Day 41/60 - Understanding Lock vs Semaphore in C#

When dealing with multithreading, synchronization mechanisms like lock and Semaphore help manage concurrent access to shared resources.

lock (Exclusive Access for One Thread)
- Ensures only one thread enters the critical section.
- Blocks other threads until the lock is released.
- Works best for short, lightweight operations.

Example:

```
private static readonly object _lockObj = new object();

void CriticalSection()
{
 lock (_lockObj)
 {
 Console.WriteLine("Thread-safe execution");
 }
}
```

Semaphore (Allows Multiple Threads)
- Controls a limited number of concurrent threads.
- Useful for managing connection pools or rate-limiting requests.

Example:

```
private static SemaphoreSlim _semaphore = new SemaphoreSlim(2); // Allows 2 threads at a time

async Task AccessResource()
{
 await _semaphore.WaitAsync();
 try
 {
 Console.WriteLine("Thread entered");
 await Task.Delay(1000); // Simulating work
 }
 finally
 {
 _semaphore.Release();
 }
}
```

When to Use?
Use lock for exclusive access when only one thread should modify data at a time.
Use SemaphoreSlim when you need controlled concurrency with a defined limit.

#dotnet #csharp #multithreading #lock #semaphore