Threads

What is a Thread?

A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

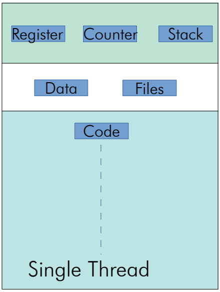
A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.

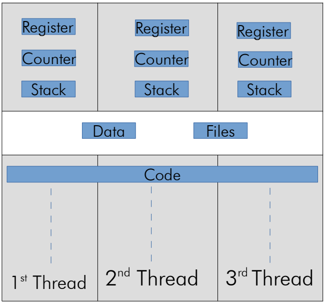
A thread is also called **a lightweight process**. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.

Threads

- Single Thread
- Multi Thread





Left Side : A single Process With Single Thread
Right Side : A single Process With 3 Threads (Multi Threads)

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

Types of Thread

Threads are implemented in following two ways:

- **User Level Threads** -- User managed threads.
- **Kernel Level Threads** -- Operating System managed threads acting on kernel, an operating system core.

User Level Threads : In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.

Kernel Level Threads : In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

Multithreading Models

Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types :

1. Many-to-many relationship
2. Many-to-one relationship
3. One-to-one relationship

Difference Between Process and Thread

Process	Thread
Process is heavy weight or resource intensive.	Thread is lightweight, taking lesser resources than a process.
Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.