**THREAD**

**What is a Thread?**

A **thread** is the **smallest unit of execution** within a process. It is a lightweight, independent flow of control that shares the **same memory space** with other threads in the same process.

**🔹 Key Characteristics of a Thread:**

* A thread **belongs to a process** and shares its **code, data, and open files**.
* It has its own **stack, program counter, and registers**.
* Threads allow **concurrent execution** within a process.
* **Faster** than creating a new process because threads share resources.

**🔹 Types of Threads:**

1. **User-Level Threads (ULT)** → Managed by user libraries, not the OS.
2. **Kernel-Level Threads (KLT)** → Managed by the OS kernel.

**🔹 Advantages of Threads:**

✔️ **Faster execution** (less overhead than processes).  
✔️ **Efficient CPU utilization** through parallel execution.  
✔️ **Quick context switching** (since threads share resources).  
✔️ **Easier communication** between threads (shared memory).

**🔹 Disadvantages of Threads:**

❌ **Synchronization issues** (since threads share memory).  
❌ **One thread crash can crash the whole process**.  
❌ **Harder to debug** due to race conditions.

**🔹 Example Use Cases:**

✅ **Web Browsers** → Multiple tabs run as separate threads.  
✅ **Word Processors** → Auto-save runs in a background thread.  
✅ **Servers** → Handle multiple client requests using threads.  
✅ **Gaming** → Graphics rendering and AI computations run in different threads.

**What is Multithreading in OS?**

**Multithreading** is the ability of a CPU to execute multiple threads **within the same process** simultaneously. It allows for better **resource utilization** and improves system performance by enabling parallel execution of tasks.

**🔹 Key Features of Multithreading:**

* **Multiple threads share** the same process **memory** (code, data, files).
* Each thread has its own **stack, registers, and program counter**.
* Threads can run **concurrently** on single-core CPUs and in **parallel** on multi-core CPUs.
* The OS manages thread scheduling and execution.

**Thread v/s Process**

**Thread vs. Process: Key Differences**

| **Feature** | **Process** | **Thread** |
| --- | --- | --- |
| **Definition** | An independent execution unit with its own memory space. | A lightweight execution unit within a process that shares memory with other threads. |
| **Memory** | Each process has its own memory space. | Threads within the same process share memory. |
| **Communication** | Inter-process communication (IPC) is required, which is slower. | Threads communicate more efficiently by sharing memory. |
| **Overhead** | More overhead in creation and context switching. | Less overhead in creation and switching. |
| **Execution** | Runs independently; failure of one process does not affect others. | A crash in one thread may affect the entire process. |
| **Concurrency** | Supports true parallel execution if running on multiple CPUs. | Supports parallel execution but relies on shared memory. |
| **Context Switching** | Heavy, as it involves switching memory and resources. | Lighter, as it only switches registers and stack. |
| **Example Use Cases** | Running separate applications (e.g., Chrome vs. Notepad). | Handling multiple tasks within the same application (e.g., multiple tabs in Chrome). |

### ****Similarity between Thread and Process****

1. **Execution Unit** → Both are used for executing tasks in a program.
2. **Concurrency & Parallelism** → Both enable multitasking and can run concurrently.
3. **CPU Utilization** → Both help in utilizing CPU efficiently by executing multiple tasks.
4. **Lifecycle** → Both go through states like creation, execution, waiting, and termination.
5. **Scheduling** → Both are managed by the operating system’s scheduler.

### ****Types of Threads in Operating Systems****

Threads in an OS can be categorized into two main types:

### ****User-Level Threads (ULT)****

* Managed **entirely by user-space libraries** without OS kernel intervention.
* The **kernel is unaware** of these threads.
* **Fast and lightweight**, as no kernel mode transitions occur.

🔹 **Example:** Java threads (managed by the JVM), POSIX threads (Pthreads).

#### ****Advantages:****

✔️ Faster creation and management.  
✔️ No kernel involvement → less overhead.  
✔️ More control for user-space applications.

#### ****Disadvantages:****

❌ If one thread blocks, the entire process blocks.

### ****Kernel-Level Threads (KLT)****

* Managed **by the OS kernel**, not by user-space libraries.
* The kernel is aware of all threads and schedules them separately.
* Supports **true parallelism** using multiple CPUs.
* **Slower than user-level threads** due to kernel overhead.

🔹 **Example:** Windows threads, Linux kernel threads, Solaris threads.

#### ****Advantages:****

✔️ OS schedules threads independently → better CPU utilization.  
✔️ One blocked thread does not affect others.  
✔️ Supports multi-core execution.

#### ****Disadvantages:****

❌ Slower thread creation and context switching.  
❌ Requires more system resources.

### ****Hybrid (User + Kernel) Threads****

Some systems use a mix of user and kernel threads to optimize performance.

🔹 **Example:** Solaris uses a two-level threading mode.

* Some threads are mapped to kernel threads, while others stay in user space.

### ****Types of Multithreading Models in OS****

Multithreading models define how user-level threads (ULTs) map to kernel-level threads (KLTs). There are three main models:

## ****Many-to-One Model (M:1)****

🔹 **Definition:** Multiple user threads are mapped to a single kernel thread.  
🔹 **How it Works:** The thread library in user space handles all threading operations.

#### ****✔️ Advantages:****

✅ **Fast & efficient** (no kernel intervention).  
✅ **Low overhead** (as the OS only manages one kernel thread).

#### ****❌ Disadvantages:****

❌ If one thread **blocks**, the entire process blocks.  
❌ Cannot take advantage of **multi-core processors** (since only one kernel thread runs).

🔹 **Example OS:** **Solaris Green Threads, GNU Portable Threads (GNU Pth)**

## ****One-to-One Model (1:1)****

🔹 **Definition:** Each user thread maps to a separate kernel thread.  
🔹 **How it Works:** The OS directly manages and schedules each thread.

#### ****✔️ Advantages:****

✅ Supports **true parallelism** on multi-core CPUs.  
✅ If one thread **blocks**, other threads keep running.

#### ****❌ Disadvantages:****

❌ **Higher resource usage** (each thread needs a kernel thread).  
❌ **Thread creation is slower** due to kernel involvement.

🔹 **Example OS:** **Windows, Linux, macOS (modern POSIX threads)**

## ****Many-to-Many Model (M:N)****

🔹 **Definition:** Multiple user threads are mapped to multiple kernel threads.  
🔹 **How it Works:** The OS schedules some user threads on available kernel threads.

#### ****✔️ Advantages:****

✅ **Flexible & efficient** (avoids kernel overhead while utilizing multi-core CPUs).  
✅ If a user thread **blocks**, others can still execute.

#### ****❌ Disadvantages:****

❌ More **complex to implement** compared to the other two models.

🔹 **Example OS:** **Solaris 9+, Windows with Fibers, modern Linux with NPTL**

### ****🔹 Summary Table****

| **Model** | **User Threads per Kernel Thread** | **Parallel Execution** | **Blocking Issue** | **Overhead** | **Example OS** |
| --- | --- | --- | --- | --- | --- |
| **Many-to-One** (M:1) | Multiple ULTs → One KLT | ❌ No | ❌ Yes | ✅ Low | Solaris Green Threads, GNU Pth |
| **One-to-One** (1:1) | One ULT → One KLT | ✅ Yes | ✅ No | ❌ High | Windows, Linux (POSIX), macOS |
| **Many-to-Many** (M:N) | Multiple ULTs → Multiple KLTs | ✅ Yes | ✅ No | 🔄 Medium | Solaris 9+, Windows Fibers, Linux NPTL |

**Threading Issues in OS**

Threading issues in an operating system (OS) arise when multiple threads execute concurrently, leading to potential problems such as race conditions, deadlocks, and resource contention. Here are some common threading issues:

the operating system (OS) does **not** schedule user-level threads directly. Instead, user-level threads are managed by a **user-level thread library** (such as pthreads, Java threads, or green threads), and the OS only schedules kernel threads.

### 1. ****Race Conditions****

* Occur when multiple threads access shared data simultaneously, and the final outcome depends on the order of execution.
* Example: Two threads updating a shared variable without proper synchronization, leading to inconsistent or incorrect results.

### 2. ****Deadlocks****

* A situation where two or more threads are waiting for resources held by each other, causing an indefinite block.
* Example: Thread A locks resource X and waits for resource Y, while Thread B locks resource Y and waits for resource X.

### 3. ****Starvation****

* A thread may be denied access to resources for an extended period because higher-priority threads keep executing.
* Example: A low-priority thread never gets CPU time in a priority-based scheduling system.

### 4. ****Context Switching Overhead****

* Frequent switching between threads increases CPU overhead due to saving and restoring thread states.
* Example: Excessive multitasking causing performance degradation.

### 5. ****Synchronization Issues****

* Improper use of synchronization mechanisms like mutexes, semaphores, or condition variables can lead to performance bottlenecks or bugs.
* Example: Overuse of locks causing unnecessary delays or underuse leading to race conditions.

#### ****Solutions to Threading Issues****

* **Mutexes and Locks**: Prevent race conditions by ensuring exclusive access to shared resources.
* **Semaphores**: Control access to limited resources among multiple threads.
* **Deadlock Prevention Algorithms**: Avoid circular waits, hold-and-wait conditions.
* **Priority Inheritance Protocol**: Solves priority inversion by temporarily boosting the priority of a low-priority thread holding a resource.
* **Efficient Thread Scheduling**: OS should optimize context switching and avoid excessive thread creation.

 If a system has **multiple CPU cores**, threads **can** execute **in parallel**.

 If a system has **a single core**, threads run **concurrently** (one at a time, but switched rapidly).

 The OS, hardware, and threading model determine the actual execution behavior.