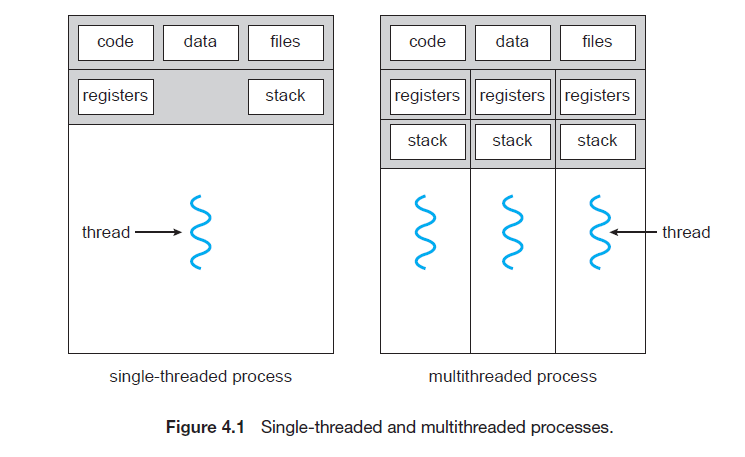
**OVERVIEW OF THREADS**

A thread is a basic unit of CPU utilization. A Thread also called lightweight process. A thread is a part of process which has thread ID and its own program counter, a stack and a set of registers that reside in a [process](https://afteracademy.com/blog/what-is-a-process-in-operating-system-and-what-are-the-different-states-of-a-process). Threads can’t exist outside any process. Also, each thread belongs to exactly one process. All the threads that belong to the same process share the code section, data section and other operating system resources like open files belonging to the process and signals.

A traditional (or heavyweight) process has a single thread of control. If a process has multiple threads of control, it can perform more than one task at a time.



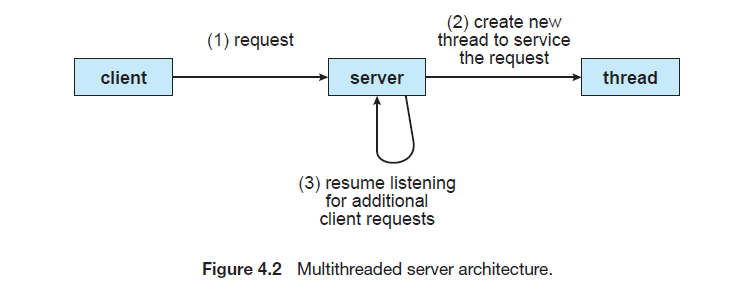
The diagram above shows the single-threaded process and the multi-threaded process. A single-threaded process is a process with a single thread. A multi-threaded process is a process with multiple threads. As the diagram clearly shows that the multiple threads in it have its own registers, stack but they share the code and data segment.

An application typically is implemented as a separate process with several threads of control.

Ex: A word processor may have a thread for displaying graphics, another thread for responding to keystrokes from the user, and a third thread for performing spelling and grammar checking in the background.

In certain situations, a single application may be required to perform several similar tasks. For example, a web server accepts client requests for web pages, images, sound, and so forth. A busy web server may have several (perhaps thousands of) clients concurrently accessing it. If the web server ran as a traditional single-threaded process, it would be able to service only one client at a time, and a client might have to wait a very long time for its request to be serviced.

One solution is to have the server run as a single process that accepts requests. When the server receives a request, it creates a separate process to service that request. In fact, this process-creation method was in common use before threads became popular. Process creation is time consuming and resource intensive, however. It is generally more efficient to use one process that contains multiple threads. If the web-server process is multithreaded, the server will create a separate thread that listens for client requests. When a request is made, rather than creating another process, the server creates a new thread to service the request and resume listening for additional requests. This is illustrated in below Figure.



When a server receives a message, it services the message using a separate thread. This allows the server to service several concurrent requests.

Finally, most operating-system kernels are now multithreaded. Several threads operate in the kernel, and each thread performs a specific task, such as managing devices, managing memory, or interrupt handling.

**Benefits of Multi-threaded processes:**

The benefits of multithreaded programming can be broken down into four major categories:

**1.Responsiveness:** Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user. This quality is especially useful in designing user interfaces.

For instance, consider what happens when a user clicks a button that results in the performance of a time-consuming operation. A single-threaded application would be unresponsive to the user until the operation had completed. In contrast, if the time-consuming operation is performed in a separate thread, the application remains responsive to the user.

**2.Resource Sharing:** Processes can only share resources through techniques such as shared memory and message passing. Such techniques must be explicitly arranged by the programmer. However, threads share the memory and the resources of the process to which they belong by default. The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.

**3.Economy:** Allocating memory and resources for process creation is costly. Because threads share the resources of the process to which they belong, it is more economical to create and context-switch threads. Empirically gauging the difference in overhead can be difficult, but in general it is significantly more time consuming to create and manage processes than threads. In Solaris, for example, creating a process is about thirty times slower than is creating a thread, and context switching is about five times slower.

**4.Scalability or Utilization of multiprocessor architecture:**

The benefits of multithreading can be even greater in a multiprocessor architecture, where threads may be running in parallel on different processing cores. A single-threaded process can run on only one processor, regardless how many are available. Multithreading on a multi-CPU machine increases concurrency.