.
- Disadvantages: If one thread blocks, the entire process blocks.

B. Kernel-Level Threads (KLT)

- Managed directly by the OS kernel.
- Advantages: Kernel can schedule threads independently; if one thread blocks, others continue.

- Disadvantages: Slower to create and manage.

C. Hybrid Threads

- Combination of ULT and KLT.
- User-level thread library maps many user threads to fewer kernel threads.

Thread Models

- Many-to-One Model – Many user threads mapped to a single kernel thread.
- One-to-One Model – Each user thread maps to a kernel thread.
- Many-to-Many Model – Multiple user threads mapped to multiple kernel threads.

Advantages of Threads

- Improved Responsiveness.
- Resource Sharing.
- Economical – Lower overhead compared to full processes.
- Efficient Communication.
- Better CPU Utilization.

Thread Operations

- Creation – pthread_create() in POSIX, CreateThread() in Windows.
- Termination – pthread_exit() or return from function.
- Synchronization – mutexes, semaphores, condition variables.
- Scheduling – Kernel schedules threads independently or according to thread model.

Context of Threads in OS

- Multithreading – Multiple threads executing concurrently within a process.
- Parallel Computing – Exploiting multi-core CPUs.
- Server Applications – Each client request can be handled by a separate thread.
- Real-Time Systems – Threads allow tasks to meet timing constraints.

Example Scenario

Web browser:

- One thread handles UI updates.
- Another thread handles downloading files.
- Another thread handles rendering a webpage.

Threads allow these tasks to happen concurrently without blocking each other, improving responsiveness.

Thread Safety Issues

- Race Condition – Two threads access shared data simultaneously, causing inconsistent results.
- Deadlock – Two or more threads wait indefinitely for resources.
- Starvation – Thread never gets CPU due to scheduling.
  Solution: Use mutexes, semaphores, or monitors to synchronize threads safely.