Thread in Operating System

Within a program, a **Thread** is a separate execution path. It is a lightweight process that the operating system can schedule and run concurrently with other threads. The operating system creates and manages threads, and they share the same memory and resources as the program that created them. This enables multiple threads to collaborate and work efficiently within a single program.
A thread is a single sequence stream within a process. Threads are also called lightweight processes as they possess some of the properties of processes. Each thread belongs to exactly one process. In an operating system that supports multithreading, the process can consist of many threads. But threads can be effective only if CPU is more than 1 otherwise two threads have to context switch for that single CPU.

Why Do We Need Thread?

- Threads run in parallel improving the application performance. Each such thread has its own CPU state and stack, but they share the address space of the process and the environment.
- Threads can share common data so they do not need to use interprocess communication. Like the processes, threads also have states like ready, executing, blocked, etc.
- Priority can be assigned to the threads just like the process, and the highest priority thread is scheduled first.
- Each thread has its own Thread Control Block (TCB). Like the process, a context switch occurs for the thread, and register contents are saved in (TCB). As threads share the same address space and resources, synchronization is also required for the various activities of the thread.

Why Multi-Threading?

A thread is also known as a lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc. More advantages of multithreading are discussed below.
Multithreading is a technique used in operating systems to improve the performance and responsiveness of computer systems. Multithreading allows multiple threads (i.e., lightweight processes) to share the same resources of a single process, such as the CPU, memory, and I/O devices.
For more, refer to Multithreading in OS.

Difference Between Process and Thread

The primary difference is that threads within the same process run in a shared memory space, while processes run in separate memory spaces. Threads are not independent of one another like processes are, and as a result, threads share with other threads their code section, data section, and OS resources (like open files and

signals). But, like a process, a thread has its own program counter (PC), register set, and stack space.

Advantages of Thread

- **Responsiveness*:*** If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.
- **Faster context switch*:*** Context switch time between threads is lower compared to the process context switch. Process context switching requires more overhead from the CPU.
- **Effective utilization of multiprocessor system*:*** If we have multiple threads in a single process, then we can schedule multiple threads on multiple processors. This will make process execution faster.
- **Resource sharing*:*** Resources like code, data, and files can be shared among all threads within a process. Note: Stacks and registers can't be shared among the threads. Each thread has its own stack and registers.
- **Communication*:*** Communication between multiple threads is easier, as the threads share a common address space. while in the process we have to follow some specific communication techniques for communication between the two processes.
- **Enhanced throughput of the system*:*** If a process is divided into multiple threads, and each thread function is considered as one job, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.

Types of Threads

Threads are of two types. These are described below.

- User Level Thread
- Kernel Level Thread

User Level Threads

User Level Thread is a type of thread that is not created using system calls. The kernel has no work in the management of user-level threads. User-level threads can be easily implemented by the user. In case when user-level threads are single-handed processes, kernel-level thread manages them. Let's look at the advantages and disadvantages of User-Level Thread.

Advantages of User-Level Threads

- Implementation of the User-Level Thread is easier than Kernel Level Thread.
- Context Switch Time is less in User Level Thread.
- User-Level Thread is more efficient than Kernel-Level Thread.
- Because of the presence of only Program Counter, Register Set, and Stack Space, it has a simple representation.

Disadvantages of User-Level Threads

- There is a lack of coordination between Thread and Kernel.
- Inc case of a page fault, the whole process can be blocked.

Kernel Level Threads

A kernel Level Thread is a type of thread that can recognize the Operating system easily. Kernel Level Threads has its own thread table where it keeps track of the system. The operating System Kernel helps in managing threads. Kernel Threads have somehow longer context switching time. Kernel helps in the management of threads.

Advantages of Kernel-Level Threads

- It has up-to-date information on all threads.
- Applications that block frequency are to be handled by the Kernel-Level Threads.
- Whenever any process requires more time to process, Kernel-Level Thread provides more time to it.

Disadvantages of Kernel-Level threads

- Kernel-Level Thread is slower than User-Level Thread.
- Implementation of this type of thread is a little more complex than a user-level thread.

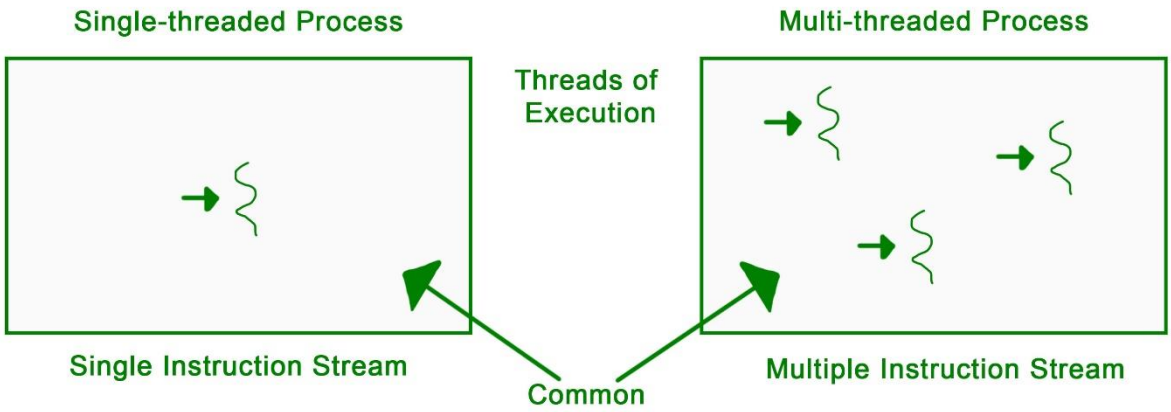
Components of Threads

These are the basic components of the Operating System.

- Stack Space
- Register Set
- Program Counter

Multithreading in Operating System

A **thread** is a path which is followed during a program's execution. Majority of programs written now a days run as a single thread. Let's say, for example a program is not capable of reading keystrokes while making drawings. These tasks cannot be executed by the program at the same time. This problem can be solved through multitasking so that two or more tasks can be executed simultaneously. Multitasking is of two types: Processor based and thread based. Processor based multitasking is totally managed by the OS, however multitasking through multithreading can be controlled by the programmer to some extent. The concept of **multi-threading** needs proper understanding of these two terms – **a process and a thread**. A process is a program being executed. A process can be further divided into independent units known as threads. A thread is like a small light-weight process within a process. Or we can say a collection of threads is what is known as a process.



Single Thread and Multi Thread Process

Applications – Threading is used widely in almost every field. Most widely it is seen over the internet nowadays where we are using transaction processing of every type like recharges, online transfer, banking etc. Threading is a segment which divide the code into small parts that are of very light weight and has less burden on CPU memory so that it can be easily worked out and can achieve goal in desired field. The concept of threading is designed due to the problem of fast and regular changes in technology and less the work in different areas due to less application. Then as says "need is the generation of creation or innovation" hence by following this approach human mind develop the concept of thread to enhance the capability of programming.

Lifecycle of a thread

There are various stages in the lifecycle of a thread. Following are the stages a thread goes through in its whole life.

- **New:** The lifecycle of a born thread (new thread) starts in this state. It remains in this state till a program starts.
- **Runnable**: A thread becomes runnable after it starts. It is considered to be executing the task given to it.
- **Waiting**: While waiting for another thread to perform a task, the currently running thread goes into the waiting state and then transitions back again after receiving a signal from the other thread.
- **Timed Waiting:** A runnable thread enters into this state for a specific time interval and then transitions back when the time interval expires or the event the thread was waiting for occurs.
- **Terminated (Dead)**: A thread enters into this state after completing its task.

Types of execution in OS

There are two types of execution:

1. **Concurrent Execution:** This occurs when a processor is successful in switching resources between threads in a multithreaded process on a single processor.

2. **Parallel Execution:** This occurs when every thread in the process runs on a separate processor at the same time and in the same multithreaded process

Drawbacks of Multithreading

Multithreading is complex and many times difficult to handle. It has a few drawbacks. These are:

- If you don't make use of the locking mechanisms properly, while investigating data access issues there is a chance of problems arising like data inconsistency and dead-lock.
- If many threads try to access the same data, then there is a chance that the situation of thread starvation may arise. Resource contention issues are another problem that can trouble the user.
- Display issues may occur if threads lack coordination when displaying data.

Benefits of Multithreading:

- Multithreading can improve the performance and efficiency of a program by utilizing the available CPU resources more effectively. Executing multiple threads concurrently, it can take advantage of parallelism and reduce overall execution time.
- Multithreading can enhance responsiveness in applications that involve user interaction. By separating time-consuming tasks from the main thread, the user interface can remain responsive and not freeze or become unresponsive.
- Multithreading can enable better resource utilization. For example, in a server application, multiple threads can handle incoming client requests simultaneously, allowing the server to serve more clients concurrently.
- Multithreading can facilitate better code organization and modularity by dividing complex tasks into smaller, manageable units of execution. Each thread can handle a specific part of the task, making the code easier to understand and maintain.