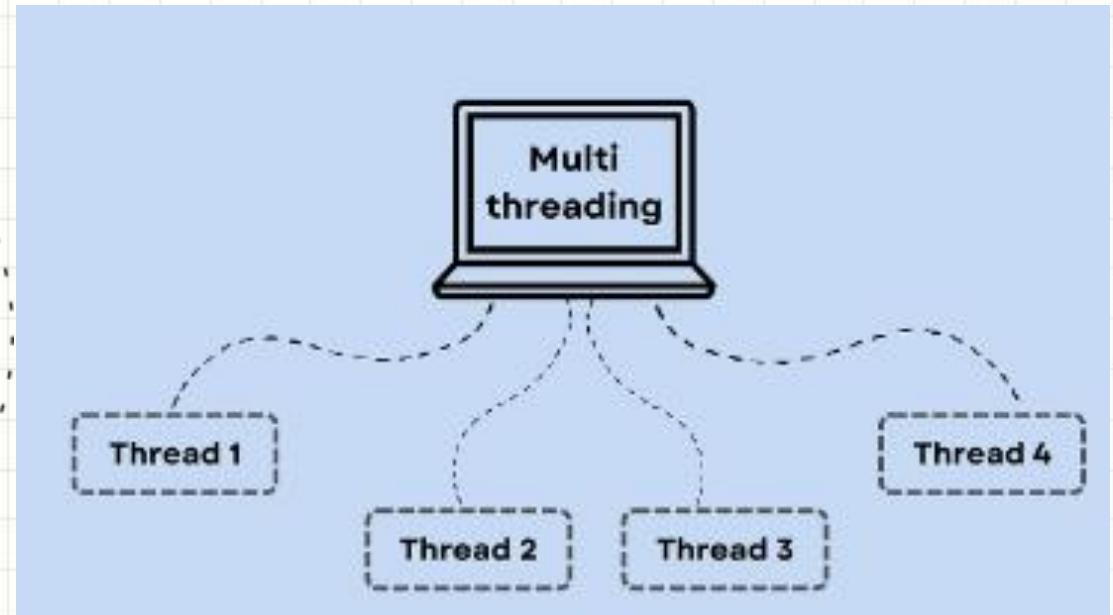


# LẬP TRÌNH ĐA LUỒNG

## (PHẦN 2)



# Nội dung

- 1** Tổng quan về tiến trình, luồng và đa luồng
- 2** Tạo và quản lý luồng trong C#
- 3** Một số giải pháp tránh xung đột
- 4** Sử dụng đa luồng trong ứng dụng WinForms

# Một số giải pháp tránh xung đột

Trong đa luồng, nhiều luồng cùng truy cập và thay đổi dữ liệu chung dễ dẫn đến:

- Race condition (tranh chấp)
- Deadlock
- Lỗi đọc/ghi sai
- Lỗi tăng/giảm biến không chính xác

Ví dụ:

```
int counter = 0;  
Parallel.For(0, 1000000, i =>  
{  
    counter++;  
});  
Console.WriteLine(counter);
```

Kết quả in ra counter là mấy?

counter không xác định được vì counter++ không phải là một thao tác nguyên tử!

# Một số giải pháp tránh xung đột

## Giải pháp 1: lock

```
class Program
{
    static int counter = 0;
    static readonly object locker = new object();

    static void Main()
    {
        Parallel.For(0, 1000000, i =>
        {
            lock (locker)
            {
                counter++;
            }
        });
        Console.WriteLine("Counter: " + counter);
    }
}
```

# Một số giải pháp tránh xung đột

## Giải pháp 2: Monitor

```
class Program
{
    static int counter = 0;
    static readonly object locker = new object();

    static void Main()
    {
        Parallel.For(0, 1000000, i =>
        {
            if (Monitor.TryEnter(locker, TimeSpan.FromMilliseconds(100))){
                try {
                    counter++;
                }
                finally {
                    Monitor.Exit(locker);
                }
            }
            else {
                // bị chiếm
            }
        });
        Console.WriteLine("Counter: " + counter);
    }
}
```

# Một số giải pháp tránh xung đột

## Giải pháp 3: Interlocked

```
class Program
```

```
{
```

```
    static int counter = 0;
```

```
    static void Main()
```

```
{
```

```
    Parallel.For(0, 1000000, i =>
```

```
{
```

```
        Interlocked.Increment(ref counter);
```

```
    };
```

```
    Console.WriteLine(counter);
```

```
}
```

```
}
```

# Một số giải pháp tránh xung đột

## Giải pháp 4: Mutex

```
class Program
{
    static int counter = 0;
    static Mutex mutex = new Mutex();
    static void Main()
    {
        Parallel.For(0, 1000000, i =>
        {
            mutex.WaitOne();
            counter++;
            mutex.ReleaseMutex();
        });
        Console.WriteLine(counter);
    }
}
```

# Một số giải pháp tránh xung đột

## Deadlock:

```
object A = new object();
object B = new object();
```

```
Task t1 = Task.Run(() =>
{
    lock (A)
    {
        Thread.Sleep(100);
        lock (B) { }
    }
});
```

```
Task t2 = Task.Run(() =>
{
    lock (B)
    {
        Thread.Sleep(100);
        lock (A) { }
    }
});
```

# Một số giải pháp tránh xung đột

## Giải pháp tránh Deadlock:

- ✓ Luôn khóa theo **thứ tự cố định**
- ✓ Tránh lock lồng nhau nếu không cần
- ✓ Sử dụng timeout (Monitor.TryEnter)
- ✓ Hạn chế dùng quá nhiều khóa

# Một số giải pháp tránh xung đột

## Tóm tắt các kỹ thuật tránh xung đột:

Kỹ thuật	Mục đích	Tốc độ	Ghi chú
lock	Khoá đơn giản	Trung bình	Dễ dùng, phổ biến
Monitor	Khóa nâng cao	Trung bình	Có timeout
Interlocked	Tăng/giảm nguyên tử	Rất nhanh	Không khóa
Mutex	Khoá xuyên process	Chậm	Dùng khi nhiều process

# Sử dụng đa luồng trong WinForms

## Các nguyên tắc:

- Mọi thao tác UI phải được thực hiện trên UI Thread (UI Thread gồm những code nằm trong các hàm xử lý sự kiện) Nếu phải cập nhật trên thread khác thì phải dùng Invoke
- Không chạy tác vụ nặng trên UI Thread
- Không khóa UI bằng các lệnh .Wait(), .Result
- Không khóa UI bằng lệnh Thread.Sleep() trong hàm xử lý sự kiện

# Sử dụng đa luồng trong WinForms

## Hướng dẫn sử dụng control BackgroundWorker:

**BackgroundWorker** là một component có sẵn trong WinForms, giúp:

- Chạy tác vụ nặng ở **background thread**
- Không làm đơ UI
- Có sẵn:
  - Sự kiện chạy nền: **DoWork**
  - Sự kiện báo xong: **RunWorkerCompleted**
  - Sự kiện báo tiến độ: **ProgressChanged**
  - Cơ chế hủy công việc (**Cancellation**)

# Sử dụng đa luồng trong WinForms

## Hướng dẫn sử dụng control BackgroundWorker:

BackgroundWorker có 3 event chính:

### 1. DoWork

1. Chạy trên **background thread**
2. Bạn viết **tác vụ nặng** ở đây (vòng lặp, xử lý file, tính toán...)

### 2. ProgressChanged (tùy chọn)

1. Chạy trên **UI thread**
2. Dùng để cập nhật: ProgressBar, Label, ListBox...

### 3. RunWorkerCompleted

1. Chạy trên **UI thread**
2. Báo công việc **đã xong / bị lỗi / bị hủy**

Thêm 2 property quan trọng:

- **WorkerReportsProgress** = true → mới dùng được ReportProgress
- **WorkerSupportsCancellation** = true → mới hỗ trợ hủy (CancelAsync)

# Sử dụng đa luồng trong WinForms

## Hướng dẫn sử dụng control **BackgroundWorker**:

Cách thêm BackgroundWorker vào WinForms dùng Designer

1. Mở Form (Form1)
2. Mở **Toolbox** → **Components** → **BackgroundWorker**
3. Kéo thả vào Form → nó xuất hiện ở **khung dưới** (component tray)
4. Chọn nó, trong Properties:
  1. WorkerReportsProgress = true
  2. WorkerSupportsCancellation = true
5. Nhấn đúp vào BackgroundWorker:
  1. Tự tạo hàm backgroundWorker1\_DoWork
6. Vào cửa sổ Properties → Events (icon sét) để gán:
  1. ProgressChanged
  2. RunWorkerCompleted