Operating Systems
Course Code: **71203002004**

Threads

by -
Minal Rajwar

What is a Thread?

- A **thread** is a lightweight part of a process that runs independently.
- Multiple threads can exist in a single process, sharing the same code, data, and OS resources, but each has its own stack, registers, and program counter.
- In systems that support **multithreading**, threads help improve performance, especially on multi-core CPUs.
- On a single CPU, threads take turns using the processor (context switching).

Why Use Threads?

Threads allow **concurrent execution**, which boosts application performance.

For example, in apps like Microsoft Word or Google Docs, typing, auto-saving, and formatting can happen at the same time.

Key features:

- Threads share memory, so no need for inter-process communication.
- Each thread has its own **Thread Control Block (TCB)**.
- Threads can have different priorities.
- Synchronization is needed to manage shared data.

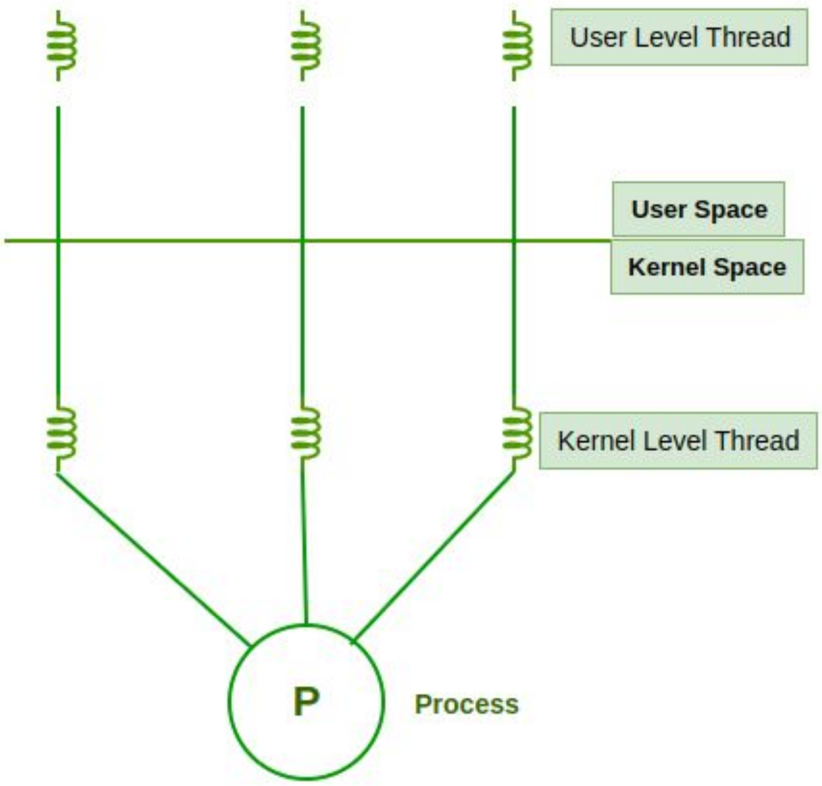
Components of Thread

- These are the basic components of the Operating System.
    - **Stack Space:** Stores local variables, function calls, and return addresses specific to the thread.
    - **Register Set:** Hold temporary data and intermediate results for the thread's execution.
    - **Program Counter:** Tracks the current instruction being executed by the thread.

Types of thread in OS

Thread are of two types:

1. User level
2. Kernel level



User-Level Threads (ULT)

- Managed by the application, not the OS.
- **Advantages:**
  - Easy to implement
  - Fast context switching
  - More efficient in simple tasks
- **Disadvantages:**
  - OS doesn't manage them, so no CPU optimization
  - One blocking thread can pause the whole process
  - Complex scheduling for multiple threads

Kernel-Level Threads (KLT)

- Managed by the OS kernel.

Advantages:

- Can run on multiple cores
- OS manages scheduling and load balancing
- Better for frequent blocking tasks

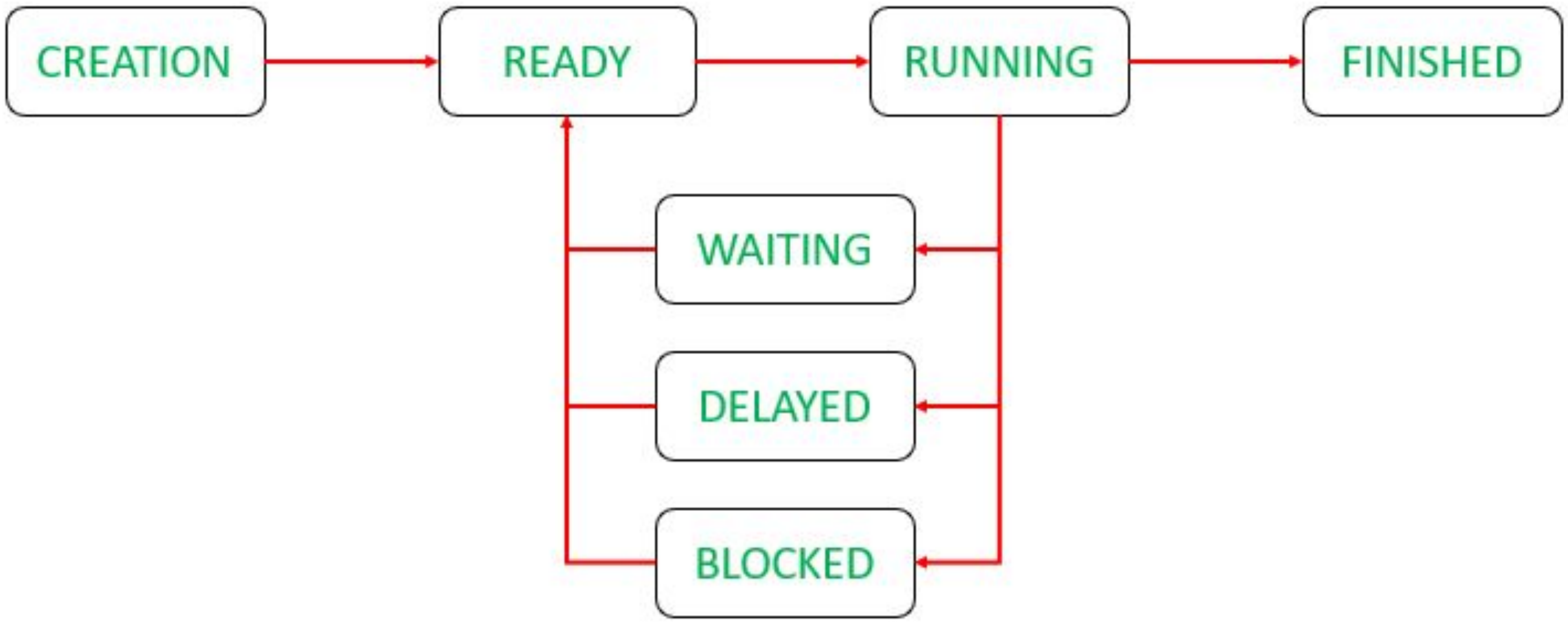
Disadvantages:

- Slower context switching
- Higher CPU overhead due to system calls
- More complex to implement

Thread States in OS

A thread in an operating system can be in one of the five-main states (excluding creation and finished states):

1. **Ready :** thread is prepared to run and waiting for CPU assignment.
2. **Running:** thread id currently being executed by the CPU.
3. **Waiting:** thread is paused , waiting for an event or signal from another thread/ process.
4. **Delayed:** thread is intentionally put to sleep for a fixed time (e.g., snoozed alarm).
5. **Blocked:** thread is waiting for an I/O operation to complete (e.g., user input).


CREATION
READY
RUNNING
FINISHED
WAITING
DELAYED
BLOCKED

Thread State Transition

1. Ready > Running

When the CPU scheduler assigns a thread to the processor.

1. Running > Waiting

When the thread needs to wait for another event or process to complete.

1. Running > Delayed

When the thread is programmatically paused for a set time.

1. Running > Blocked

When the thread performs an I/O and must wait for it to finish

1. Running > Finished

When the thread completes its task.

Key Differences

- **Waiting**: Waits for an external signal or event with **known burst time** (e.g., waiting for another thread).
- **Blocked**: Waits for **unknown time**, often due to user input or I/O operations.

Thread Control Block (TCB)

To track thread states and manage scheduling, the **OS uses a Thread Control Block (TCB)**, which stores:

- Thread ID
- State
- Program counter
- Register values
- Stack pointer
- Scheduling information

Difference between Thread and Process

Feature	**Process**	**Thread**
Memory	Has its own separate memory space	Shares memory with other threads
Independence	Independent from other processes	Dependent on other threads in the process
Communication	Requires inter-process communication	Can communicate directly via shared memory
Resources	Has its own OS resources	Shares OS resources (e.g., files, signals)
Components	Has its own PC, registers, and stack	Has its own PC, registers and stack
Overhead	Higher(more resource intensive)	Lower(lightweight)

Feature	Process	Thread
Context Switching	Slower (more data to save/load)	Faster (less data to manage)
Creation Time	Slow to create	Faster to create
Crash Impact	One process crashing doesn’t affect others	A crashing thread can affect the whole process.

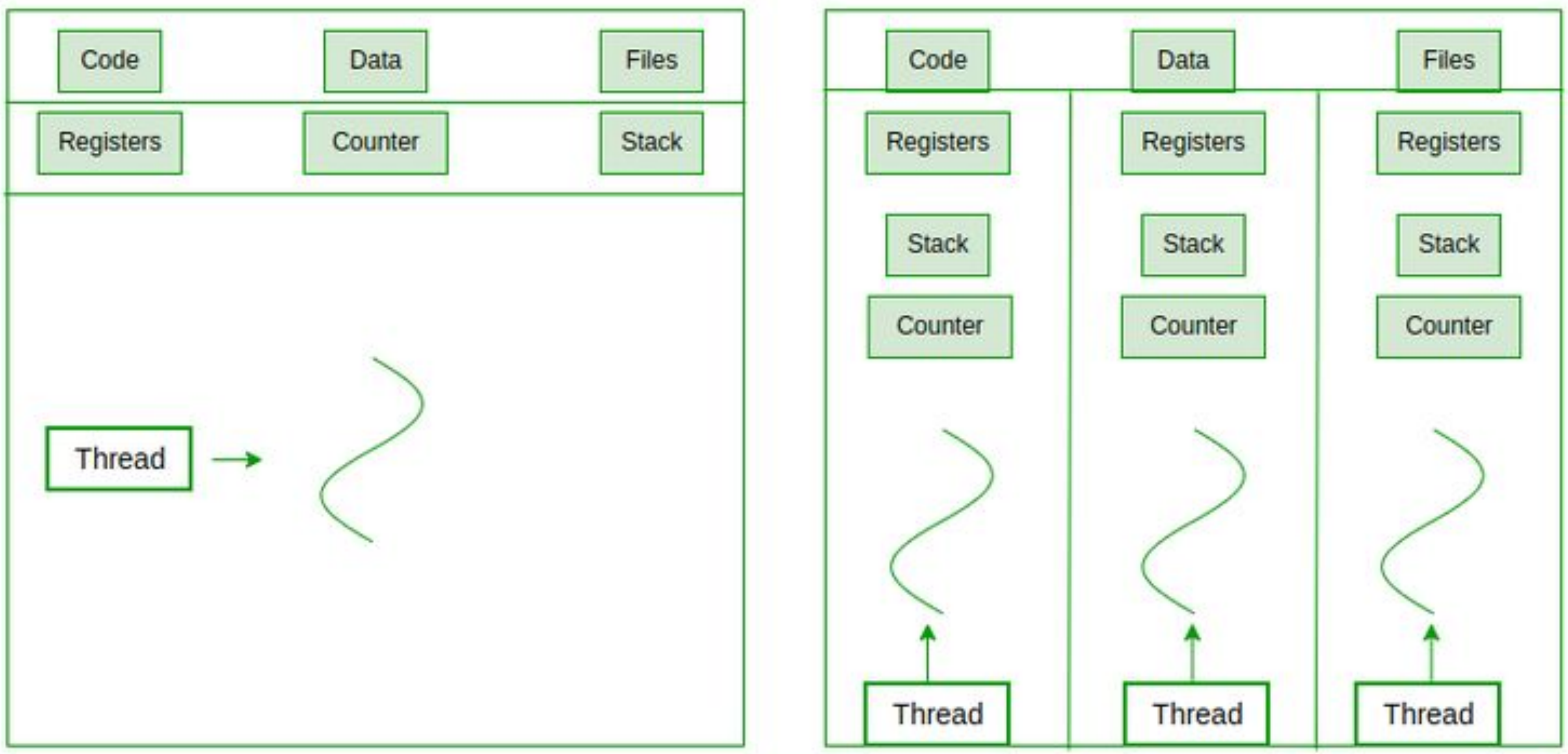
What is a Multi-Threading?

Multithreading is the ability of a process to run multiple threads concurrently to achieve **parallelism** and improve performance.

- A **thread** is a lightweight unit of a process.
- All threads in a process share the same memory and resources (code, data, files).
- Each thread has its own **stack**, **program counter**, and **registers**.

Examples:

- In a browser, each tab can be a thread.
- In MS Word, one thread handles typing, another handles formatting, another handles autosaving, etc.



Single Threaded Process

Multi Threaded Process

How It Works

- Threads can be managed by the **operating system (kernel-level)** or by the **application itself (user-level)**.
- For example, **Java** uses the JVM to manage threads without depending on the OS.
- User-level threads are handled by a **thread library** within the application; the OS is unaware of them.

Benefits of Multithreading

- **Responsiveness:** threads can return result as they finish, improving user experience.
- **Faster Context Switching:** Switching between threads is quicker than between full process.
- **Better Use of Multiprocessors :** Threads can run in parallel on multiple CPU cores, speeding up execution.
- **Resource Sharing:** Threads share code, data and files, reducing duplication.

- **Easier Communication:** threads use shared memory, avoiding complex inter process communication.
- **Higher Throughout:** More work (threads) done in less time increases system efficiency.

DISCUSSION & REVISION

1. What does OS assign to each new process?
2. Which component decides which process runs next?
3. What happens to a process when it finishes execution?
4. What do processes use to communicate with each other?
5. What prevents processes from waiting forever in a loop?