Process vs Thread

»

In computing, a process and a thread are fundamental units of execution, but they represent different levels of abstraction and control within an operating system.

Process

»

A process is a program in execution, with its own memory space, resources, and state.

Each process is isolated from other processes, meaning they cannot directly access each other's memory or variables.

Processes are managed by the operating system and have their own system resources (e.g., CPU, memory, file handles).

Communication between processes is typically achieved through inter-process communication (IPC) mechanisms like pipes, sockets, or message queues.

Thread

»

A thread is a subset of a process and represents a single path of execution within that process.

Threads within a process share the same memory space, allowing them to access shared variables and data structures directly.

Threads can communicate with each other by sharing memory, making communication more efficient than inter-process communication.

Threads within a process have access to the process's resources (e.g., file handles, open sockets).

In summary, a process is an independent program with its own memory and resources, while a thread is a unit of execution within a process that shares the process's memory space.

Multiple threads can exist within a single process and share data and resources, making them more lightweight compared to processes.

Here are some key differences between Processes and Threads

»

Isolation

Processes are isolated from each other, while threads share memory and resources within a process.

Communication

Processes communicate via IPC mechanisms, whereas threads can communicate through shared memory.

Overhead

Creating a new process is more resource-intensive (higher overhead) compared to creating a new thread within an existing process (lower overhead).

Fault Isolation

If a thread crashes, it can potentially crash the entire process. In contrast, if a process crashes, it doesn't affect other processes.

Scalability

Threads can be more efficient for tasks that benefit from parallelism and can scale across multiple CPU cores.

Synchronization

Threads within a process need synchronization mechanisms to coordinate access to shared resources, while processes do not have this concern.

Applications often use a combination of processes and threads to leverage both the benefits of isolation (processes) and efficient communication and resource sharing (threads).