# LẬP TRÌNH HƯỚNG SỰ KIỆN

Giảng viên: ThS. Phan Thanh Toàn

# BÀI 4
# CƠ CHẾ DELEGATE VÀ THREADING

Giảng viên: ThS. Phan Thanh Toàn

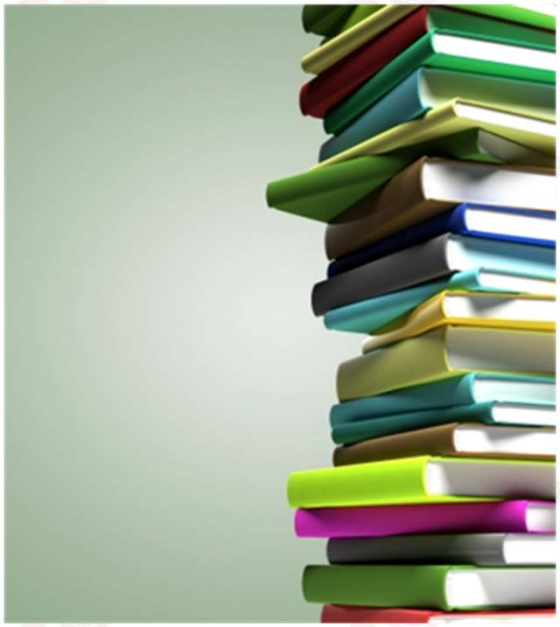
# MỤC TIÊU BÀI HỌC

- Phân biệt được delegate và event.
- Phân biệt được các ứng dụng đơn tiến trình và đa tiến trình.
- Trình bày được khái niệm đồng bộ trong ứng dụng.
- Vận dụng được ngôn ngữ C# vào triển khai các ứng dụng đồng bộ và đa tiến trình.

# CÁC KIẾN THỨC CẦN CÓ

Để học được môn học này, sinh viên phải học xong các môn học:

- Cơ sở lập trình;
- Lập trình hướng đối tượng;
- Cơ sở dữ liệu;
- Hệ quản trị cơ sở dữ liệu SQL Server.



# HƯỚNG DẪN HỌC

- Đọc tài liệu tham khảo.
- Thảo luận với giáo viên và các sinh viên khác về những vấn đề chưa hiểu rõ.
- Trả lời các câu hỏi của bài học.

# CẤU TRÚC NỘI DUNG

# 4.1. DELEGATE VÀ EVENT

4.1.1. Delegate

4.1.2. Event

## 4.1.1. DELEGATE

- Delegate là cơ chế ủy quyền trong lập trình hướng đối tượng.
- Delegate được sử dụng như một con trỏ hàm.
- Delegate đóng gói một phương thức với một tập hợp tham số và kiểu trả về xác định.
- Delegate có thể đóng gói cả phương thức static và phương thức instance.
- Single-cast delegate: Chứa một phương thức.
- Multi-cast delegate: Chứa nhiều phương thức.

## 4.1.1. DELEGATE (tiếp theo)

- Định nghĩa delegate:

```
public delegate <kiểu dữ liệu> <Tên delegate> (danh sách đối số);
```

- Khởi tạo delegate và trỏ delegate tới các phương thức:

```
<Tên delegate> <tên biến> =new <Tên delegate>(Tên phương thức);
```

- Ví dụ: Khai báo một delegate TinhToan và xây dựng 2 hàm Tong (tính tổng 2 số nguyên) và hàm Tru (tính hiệu 2 số nguyên), sau đó khởi tạo và trỏ delegate tới 2 hàm Tong và Tru.
  - Khai báo delegate TinhToan

```
public delegate int TinhToan(int x,int y);
```

  - Xây dựng hàm Tong và Hieu

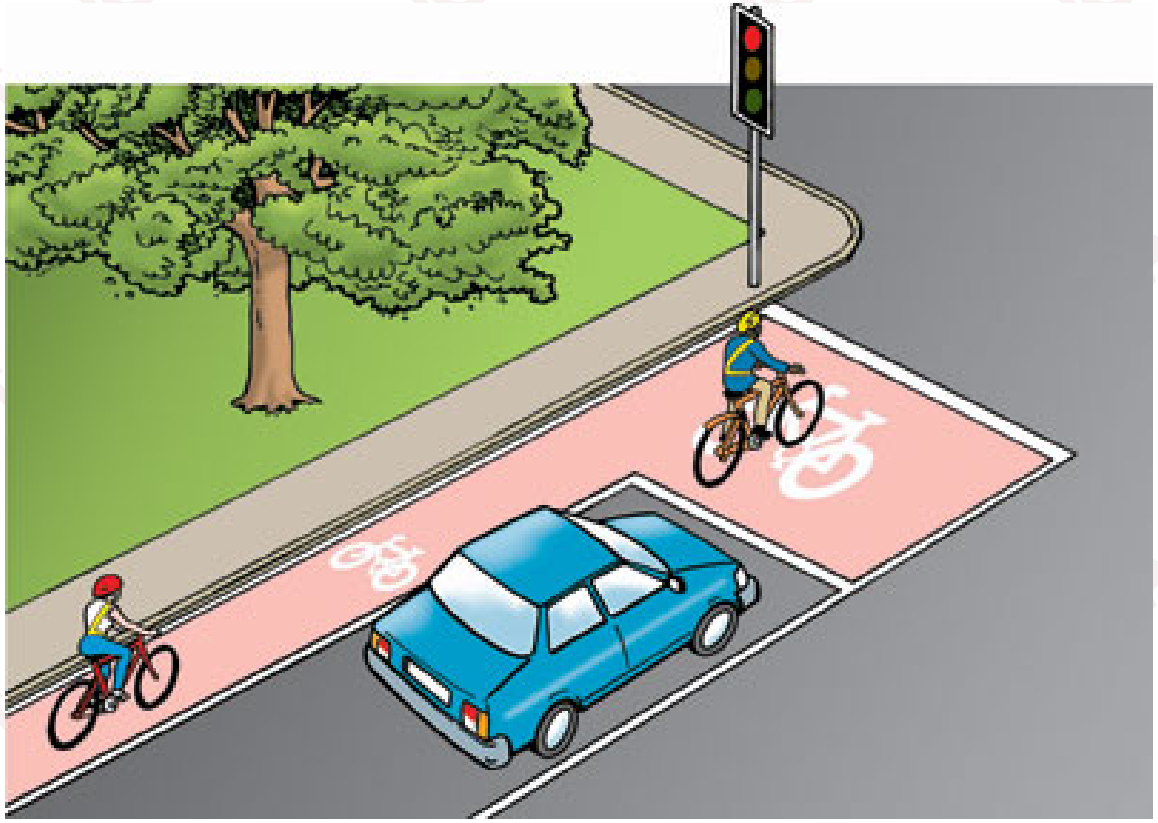
```
static int Tong(int a, int b)
{
return a+b;
}
static int Tru(int a, int b)
{
return a-b;
}
```

## 4.1.1. DELEGATE (tiếp theo)

- Hàm main() sẽ khởi tạo delegate pheptoan và trỏ đến 2 hàm Tong, Tru để thực hiện tính tổng và hiệu của 2 số nguyên

```
 static void Main(string[] args)
{
TinhToan pheptoan = new TinhToan(Tong);
int tong = pheptoan(30, 40);
pheptoan = new TinhToan(Tru);
int hieu = pheptoan(50, 30);
Console.WriteLine("TONG LA: {0}",tong);
Console.WriteLine("HIEU LA: {0}", hieu);
}
```

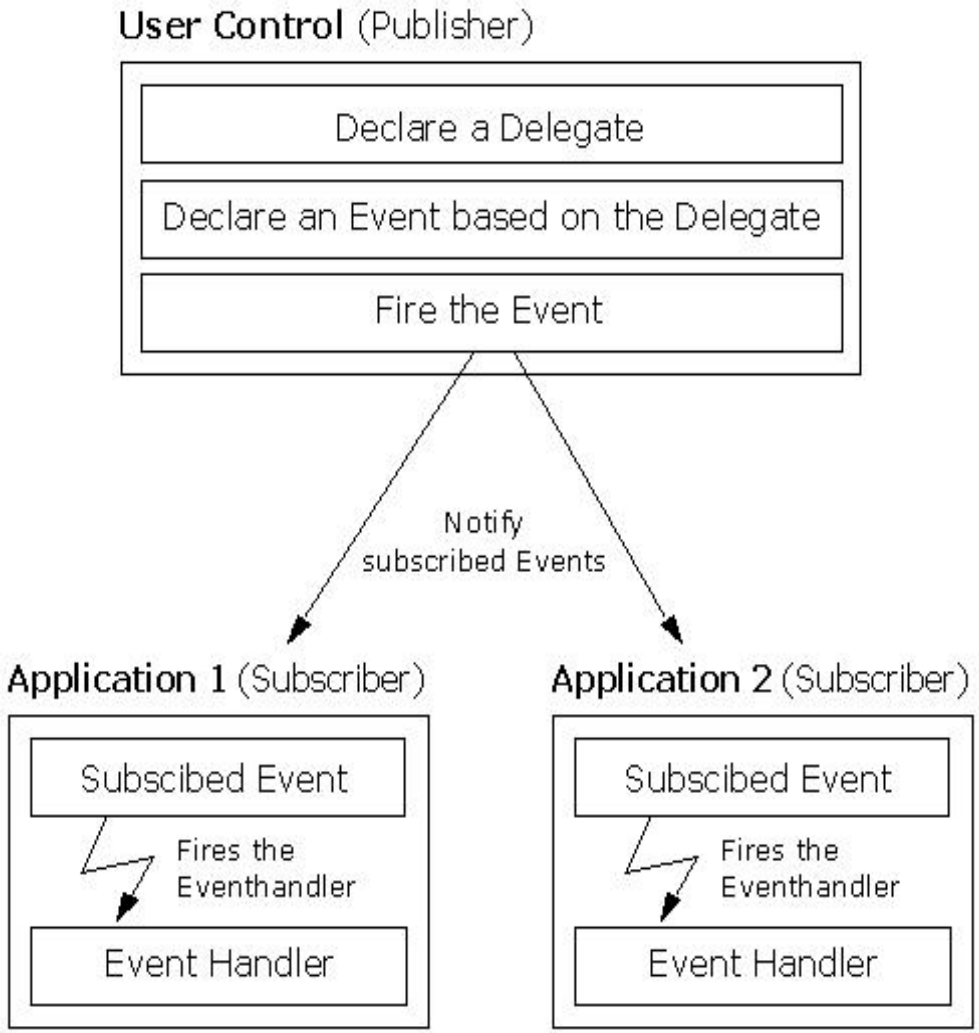
## 4.1.2. EVENT



- Sự kiện là một hành động xác định xảy ra trên một đối tượng.
- Cơ chế gửi thông điệp giữa các lớp hay các đối tượng.
- Gửi thông báo cho lớp khác khi phát sinh ra một sự kiện.
- Mỗi sự kiện thực chất là một delegate.

## 4.1.2. EVENT (tiếp theo)

- Mô hình sự kiện trong C#:
  - Người xuất bản (publisher);
  - Người đăng kí (Subscriber);
  - Publisher thực hiện một số việc và phát ra một sự kiện;
  - Subscriber sẽ mô tả và nhận sự kiện.



## 4.1.2. EVENT (tiếp theo)

- Các quy ước về event trong C#:
  - Event Handlers không có giá trị trả về.
  - Có 2 tham số:
    - Một là nguồn phát sinh sự kiện (publisher);
    - Hai là đối tượng kế thừa từ EventArgs.
  - Những sự kiện là thuộc tính của lớp phát sinh sự kiện.
  - Sử dụng từ khóa event để điều khiển cách mà các thuộc tính event được truy cập bởi các lớp mô tả.

## 4.1.2. EVENT (tiếp theo)

- Lớp Publisher sẽ quyết định khi nào một sự kiện xuất hiện.
- Lớp Subscriber quyết định hành động nào sẽ được thực hiện để đáp ứng sự kiện.
- Một sự kiện có thể có nhiều lớp Subscriber.
- Một subscriber có thể xử lí nhiều sự kiện từ nhiều lớp Publisher.
- Các sự kiện không có lớp Subscriber sẽ không bao giờ xuất hiện.
- Xử lí sự kiện:
  - Khai báo delegate xử lí sự kiện:

```
public delegate void HandlerName(object obj, EventArgs arg);
```

  - Khai báo event:

```
public event HandlerName OnEventName;
```

## 4.1.2. EVENT (tiếp theo)

- Ví dụ: Xây dựng một lớp thực hiện yêu cầu (Publisher), mỗi giây sẽ phát sinh một sự kiện.
- Cho phép 2 lớp khác đăng kí xử lí sự kiện (Subscriber) mỗi lớp có cách xử lí riêng:
  - Lớp A: Hiển thị thời gian theo “mô phỏng đồng hồ Analog”;
  - Lớp B: Hiển thị thời gian theo “mô phỏng đồng hồ Digital”
- Tạo lớp Clock:
  - Khai báo sự kiện: OnSecondChange;
  - Phương thức Run: Cứ 1 giây phát sinh sự kiện OnSecondChange.
- Tạo 2 lớp AnalogClock và DigitalClock nhận và xử lí sự kiện OnSecondChange của lớp Clock.
- Xây dựng lớp Clock:
  - Khai báo delegate để xử lí sự kiện SecondChangeHandler
  - Khai báo sự kiện: SecondChangeHandler
  - Phương thức Run(): Kiểm tra xem có hàm xử lí được đăng kí không, nếu có gọi hàm xử lí sự kiện đã đăng kí

```
if (OnSecondChange != null)
OnSecondChange(this, new EventArgs());
```

## 4.1.2. EVENT (tiếp theo)

```
class Clock
{
  public delegate void SecondChangeHandler(object clock,
  EventArgs info);
 public event SecondChangeHandler OnSecondChange;
 public void Run()
 {
 while (true)
 {
 Thread.Sleep(1000);
 if (OnSecondChange != null)
 {
 OnSecondChange(this, new EventArgs());
 }
 }
 }
}
```

## 4.1.2. EVENT (tiếp theo)

- Xây dựng lớp DigitalClock:

  Xây dựng phương thức xử lí Show của DigitalClock

```
public void Show(object obj, EventArgs args)
{
    DateTime d = DateTime.Now;
    Console.WriteLine("Digital Clock is:
     {0}:{1}:{2}",d.Hour,d.Minute,d.Second);
}
```

- Đăng kí xử lí sự kiện trong lớp DigitalClock:

```
public void Subscribe(Clock theClock)
{
 theClock.OnSecondChange+=new Clock.SecondChangeHandler(Show);
}
```

## 4.1.2. EVENT (tiếp theo)

```
class DigitalClock
{
 public void Subscribe(Clock theClock)
 {
 theClock.OnSecondChange+=new Clock.SecondChangeHandler(Show);
 }
 public void Show(object obj, EventArgs args)
 {
 DateTime d = DateTime.Now;
    Console.WriteLine("Digital Clock is:
  {0}:{1}:{2}",d.Hour,d.Minute,d.Second);
 }
}
```

## 4.1.2. EVENT (tiếp theo)

- Tương tự xây dựng lớp AnalogClock:

```
class AnalogClock
{
 public void Subscribe(Clock theClock)
 {
 theClock.OnSecondChange += new
  Clock.SecondChangeHandler(Show);
 }
public void Show(object obj, EventArgs args)
{
 DateTime d = DateTime.Now;
 Console.WriteLine("Analog Clock is: {0}:{1}:{2}", d.Hour,
  d.Minute, d.Second);
}
}
```

## 4.1.2. EVENT (tiếp theo)

- Gọi thực thi các sự kiện của lớp DigitalClock và AnalogClock

```
Clock myClock = new Clock();
DigitalClock c1 = new DigitalClock();
AnalogClock c2 = new AnalogClock();
c1.Subscribe(myClock);
c2.Subscribe(myClock);
myClock.Run();
```

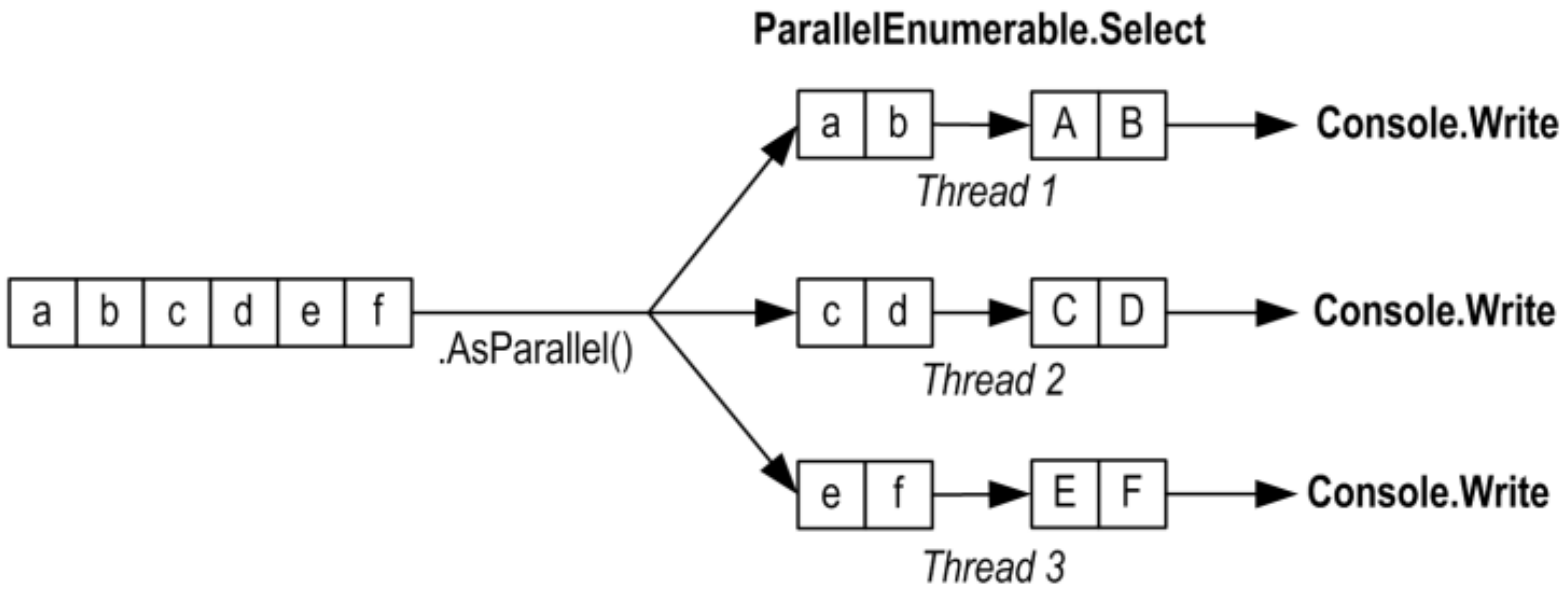
# 4.2. THREADING

4.2.1. Khai báo
và khởi tạo tiến trình

4.2.2. Lập trình đồng bộ

## 4.2.1. KHAI BÁO VÀ KHỞI TẠO TIẾN TRÌNH

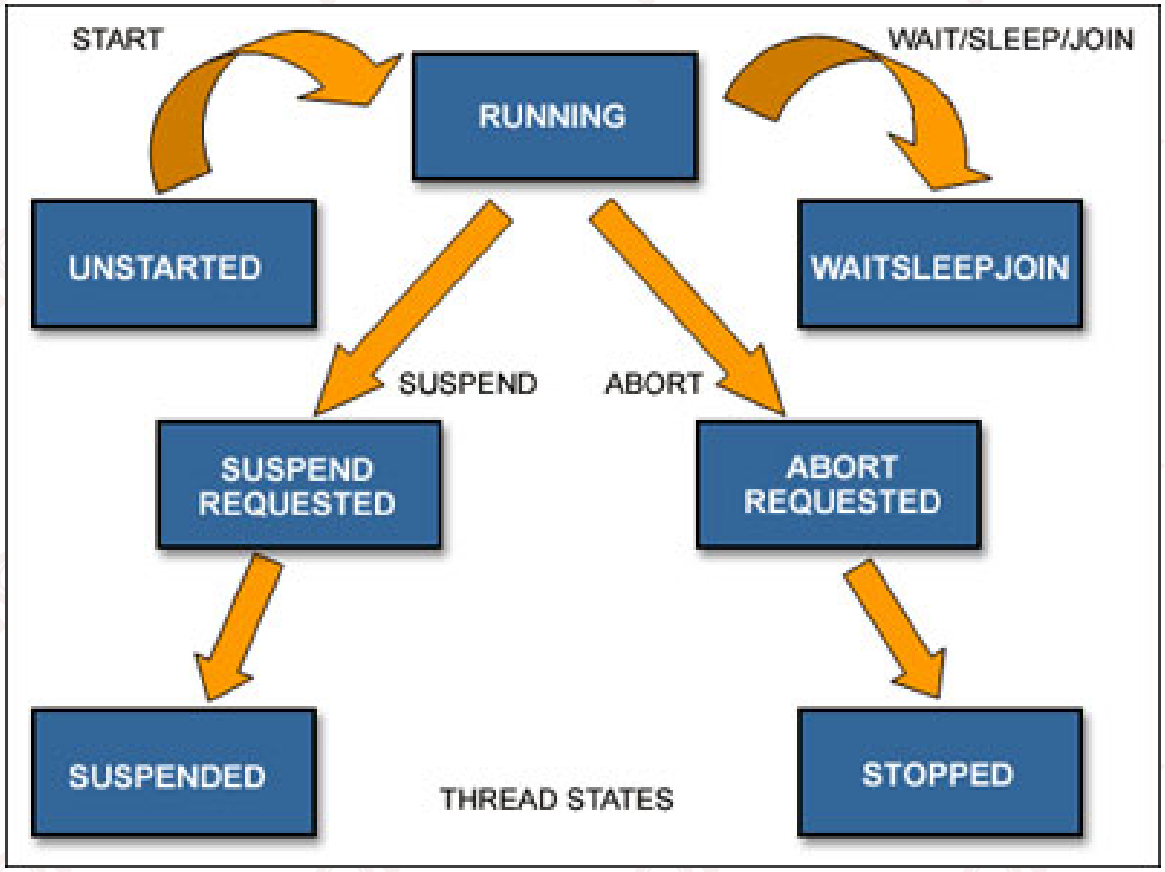
- Tiến trình là một chương trình đang hoạt động.
- Có 2 loại tiến trình: tiến trình của hệ điều hành, tiến trình của người dùng.
- Threading là kĩ thuật hỗ trợ thực hiện nhiều tiến trình tại một thời điểm.
- Lớp Thread được sử dụng để tạo và thực thi tiến trình.




```
"abcdef".AsParallel().Select (c => char.ToUpper(c)).ForAll (Console.Write)
```

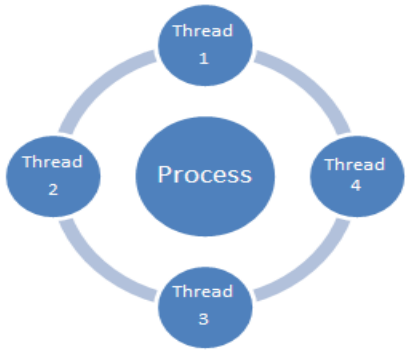
## 4.2.1. KHAI BÁO VÀ KHỞI TẠO TIẾN TRÌNH

- Các thuộc tính:
  - IsAlive;
  - IsBackground;
  - IsThreadPoolThread;
  - Name;
  - Priority;
  - ThreadState;
  - CurrentThread.
- Các phương thức:
  - Abort;
  - Interrupt;
  - Join;
  - Resume;
  - Start;
  - Suspend;



## 4.2.1. KHAI BÁO VÀ KHỞI TẠO TIẾN TRÌNH (tiếp theo)

- Các thao tác trên tiến trình:
  - Khởi tạo tiến trình;
  - Thực thi tiến trình;
  - Ghép nối tiến trình;
  - Tạm dừng và hủy bỏ tiến trình.




- Khởi tạo tiến trình:
  - Tạo phương thức không tham số, không kiểu dữ liệu trả về;
  - Tạo ủy nhiệm hàm ThreadStart với phương thức vừa tạo;
  - Tạo Thread mới với ủy nhiệm hàm ThreadStart vừa tạo.
- Ví dụ: Xây dựng lớp Athlete với phương thức Run

```
 public void Run()
{
float currentLength = 0;
while (currentLength < doanduong)
{
currentLength += (float)speed / 1000;
}
Console.Write("\n{0} ve dich\n", name);
}
```

## 4.2.1. KHAI BÁO VÀ KHỞI TẠO TIẾN TRÌNH (tiếp theo)

- Thực thi tiến trình:
  - Tiến trình được khởi tạo không tự thực thi;
  - Gọi phương thức Start để thực thi tiến trình.

```
Athlete v1 = new Athlete("Nguyen Van Hung",10);
Thread t1 = new Thread(new ThreadStart(v1.Run));
t1.Start();
```

- Thực thi đa tiến trình:
  - Thread hỗ trợ đa tiến trình:

    Ví dụ: Tạo 2 vận động viên (thuộc lớp Athlete) và cho thực hiện đồng thời

```
Athlete v1 = new Athlete("Nguyen Van Hung",10);
Athlete v2 = new Athlete("Le Van Lam",15);
Thread t1 = new Thread(new ThreadStart(v1.Run));
Thread t2 = new Thread(new ThreadStart(v2.Run));
t1.Start();
t2.Start();
```

## 4.2.1. KHAI BÁO VÀ KHỞI TẠO TIẾN TRÌNH (tiếp theo)

- Thiết lập quyền ưu tiên cho các tiến trình:

  <Tên Thread>. Priority = <cấp độ ưu tiên>

  Ví dụ:

```
Athlete v1 = new Athlete("Nguyen Van Hung",10);
Athlete v2 = new Athlete("Le Van Lam",15);
Thread t1 = new Thread(new ThreadStart(v1.Run));
Thread t2 = new Thread(new ThreadStart(v2.Run));
t1.Priority = ThreadPriority.Normal;
t2.Priority = ThreadPriority.Highest;
t1.Start();
t2.Start();
```

- Loại bỏ tiến trình:

  <Tên Thread>. Abort();

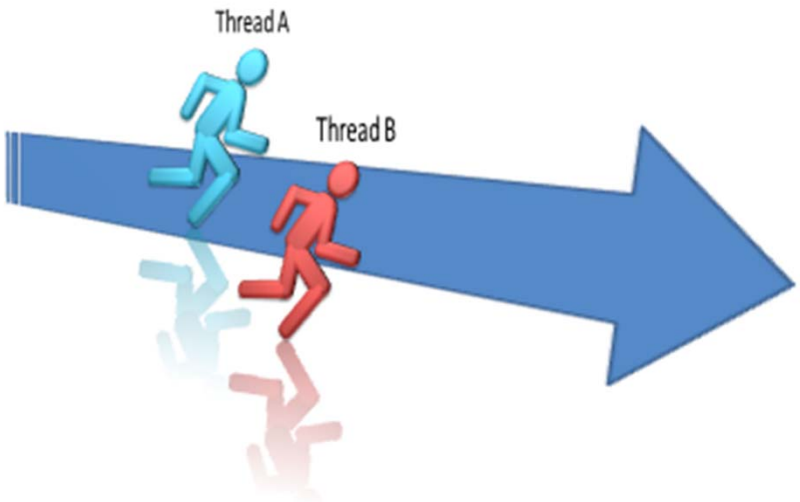
- Tạm dừng một tiến trình:

  <Tên Thread>. Suspend();

  Phương thức này sẽ tạm dừng một tiến trình cho đến khi gặp phương thức Resume().

## 4.2.2. LẬP TRÌNH ĐỒNG BỘ

- Bảo vệ tài nguyên, chỉ cho phép một tiến trình được truy cập dữ liệu.
- Đồng bộ hóa được cung cấp bởi một khóa trên đối tượng, khóa ngăn cản thread khác truy cập vào đối tượng nếu thread thứ nhất chưa trả lại quyền truy cập đối tượng.




**Ví dụ:** Chương trình xây dựng lớp Integer với phương thức Increment để tăng biến counter lên 1. Trong phương thức Increment sẽ sử dụng một biến temp để lưu tạm thời giá trị của biến counter, Thread thứ nhất sẽ tăng biến temp lên 1 và gán trở lại biến counter, sau đó Thread thứ 2 lại đọc giá trị của biến counter ra biến temp và tăng biến temp lên 1 rồi lại gán trở lại biến counter.

## 4.2.2. LẬP TRÌNH ĐỒNG BỘ

```
class Integer
{
 static int counter = 0;
 public void Increment()
 {
 while (counter < 10)
 {
 int temp = counter;
 temp++;
 counter++;
 Thread.Sleep(1000);
 counter = temp;
 Console.WriteLine("Thread {0}. Increment: {1}",
  Thread.CurrentThread.Name,counter);
 }
 }
}
```

## 4.2.2. LẬP TRÌNH ĐỒNG BỘ

- Sử dụng từ khóa lock để khóa khối code cần thực thi

```
static int counter = 0;
public void Increment()
{
 while (counter < 10)
 {
 lock (this)
 {
 int temp = counter;
 temp++;
 Thread.Sleep(1000);
 counter = temp;
 }
 Console.WriteLine("Thread {0}. Increment: {1}",
  Thread.CurrentThread.Name,counter);
 }
}
```

# TÓM LƯỢC CUỐI BÀI

Trong bài này, chúng ta đã nghiên cứu các nội dung chính sau:

- Delegate cơ chế ủy quyền trong lập trình hướng đối tượng;
- Khai báo và khởi tạo sự kiện;
- Lập trình đa luồng và các ứng dụng đồng bộ hóa.