Multithreading

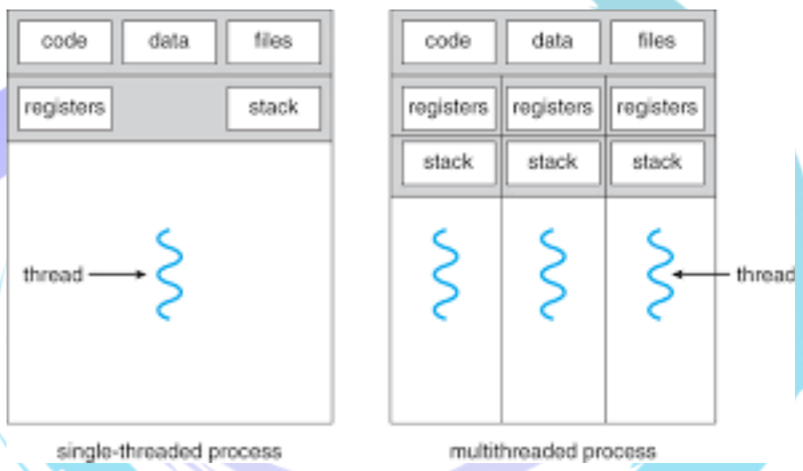
- Executing multiple tasks at a time is called as Multitasking.
- There are two ways in which we can perform multitasking as
    - MultiProcessing
    - MultiThreading
- In case of MultiProcessing to perform multiple tasks separate process gets created which executes simultaneously.
- In case of MultiThreading to perform multiple tasks separate thread gets created which executes simultaneously.
- In computer architecture, multithreading is the ability of a central processing unit (CPU) to execute multiple threads concurrently, supported by the operating system.
- This approach differs from multiprocessing. In a multithreaded application, the processes and threads share the resources of a single or multiple cores.
- Threading enables our C# program to perform concurrent processing so that we can do more than one operation at a time.

- Threads have the following properties:
    - Threads enable our program to perform concurrent processing.
    - The .NET Framework System.Threading namespace makes using threads easier.
    - Threads share the application's resources.

- By default, a C# program has one thread. However, auxiliary threads can be created and used to execute code in parallel with the primary thread. These threads are often called worker threads.
- Worker threads can be used to perform time-consuming or time-critical tasks without tying up the primary thread.
- For example, worker threads are often used in server applications to fulfill incoming requests without waiting for the previous request to be completed.
- Worker threads are also used to perform "background" tasks in desktop applications so that the main thread which drives user interface elements remains responsive to user actions.
- Threading solves problems with throughput and responsiveness, but it can also introduce resource-sharing issues such as deadlocks and race conditions.
- Multiple threads are best for tasks that require different resources such as file handles and network connections.

- Assigning multiple threads to a single resource is likely to cause synchronisation issues, and having threads frequently blocked when waiting for other threads defeats the purpose of using multiple threads.
- A common strategy is to use worker threads to perform time-consuming or time-critical tasks that do not require many of the resources used by other threads. Naturally, some resources in our program must be accessed by multiple threads.
- For these cases, the System.Threading namespace provides classes for synchronizing threads.

Consider below diagram which is used to demonstrate difference between Single threaded application and Multithreaded application.



Some characteristics of Multithreaded application

1. Every thread is the part of same process address space.
2. Every thread has its own stack.
3. Every thread has its own ID.
4. Every thread has its own CPU quantum.
5. Two sibling threads can communicate with each other directly.
6. Parent thread can kill its child thread.
7. Operating system can kill any thread at any time without intimation.
8. Every thread has its own priority.
9. We can not predict the execution sequence of threads.