C# Multithreading

INSTRUCTOR: ADEEN BIN WAHEED

LECTURE 15

Delegators

C# delegates are similar to pointers to functions, in C or C++. A delegate is a reference type variable that holds the reference to a method. The reference can be changed at runtime.

Delegates are especially used for implementing events and the call-back methods. All delegates are implicitly derived from the System.Delegate class.

Declaring a delegator:

Delegate declaration determines the methods that can be referenced by the delegate. A delegate can refer to a method, which has the same signature as that of the delegate.

For example, consider a delegate:

```
public delegate int MyDelegate (string s);
```

Delegators

Running Demo

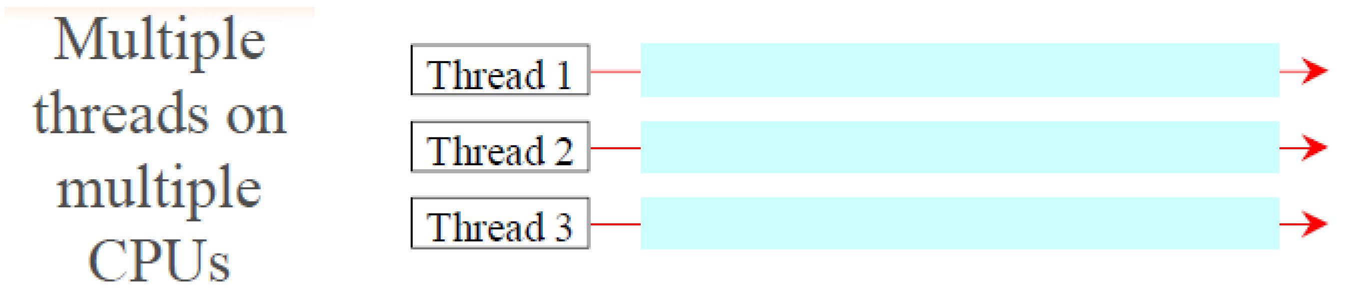
Thread

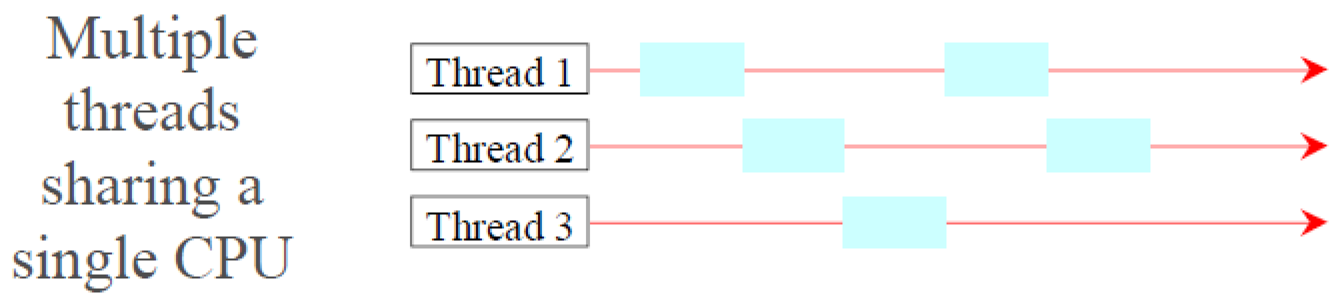
- Thread is a single sequential flow of control within a program.
- Single-threaded program can handle one task at any time.
- Multitasking allows single processor to run several concurrent threads.
- Most modern operating systems support multitasking.

Advantages of Multithreading

- Reactive systems –constantly monitoring
- Server can handle multiple clients simultaneously.
- Can take advantage of parallel processing.
- Different processes share memory space.
- A thread can execute concurrently with other threads within a single process.
- All threads managed by the OS share memory space and can communicate with each other.

Threads Concept





Thread Life Cycle

The life cycle of a thread starts when an object of the System.Threading. Thread class is created and ends when the thread is terminated or completes execution.

Following are the various states in the life cycle of a thread:

The Unstarted State: It is the situation when the instance of the thread is created but the Start method is not called.

The Ready State: It is the situation when the thread is ready to run and waiting CPU cycle.

The Not Runnable State: A thread is not executable, when:

Sleep method has been called

Wait method has been called

Blocked by I/O operations

The Dead State: It is the situation when the thread completes execution or is aborted.

The Main Thread

In C#, the System.Threading.Thread class is used for working with threads. It allows creating and accessing individual threads in a multithreaded application. The first thread to be executed in a process is called the main thread.

- When a C# program starts execution, the main thread is automatically created.
- The threads created using the Thread class are called the child threads of the main thread.
- You can access a thread using the **CurrentThread** property of the Thread class.

The Main Thread

Running Demo

Properties of the Thread Class

Property	**Description**
CurrentCulture	Gets or sets the culture for the current thread.
CurrentThread	Gets the currently running thread.
CurrentUICulture	Gets or sets the current culture used by the Resource Manager to look up culture-specific resources at run-time.
ExecutionContext	Gets an ExecutionContext object that contains information about the various contexts of the current thread.
IsAlive	Gets a value indicating the execution status of the current thread.
IsBackground	Gets or sets a value indicating whether or not a thread is a background thread.
IsThreadPoolThread	Gets a value indicating whether or not a thread belongs to the managed thread pool.
ManagedThreadId	Gets a unique identifier for the current managed thread.
Name	Gets or sets the name of the thread.
Priority	Gets or sets a value indicating the scheduling priority of a thread.
ThreadState	Gets a value containing the states of the current thread.

Creating Threads

Threads are created by extending the Thread class. The extended Thread class then calls the **Start()** method to begin the child thread execution.

Start Function:

public void Start()Starts a thread.

Creating Threads

Running Demo