Threads in Operating System

A thread is the smallest unit of execution which has its own thread ID, program counter, register set and stack.

All the threads that belong to the same process share the code, data section and other resources belonging to the process.

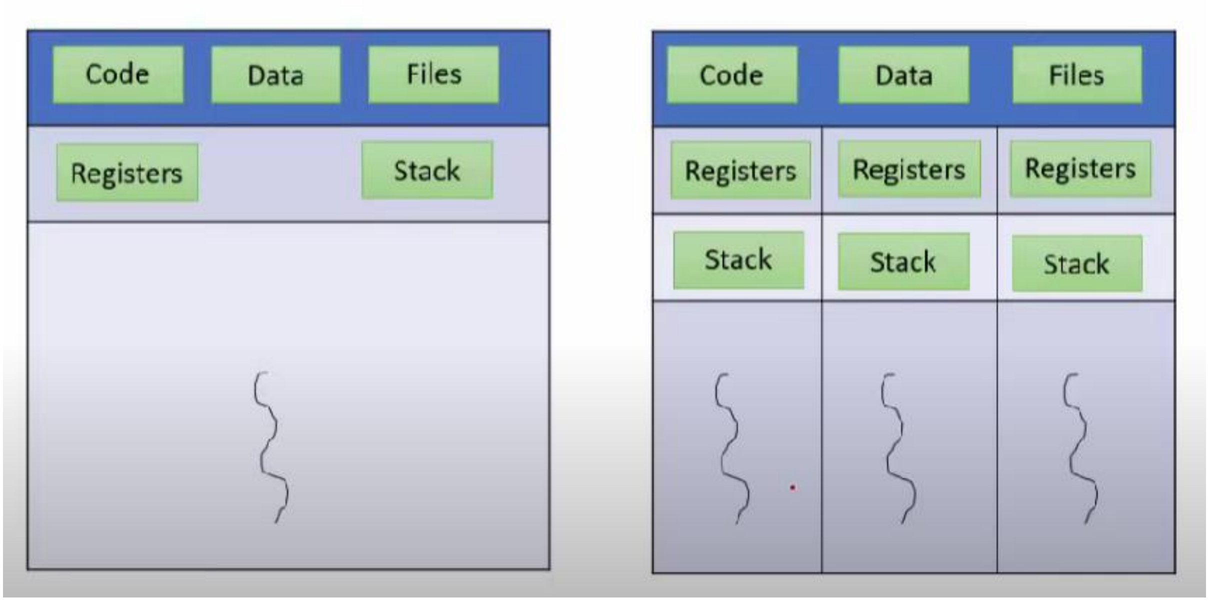
Thread is a part of the process.

A thread cannot exist without a process.

A thread is also called a lightweight process.

Multithreading is a phenomenon of executing multiple threads at the same time.

Single Threaded Process ***Multi-Threaded Process***



Benefits of creating threads in Operating System

1. **Responsiveness** – Multi-threading increases the responsiveness of the process. For example, in MSWord while one thread does the spelling check the other thread allows you to keep tying the input. Therefore, you feel that Word is always responding.
2. **Resource sharing** – All the threads share the code and data of the process. Therefore, this allows several threads to exist within the same address space
3. **Economy** –It is convenient to create threads. Since they share resources, they are less costly

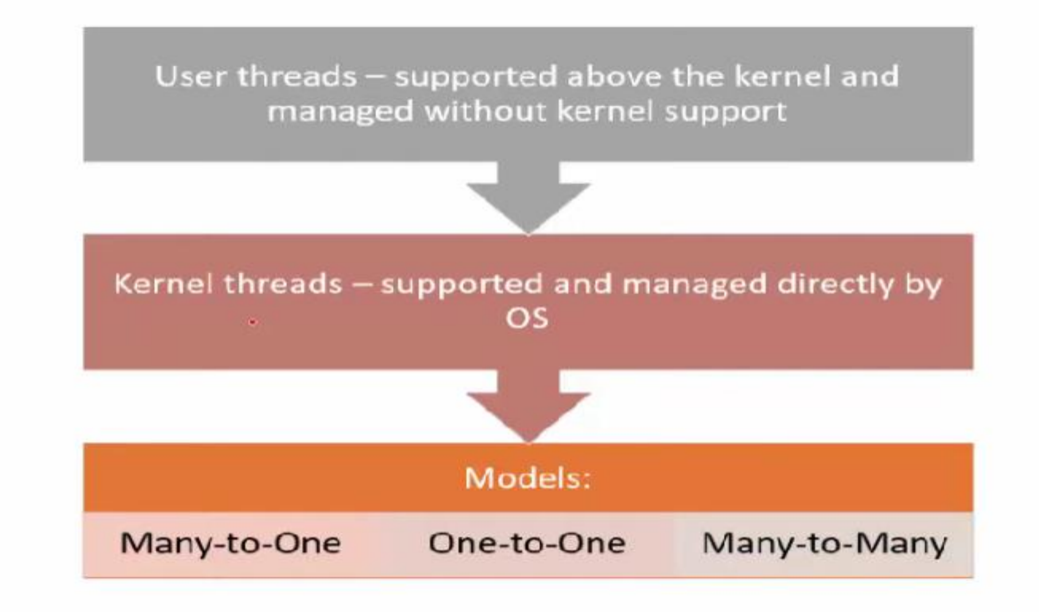
Types Of Threads

There are two kinds of threads in the system – *user threads* and *kernel threads.*

User threads are supported above the kernel and managed without kernel support

Kernel threads are supported and managed directly by OS.

Ultimately, a relationship must exist between user threads and kernel threads. In this section, we look at three common ways of establishing such a relationship.



User threads – supported above the kernel and managed without kernel support

Kernel threads – supported and managed directly by OS

Models:

Many-to-One

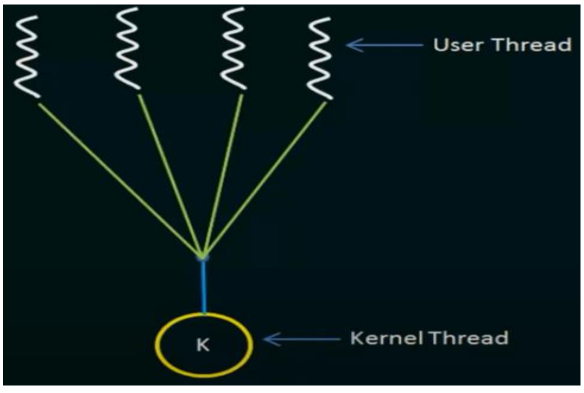
One-to-One

Many-to-Many

Multi-threading Models

Many to One Model

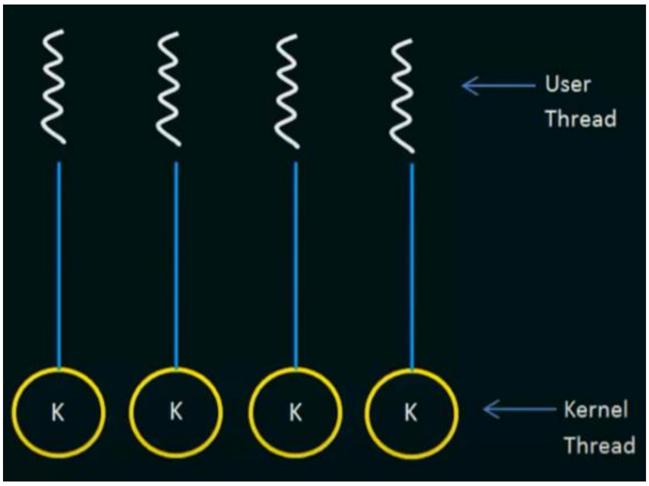
1. Many user-level threads mapped to a single kernel thread
2. Thread management is done in user space
3. Drawback: The Entire process will block if a thread makes a blocking system call



One to One Model

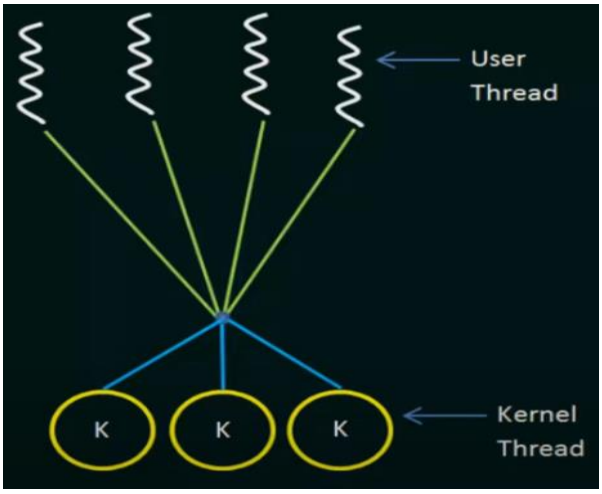
1. Each user-level thread maps to kernel thread
2. Provides more concurrency than many-to-many model by allowing another thread to run when a thread makes a blocking system call.
3. Drawback: 1. creating a user thread requires creating a corresponding kernel thread. (Time consuming)

   2. Because the overhead of creating kernel threads can burden the performance of an application, most implementations of this model restrict the number of kernel threads supported by the system.



Many to Many Model

1. Allows many user-level threads to be mapped to smaller or equal number of kernel threads.
2. The number of kernel threads may be specific to a particular application.
3. Also when a thread makes a blocking system call, the kernel can schedule another thread for execution.



Difference between process and thread in OS

Parameter	Process	Thread
Definition	Process means a program is in execution.	Thread means a segment of a process.
Lightweight	The process is not Lightweight.	Threads are Lightweight.
Termination time	The process takes more time to terminate.	The thread takes less time to terminate.
Creation time	It takes more time for creation.	It takes less time for creation.
Communication	Communication between processes needs more time compared to thread.	Communication between threads requires less time compared to processes.
Context switching time	It takes more time for context switching.	It takes less time for context switching.
Resource	Process consume more resources.	Thread consume fewer resources.
Memory	The process is mostly isolated.	Threads share memory.
Sharing	It does not share data	Threads share data with each other.

Difference between User Level thread and Kernel Level thread

S. No.	Parameters	User Level Thread	Kernel Level Thread
1.	**Implemented by**	User threads are implemented by users.	Kernel threads are implemented by Operating System (OS).
2.	**Recognize**	Operating System doesn’t recognize user level threads.	Kernel threads are recognized by Operating System.
3.	**Implementation**	Implementation of User threads is easy.	Implementation of Kernel thread is complicated.
4.	**Context switch time**	Context switch time is less.	Context switch time is more.

S. No.	Parameters	User Level Thread	Kernel Level Thread
5.	**Hardware support**	Context switch requires no hardware support.	Hardware support is needed.
6.	**Blocking operation**	If one user level thread performs blocking operation then entire process will be blocked.	If one kernel thread performs blocking operation then another thread can continue execution.
7.	**Creation and Management**	User level threads can be created and managed more quickly.	Kernel level threads take more time to create and manage.
8.	**Operating System**	Any operating system can support user-level threads.	Kernel level threads are operating system-specific.
Example		Example: Java thread, POSIX threads.	Example: Window Solaris.