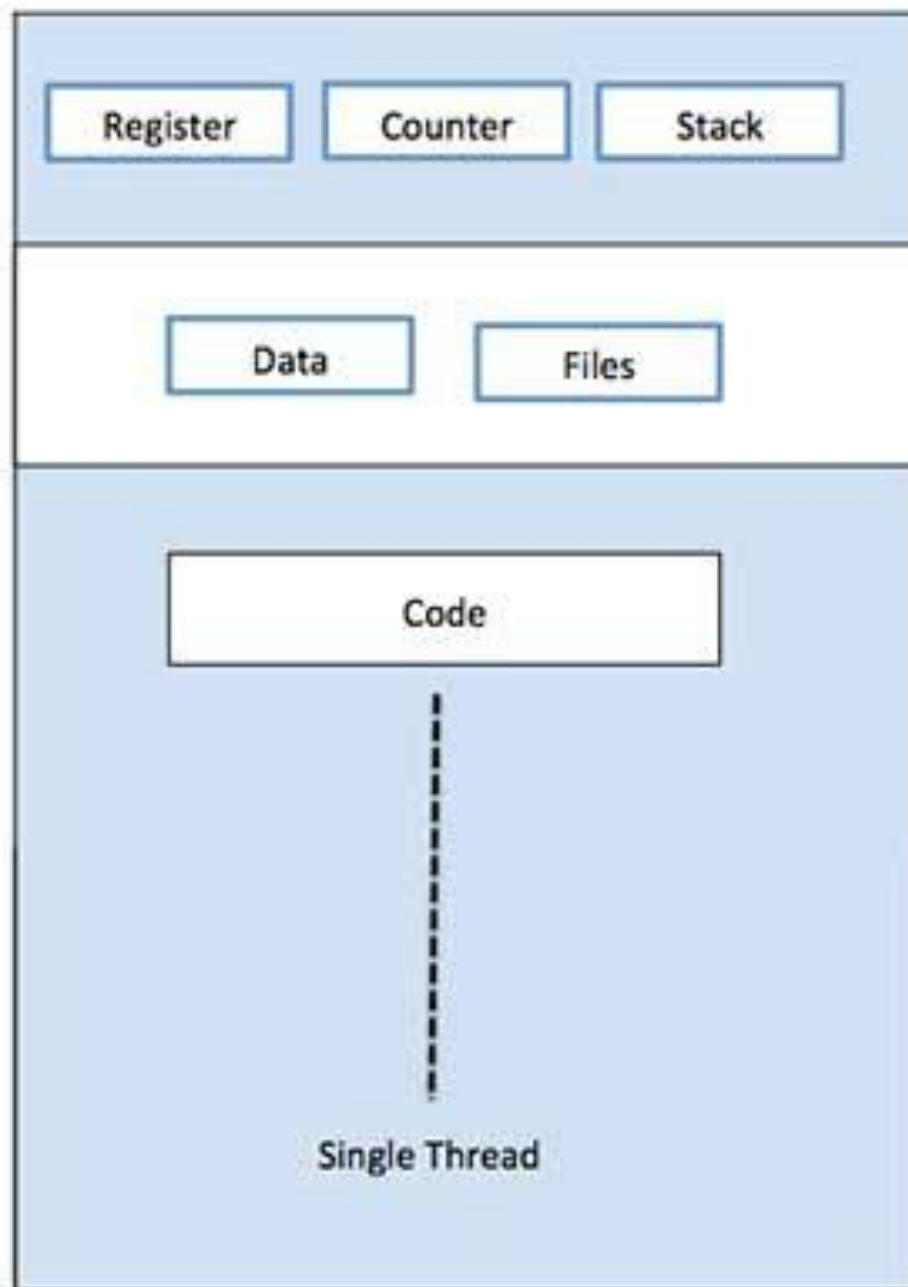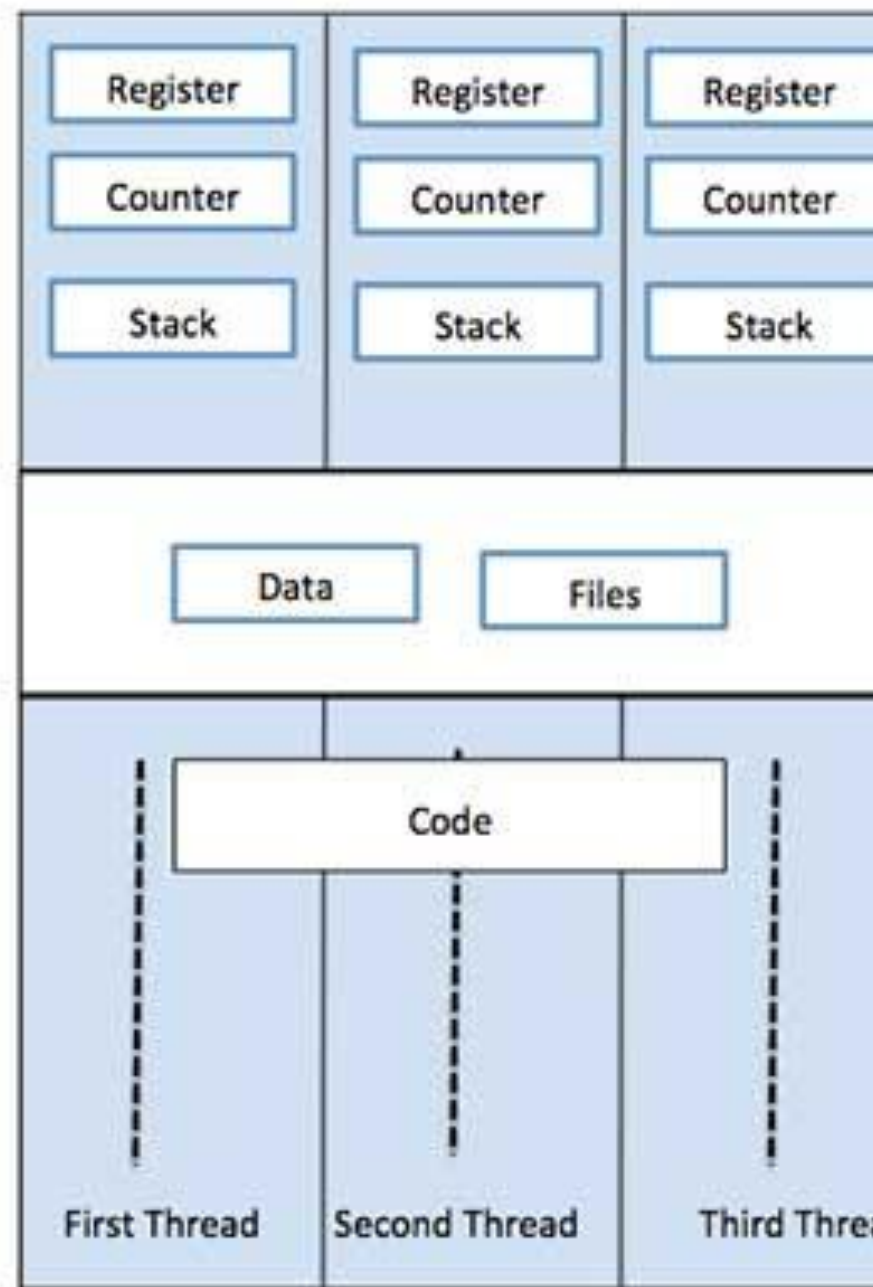## What is Thread?

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 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

| Register | Counter | Stack |
| --- | --- | --- |

| Data | Files |
| --- | --- |

Code

Single Thread

Single Process P with single thread

| Register | Register | Register |
| --- | --- | --- |
| Counter | Counter | Counter |
| Stack | Stack | Stack |

| Data | Files |
| --- | --- |

Code

| First Thread | Second Thread | Third Three |
| --- | --- | --- |

Single Process P with three threads

# Difference between Process and Thread

| | | |
|---|---|---|
| 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, a second thread in the same task can run. |
| 5 | Multiple processes without using threads use more resources. | Multiple threaded processes use fewer resources. |
| 6 | In multiple processes each process operates independently of the others. | One thread can read, write or change another thread's data. |

**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.
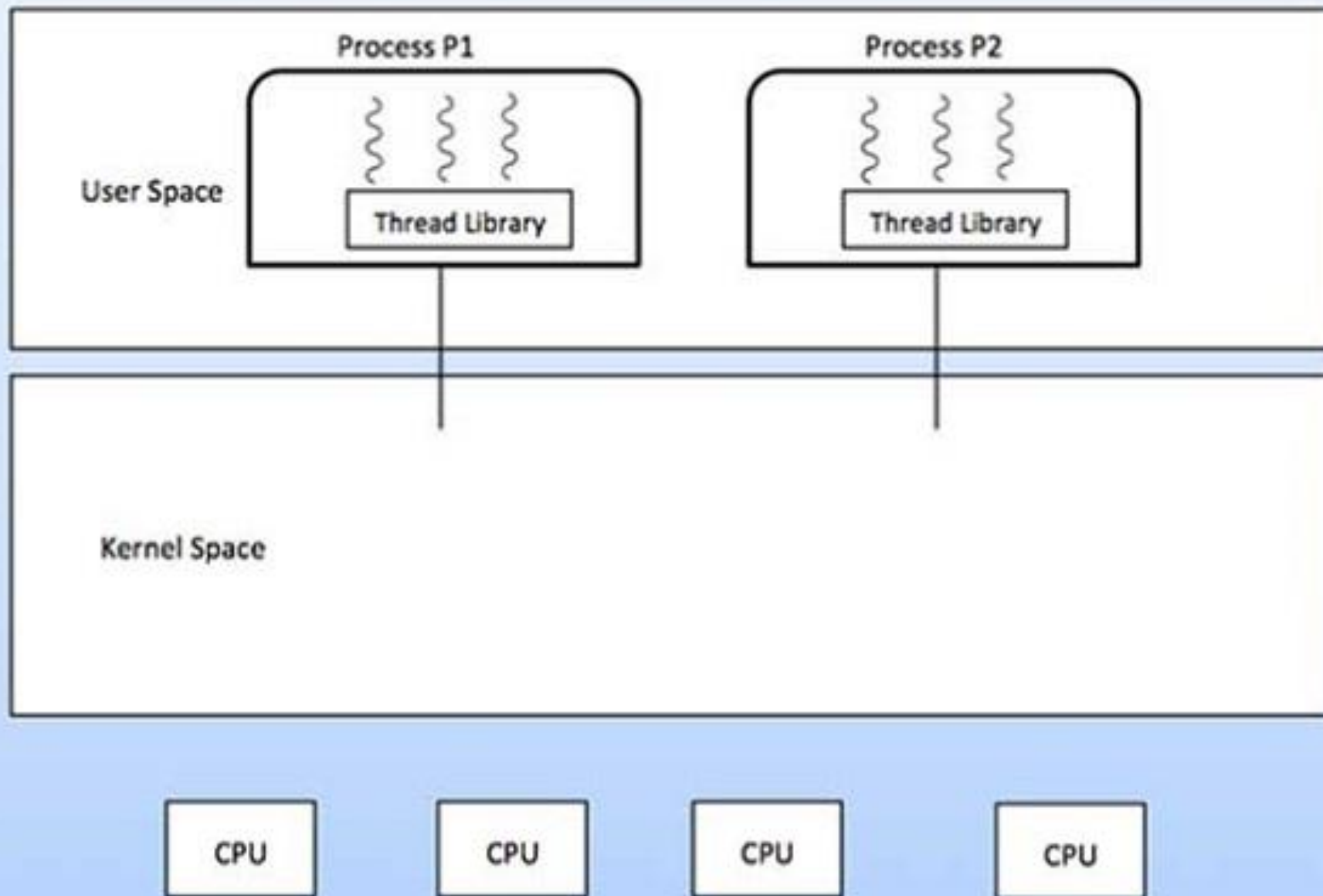
**Advantages**

•Thread switching does not require Kernel mode privileges.

•User level thread can run on any operating system.

•Scheduling can be application specific in the user level thread.

•User level threads are fast to create and manage.

•

**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.

The Kernel maintains context information for the process as a whole and for individuals threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

**Advantages**

•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 be multithreaded.

**Disadvantages**

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

•Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

# Difference between User-Level & Kernel-Level Thread

| S.N. | User-Level Threads | Kernel-Level Thread |
|------|--------------------|---------------------|
| 1 | User-level threads are faster to create and manage. | Kernel-level threads are slower to create and manage. |
| 2 | Implementation is by a thread library at the user level. | Operating system supports creation of Kernel threads. |
| 3 | User-level thread is generic and can run on any operating system. | Kernel-level thread is specific to the operating system. |
| 4 | Multi-threaded applications cannot take advantage of multiprocessing. | Kernel routines themselves can be multithreaded. |