# Bài: Multi Threading (đa luồng) trong C#

*Xem bài học trên website để ủng hộ Kteam: Multi Threading (đa luồng) trong C#*

*Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage How Kteam nhé!*

## Dẫn nhập

Ở các bài học trước, chúng ta đã cùng nhau tìm hiểu về EVENT CHUẨN .NET TRONG C# Hôm nay chúng ta sẽ cùng tìm hiểu về **Multi threading trong C#**.

---

## Nội dung

Để đọc hiểu bài này tốt nhất các bạn nên có kiến thức cơ bản về các phần:

- LẬP TRÌNH C# CƠ BẢN
- LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG trong C#
- EVENT CHUẨN .NET TRONG C#

Trong bài học này, chúng ta sẽ cùng tìm hiểu các vấn đề:

- Multi threading là gì? Vì sao cần Multi threading.
- Cách dùng Thread trong C#
- Cách dùng multi Thread trong C#
- Các lưu ý khi dùng Thread trong C#

---

## Multi threading là gì? Vì sao cần Multi threading

**Multi threading** có thể hiểu là một kỹ thuật để cùng **một lúc xử lý nhiều tác vụ**. Bản chất chương trình C# được tạo ra sẽ có **hai luồng chạy chính**. Luồng thứ nhất là MainThread*(luồng chính của chương trình mặc định là hàm Main)* và UIThread*(luồng cập nhật giao diện)*.

Các code xử lý mà chúng ta code chính là nằm trên luồng MainThread. Và chương trình **sẽ thực hiện tuần tự từng dòng code từ trên đi xuống**. Nên nếu chúng ta có 3 hàm xử lý cần chạy cùng lúc là không thể. Vì chương trình **phải đợi hàm trước đó xử lý xong** mới đến hàm kế tiếp.

Vậy để giải quyết việc cùng một lúc, có thể xử lý nhiều tác vụ, multi threading ra đời.

Trong C# chúng ta có thể dùng Thread để thực hiện đa luồng.*(Cứ tưởng tượng rằng chúng ta sẽ làm cho chương trình không chỉ còn 2 luồng chạy cơ bản song song mà có thêm n luồng do chúng ta tạo ra và chạy song song nhau)*

---

## Cách dùng Thread trong C#

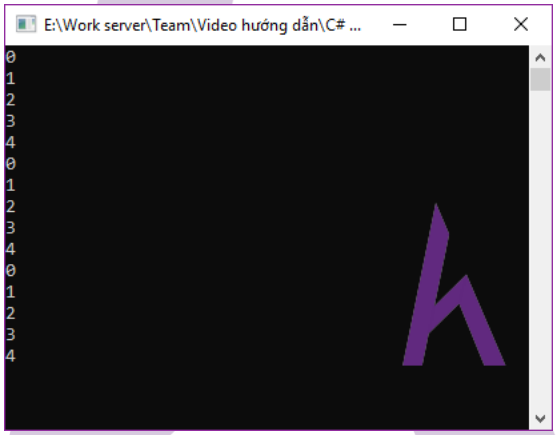
Class Thread nằm trong thư viện System.Threading của **.Net**. Để có thể sử dụng Thread thì chúng ta sẽ using System.Threading.

Chúng ta tạo một hàm DemoThread với mục đích là giả lập lại một hàm xử lý mất thời gian 5 giây. Ở đây mình dùng Thread.Sleep để mô phỏng lại việc sẽ tiêu tốn một giây cho mỗi lần lặp xử lý với 5 lần lặp. Thread.Sleep có nhiệm vụ làm cho luồng hiện tại ngủ theo thời gian được cài đặt.

**C#:**

```
class Program
    {
        static void Main(string[] args)
        {
            DemoThread();
            DemoThread();
            DemoThread();

            Console.ReadLine();
        }

        static void DemoThread()
        {
            // Thực hiện vòng lặp 5 lần. Mỗi lần tốn 1 giây
            for (int i = 0; i < 5; i++)
            {
                // Làm gì đó tốn 1s. Dùng Thread.Sleep để luồng hiện tại ngủ theo thời gian được cài đặt.
                // Mục đích để giả lập độ trễ của code xử lý
                Thread.Sleep(TimeSpan.FromSeconds(1));
                Console.WriteLine(i);
            }
        }
    }
```

Kết quả có thể thấy màn hình **Console** sẽ in ra giá trị của I sau mỗi 1 giây. Sau 5 giây thì hoàn thành. Vậy nếu chúng ta gọi 3 hàm DemoThread này liên tục sẽ tiêu tốn 15s để hoàn thành chương trình.




Bây giờ chúng ta sẽ cho mỗi hàm demo vào một luồng riêng biệt để thực hiện. Lúc này tốc độ sẽ tăng lên đáng kể vì 3 luồng chạy song song nhau. Chúng ta dùng cách tạo và gọi Thread và sử dụng tối ưu nhất để bạn dễ tiếp cận. Bạn có thể tìm hiểu thêm về ThreadStart nếu muốn.

Vì muốn biết giá trị của i được in ra từ hàm nào nên mình truyền thêm tham số cho hàm DemoThread và in kèm giá trị đó.

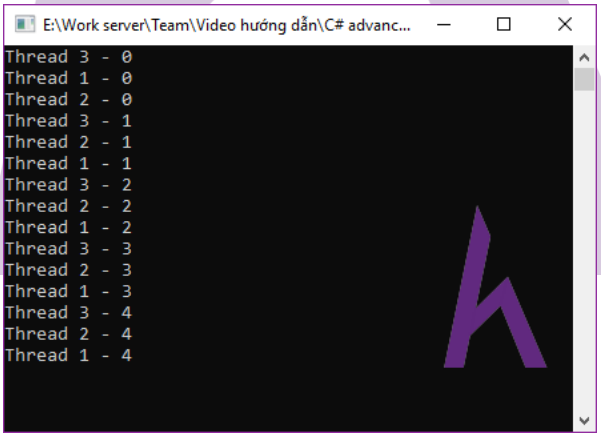
Thread chỉ bắt đầu chạy khi bạn gọi hàm Start.

Sau khi thực hiện thì 5s sau chương trình đã hoàn thành.

:

```
class Program
    {
        static void Main(string[] args)
        {
            /* Tạo một Thread t với anonymous function và gọi hàm DemoThread bên trong
             * Thread chỉ bắt đầu chạy khi gọi hàm Start
             * Bạn có thể thực hiện một hàm hay nhiều dòng code ở bên trong anonymous function này
             */
            Thread t = new Thread(()=> {
                DemoThread("Thread 1");
            });
            t.Start();

            Thread t2 = new Thread(() => {
                DemoThread("Thread 2");
            });
            t2.Start();

            Thread t3 = new Thread(() => {
                DemoThread("Thread 3");
            });
            t3.Start();

            Console.ReadLine();
        }

        static void DemoThread(string threadIndex)
        {
            // Thực hiện vòng lặp 5 lần. Mỗi lần tốn 1 giây
            for (int i = 0; i < 5; i++)
            {
                // Làm gì đó tốn 1s. Dùng Thread.Sleep để luồng hiện tại ngủ theo thời gian được cài đặt.
                // Mục đích để giả lập độ trễ của code xử lý
                Thread.Sleep(TimeSpan.FromSeconds(1));
                Console.WriteLine(threadIndex + " - " + i);
            }
        }
    }
```

**Kết quả:**

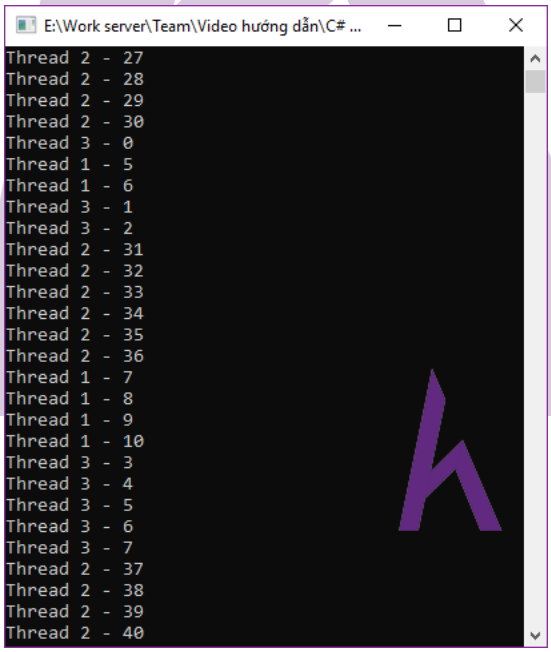



Hiện tại bạn thấy kết quả in ra vẫn theo một trật tự vì mình dùng Thread.Sleep bên trong hàm DemoThread. Bây giờ mình sẽ thay đổi hàm này theo hướng không sleep nữa và cho tăng số lần lặp lên.

:

```
class Program
    {
        static void Main(string[] args)
        {
            /* Tạo một Thread t với anonymous function và gọi hàm DemoThread bên trong
             * Thread chỉ bắt đầu chạy khi gọi hàm Start
             * Bạn có thể thực hiện một hàm hay nhiều dòng code ở bên trong anonymous function này
             */
            Thread t = new Thread(()=> {
                DemoThread("Thread 1");
            });
            t.Start();

            Thread t2 = new Thread(() => {
                DemoThread("Thread 2");
            });
            t2.Start();

            Thread t3 = new Thread(() => {
                DemoThread("Thread 3");
            });
            t3.Start();

            Console.ReadLine();
        }

        static void DemoThread(string threadIndex)
        {
            for (int i = 0; i < 100; i++)
            {
                Console.WriteLine(threadIndex + " - " + i);
            }
        }
    }
```

**Kết quả:**




Bạn nhận thấy rằng thứ tự thực hiện của 3 luồng này lộn xộn và giá trị i in ra cũng vậy. Lý do là vì 3 luồng này đang chạy song song độc lập nhau.

# Cách dùng Multi Threading trong C#

Vậy là bạn đã có thể tắc tốc độ xử lý chương trình của mình bằng Thread. Nhưng mình còn có thể tăng tốc độ xử lý lên nhiều nữa bằng kỹ thuật Multi Threading.

Vậy bây giờ mình muốn thực hiện gọi hàm DemoThread với 5 lần thì mình bắt buộc phải dùng vòng lặp thay vì tạo tay 5 biến Thread để **Start**.

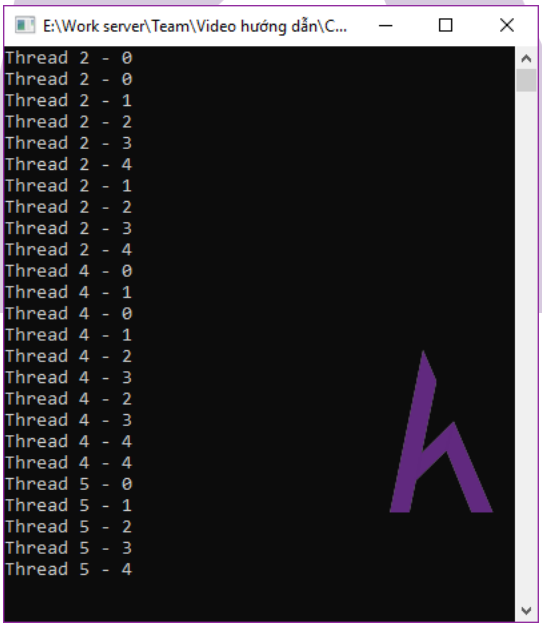
Ý tưởng đơn giản là tạo một vòng lặp 5 lần. Mỗi lần tạo ra một Thread mới và đưa biến i vào hàm DemoThread.

Cùng thực hiện nhé!

:

```
class Program
    {
        static void Main(string[] args)
        {
            for (int i = 0; i < 5; i++)
            {
                Thread t = new Thread(() => {
                    DemoThread("Thread " + i);
                });
                t.Start();
            }

            Console.ReadLine();
        }

        static void DemoThread(string threadIndex)
        {
            for (int i = 0; i < 5; i++)
            {
                Console.WriteLine(threadIndex + " - " + i);
            }
        }
    }
```

**Kết quả:**



Bạn có thể nận thấy. Thiếu đâu Thread 1 và Thread 3.

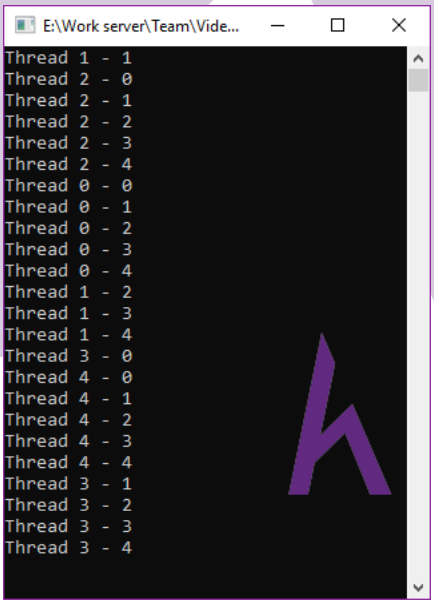
Lý do là vì Thread không **Start** ngay khi bạn gọi hàm **Start**. Mà hệ điều hành sẽ tự điều phối sao cho hợp lý để tối ưu tài nguyên. Trong code bạn đưa vào là biến i. Có thể DemoThread được **Start** ở lần lặp sau đó, suy ra biến i lúc này có giá trị là **2**. Nên thành ra Thread 2 được thực hiện tới 2 lần.

Để giải quyết tình trạng Missed Thread này. Bạn có thể dùng foreach hoặc tạo một biến tạm để lưu giá trị i bên ngoài Thread rồi dùng biến đó thay cho i.

:

```
class Program
    {
        static void Main(string[] args)
        {
            for (int i = 0; i < 5; i++)
            {
                int tempI = i;
                Thread t = new Thread(() => {
                    DemoThread("Thread " + tempI);
                });
                t.Start();
            }

            Console.ReadLine();
        }

        static void DemoThread(string threadIndex)
        {
            for (int i = 0; i < 5; i++)
            {
                Console.WriteLine(threadIndex + " - " + i);
            }
        }
    }
```

Kết quả như mong đợi:




## Lưu ý khi dùng Thread trong C#

Nếu Thread của bạn xử lý nhiều hoặc lặp vô tận. Bạn nên set thuộc tính IsBackground cho Thread là true để khi chương trình của bạn tắt, Thread của bạn cũng sẽ được giải phóng. Còn nếu IsBackground = false. Thì chương trình của bạn phải đợi Thread này chạy xong thì mới có thể kết thúc.

:

```
t.IsBackground = true;
```

Nếu bạn code MultiThreading ở các ngôn ngữ khác như Winform hay WPF thì nên đưa code xử lý liên quan đến giao diện vào trong **Invoke** để tránh đụng độ tài nguyên khi sử dụng.(cấu trúc Invoke có thể khác một chút tùy vào ngôn ngữ bạn dùng)

- Invoke trong Winform:

:

```
// Invoke cần một đối tượng giao diện để Invoke. Ở đây mình dùng this là form hiện tại.
// Bạn có thể dùng control bất kỳ thay thế
this.Invoke(new Action(()=> {
// code của bạn
}));
```

- Invoke trong WPF:

:

```
// Invoke cần một đối tượng giao diện để Invoke.
            // Ở đây mình dùng this là Window hiện tại.
            // Bạn có thể dùng control bất kỳ thay thế
            // WPF cần Dispathcer để Invoke
            this.Dispatcher.Invoke(new Action(()=> {
                // code của bạn
            }));
```

## Kết luận

Nội dung bài này giúp các bạn nắm được:

- Multi threading là gì. Vì sao cần Multi threading.
- Cách dùng Thread trong C#
- Cách dùng multi Thread trong C#
- Các lưu ý khi dùng Thread trong C#

Vậy là chúng ta đã kết thúc KHÓA HỌC C# NÂNG CAO. Mong rằng nhưng bài học này sẽ giúp ích cho công việc của bạn.

Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên "**Luyện tập – Thử thách – Không ngại khó**".