**Process:**

* An executing instance of a program is called a process.
* Some operating systems use the term ‘task‘ to refer to a program that is being executed.
* A process is always stored in the main memory also termed as the primary memory or random access memory.
* Therefore, a process is termed as an active entity. It disappears if the machine is rebooted.
* Several process may be associated with a same program.
* On a multiprocessor system, multiple processes can be executed in parallel.
* On a uni-processor system, though true parallelism is not achieved, a process scheduling algorithm is applied and the processor is scheduled to execute each process one at a time yielding an illusion of concurrency.
* **Example:** Executing multiple instances of the ‘Calculator’ program. Each of the instances are termed as a process.

**Thread:**

* A thread is a subset of the process.
* It is termed as a ‘lightweight process’, since it is similar to a real process but executes within the context of a process and shares the same resources allotted to the process by the kernel.
* Usually, a process has only one thread of control – one set of machine instructions executing at a time.
* A process may also be made up of multiple threads of execution that execute instructions concurrently.
* Multiple threads of control can exploit the true parallelism possible on multiprocessor systems.
* On a uni-processor system, a thread scheduling algorithm is applied and the processor is scheduled to run each thread one at a time.
* All the threads running within a process share the same address space, file descriptors, stack and other process related attributes.
* Since the threads of a process share the same memory, synchronizing the access to the shared data within the process gains unprecedented importance.
* Every process is a thread (primary thread).
* But every thread is not a process. It is a part(entity) of a process.

17

**Process:**

1. Process is a heavy weight process.
2. Process is a separate program that has separate memory,data,resources ect.
3. Context switch between the process is time consuming.

Example:   
Say, opening any browser (mozilla, Chrome, IE). At this point new process will start to execute.

**Threads:**

1. Threads are light weight processes.Threads are bundled inside the process.
2. Threads have a shared memory,data,resources,files etc.
3. Context switch between the threads are not much time consuming as Process.

Example:   
Opening multiple tabs in the browser.

**The difference between foreground and background threads**

In .NET, there are two types of threads:

Foreground

background.

Threads are, by default, created as foreground threads. You could set the thread state explicitly as background. The difference is that foreground threads keep an application alive as long as they are running, while background threads do not.

In other words, when you close applications, all background threads are automatically terminated.

So, when all the foreground threads complete execution, the application can exit before the background threads return. After all the foreground threads have been stopped, or after the application exits, the system stops all background threads.

# C# Thread Life Cycle

In C#, each thread has a life cycle. The life cycle of a thread is started when instance of System.Threading.Thread class is created. When the task execution of the thread is completed, its life cycle is ended.

There are following states in the life cycle of a Thread in C#.

* Unstarted
* Runnable (Ready to run)
* Running
* Not Runnable
* Dead (Terminated)

## Unstarted State

When the instance of Thread class is created, it is in unstarted state by default.

## Runnable State

When start() method on the thread is called, it is in runnable or ready to run state.

## Running State

Only one thread within a process can be executed at a time. At the time of execution, thread is in running state.

## Not Runnable State

The thread is in not runnable state, if sleep() or wait() method is called on the thread, or input/output operation is blocked.

## Dead State

After completing the task, thread enters into dead or terminated state.