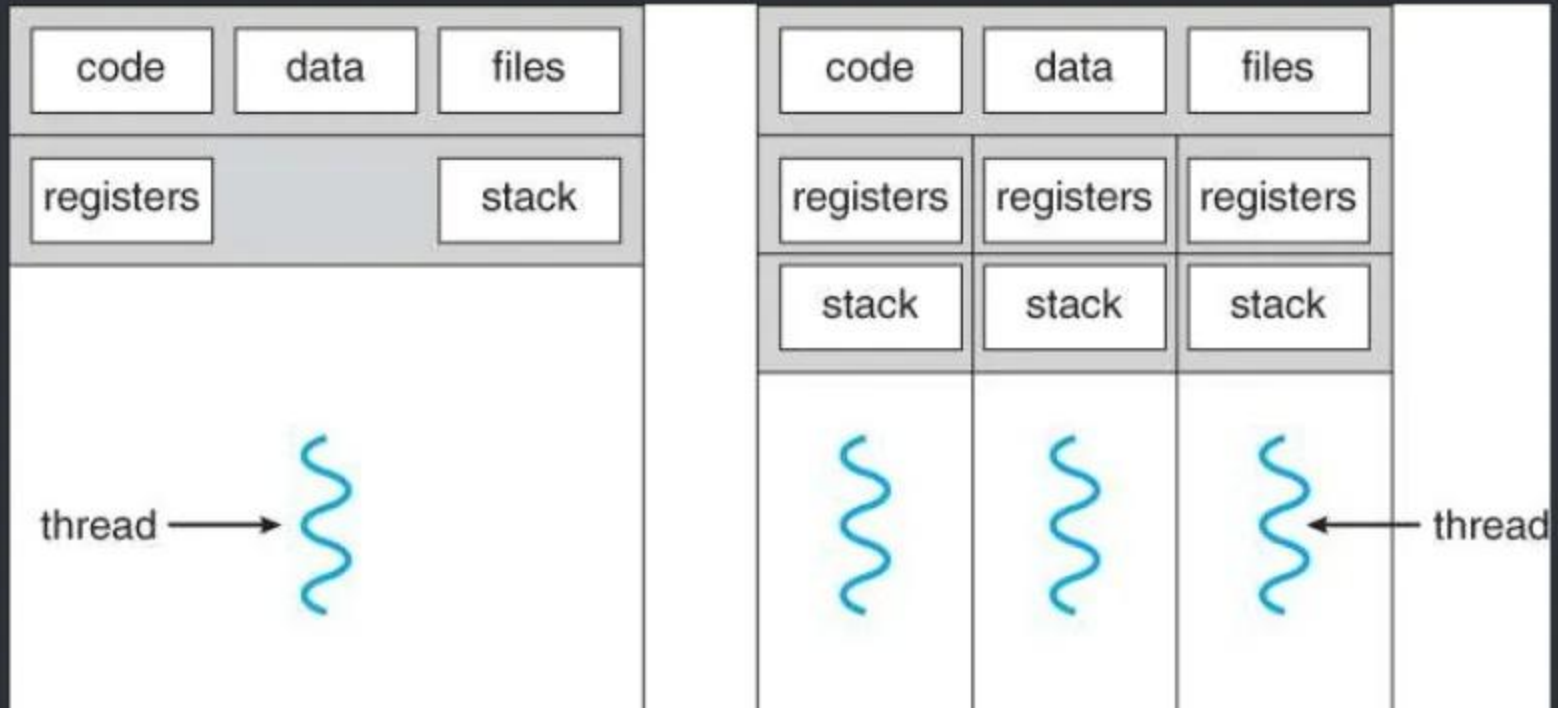# THREADS
(OPERATING SYSTEM)

What is Thread ?

A thread is a flow of execution through the process code, with its own program counter, system registers and stack. A thread is also called a light weight process. Threads provide a way to improve application performance through parallelism.

Each thread belongs to exactly one process and no thread can exist outside a process.

Single threaded Process and Multi-threaded Process

Difference between Process and Threads :

S.N.	Process	Threads
1.	Process is heavy weight or resource intensive.	Thread is light weight taking lesser resources than a process.
2.	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3.	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.
4.	If one process is blocked then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, second thread in the same task can run.
5.	Multiple processes without using	Multiple threaded processes use

Advantages of Threads :

- Thread minimizes context switching time.

- Use of threads provides concurrency within a process.

- Efficient communication.

- Economy- It is more economical to create and context switch threads.

- Utilization of multiprocessor architectures to a greater scale and efficiency.

Disadvantages of Threads :

- Blocking of parent threads will stop all child thread.

- Security.

Types of Threads :

Threads are implemented in following two ways :

- User Level Threads (ULT)

- Kernel Level Threads (KLT)

User Level Threads (ULT) :

   User level thread implement in user level libraries, so thread switching does not need to call operating system and to cause interrupt to the kernel.

   In fact, the kernel knows nothing about user level threads and manages them as if they were single-threaded process.

Advantages of ULT :

- User level threads does not require modification to operating system.

- Easy to represent and manage.

- User level thread can run on any operating system.

- User level threads are fast and efficient.

Disadvantages of ULT :

- There is a lack of coordination between threads and operating system kernel.

Kernel Level Threads (KLT) :

In this method, the kernel knows about and manages the threads. No runtime system is needed in this case. Operating system kernel provides system call to create and manage threads.

Advantages of KLT :

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.

- If one thread in a process is blocked the kernel can schedule another thread of the same process.

- Kernel routines themselves can multithreaded.

Disadvantages of KLT :

- Kernel thread are generally slower to create and manage than the user threads.

Difference between ULT and KLT :

S.N.	User Level Thread	Kernel Level Thread
1.	User level thread are faster to create and manage.	Kernel level thread are slower to create and manage.
2.	Implementation is by a thread library at the user level.	Operating system supports creation of kernel level threads.
3.	ULT is generic and can run on any operating system.	KLT is specific to the operating system.
4.	Multi-Threaded can not take	Kernel routines themselves can be

Multi-Threading Models :

Some operating system provides a combined user level thread and kernel level thread facility. Solaris is a good example of this combined approch.
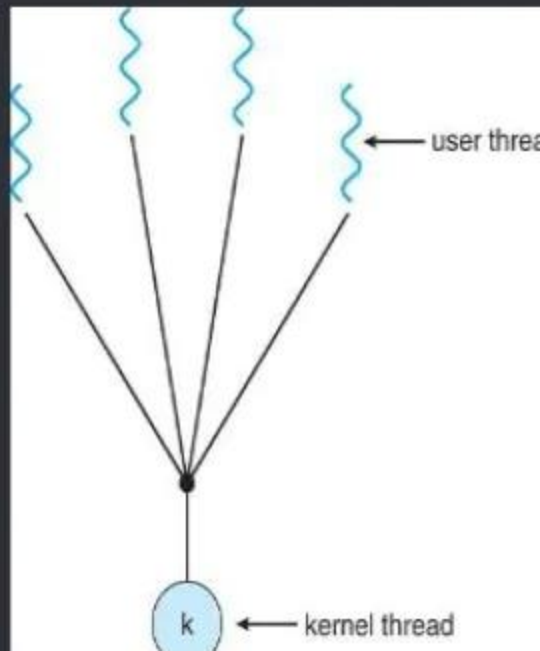
Multi-Threading models are three types.

- Many-to-One Model
- One-to-One Model
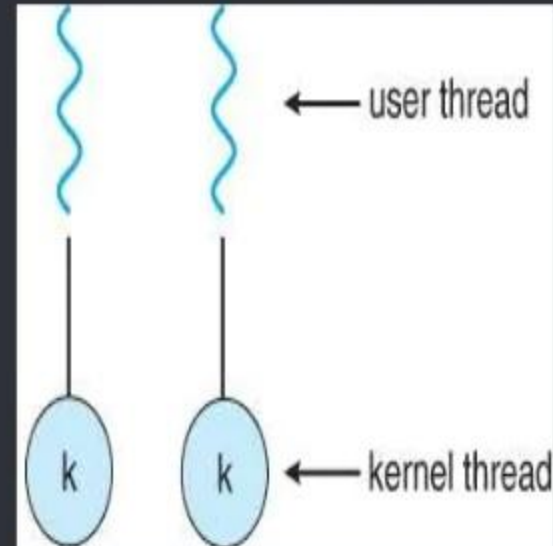- Many-to-Many Model

Many-to-One Model :

Implementation of the many-to-one model (many user level thread to one kernel level thread) allow the application to create any number of threads that can execute concurrently. In this implementation, all threads activity is restricted to user space.

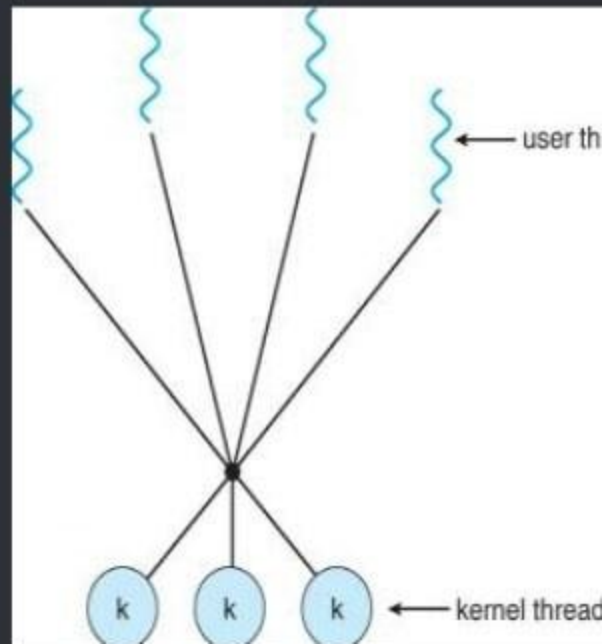Additionally only one thread can access the kernel at a time, so only

One-to-One Model :

The one-to-one model (one user level thread to one kernel level thread) is among the earliest implementations of true multithreading. In this implementation, each ULT created by the application is known to the kernel,and all threads ca access the kernel at the same time.

Many-to-Many Model :

The many-to-many model (many user level thread to many kernel level thread) avoids many of the limitations of the one-to-one model, while extending multithreading capabilities even further. The many-to-many model also called two-level model, minimize programing effort while reducing the cost and weight of each thread.

user th ←

k   k   k   ← kernel thread

THANK YOU !