**MSSV: 2312647**

**Họ tên: Lê Anh Khoa**

**Lớp: CTK47A**

**LAB 5**

1. **Lý thuyết**
2. **Sử dụng Tiểu trình (Thread)**

Tiểu trình (Thread) là đơn vị nhỏ nhất của quá trình thực thi trong một chương trình. Mỗi tiến trình (process) có thể bao gồm nhiều tiểu trình cùng chia sẻ vùng nhớ chung, nhưng có ngăn xếp riêng để lưu trữ dữ liệu tạm thời.

Việc sử dụng tiểu trình giúp chương trình thực hiện đồng thời nhiều công việc, tận dụng tốt hơn tài nguyên CPU và nâng cao hiệu suất xử lý, đặc biệt trong các ứng dụng mạng, máy chủ, hoặc giao diện người dùng.

1. **Cách tạo và thực thi tiểu trình**

Trong C#, tiểu trình được quản lý thông qua lớp Thread trong namespace System.Threading.

Có hai cách phổ biến để tạo và chạy một thread:

* **Cách 1:** Truyền một phương thức cần chạy vào đối tượng ThreadStart.

**Thread t = new Thread(new ThreadStart(MyMethod));**

**t.Start();**

* Sử dụng **lambda expression** hoặc **delegate** để truyền trực tiếp đoạn mã cần chạy.

**Thread t = new Thread(() => Console.WriteLine("Hello Thread!"));**

**t.Start();**

Ngoài ra, ta có thể điều khiển hoạt động của thread bằng các phương thức:

* Sleep(ms) – Tạm dừng thread trong thời gian chỉ định (tính bằng mili-giây).
* Abort() – Dừng thread (ít dùng vì dễ gây lỗi).
* Join() – Đợi thread kết thúc trước khi tiếp tục chương trình.

1. **Thread Pool (Bể tiểu trình)**

ThreadPool là một cơ chế giúp **tái sử dụng các tiểu trình sẵn có** thay vì tạo mới mỗi lần.

Nó quản lý tập hợp các thread hoạt động trong nền, giúp giảm chi phí tạo/hủy thread liên tục và tối ưu hiệu năng cho các tác vụ nhỏ, chạy ngắn.

Ví dụ:

**ThreadPool.QueueUserWorkItem(WorkMethod);**

Trong đó WorkMethod là hàm công việc mà thread trong pool sẽ thực thi.  
Cơ chế Thread Pool thường được dùng trong các ứng dụng cần xử lý nhiều yêu cầu đồng thời, như máy chủ TCP, dịch vụ Web, hoặc hệ thống đa người dùng.

1. **Thực hành**

using System;

using System.CodeDom;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace ThreadExample

{

internal class MyThreadClass

{

private const int RANDOM\_SLEEP\_MAX = 1000;

private const int LOOP\_COUNT = 10;

private String greeting;

public MyThreadClass(String greeting)

{

this.greeting = greeting;

}

public void runMyThread()

{

Random rand = new Random();

for(int x = 0; x < LOOP\_COUNT; x++)

{

Console.WriteLine(greeting + "(Thread ID: " + Thread.CurrentThread.GetHashCode() + ")");

try

{

Thread.Sleep(rand.Next(0, RANDOM\_SLEEP\_MAX));

}

catch (ThreadInterruptedException)

{

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace ThreadExample

{

internal class Program

{

static void Main(string[] args)

{

MyThreadClass mtc1 = new MyThreadClass("Day la tieu trinh thu 1 ");

Thread thread1 = new Thread(new ThreadStart(mtc1.runMyThread));

thread1.Start();

MyThreadClass mtc2 = new MyThreadClass("Day la tieu trinh thu 2 ");

Thread thread2 = new Thread(new ThreadStart(mtc2.runMyThread));

thread2.Start();

Console.ReadKey();

}

}

}

**Giải thích ý nghĩa của chương trình:**

Chương trình trên minh họa cách tạo và thực thi nhiều tiểu trình (thread) trong ngôn ngữ C#. Ở đây, lớp MyThreadClass đóng vai trò mô phỏng một công việc đơn giản mà mỗi tiểu trình sẽ thực hiện, còn hàm Main trong lớp Program chịu trách nhiệm khởi tạo và kích hoạt các tiểu trình đó chạy song song.

Trong phương thức Main, ta tạo hai đối tượng MyThreadClass, mỗi đối tượng nhận một chuỗi thông báo riêng như “Đây là tiểu trình thứ 1” và “Đây là tiểu trình thứ 2”. Sau đó, hai đối tượng Thread được khởi tạo tương ứng, mỗi thread được gắn với hàm runMyThread — đây là hàm mà thread sẽ thực thi khi được kích hoạt bằng lệnh Start(). Việc gọi Start() không chạy ngay trong luồng chính, mà yêu cầu hệ thống khởi động hai luồng riêng biệt chạy độc lập. Điều này có nghĩa là cả hai tiểu trình có thể hoạt động đồng thời, cùng chia sẻ CPU theo cách xen kẽ nhau.

Trong lớp MyThreadClass, hàm runMyThread() mô tả công việc mà mỗi tiểu trình sẽ làm. Cụ thể, mỗi thread sẽ thực hiện vòng lặp 10 lần (LOOP\_COUNT = 10), và trong mỗi vòng lặp, nó in ra chuỗi greeting kèm theo mã định danh của thread (Thread.CurrentThread.GetHashCode()). Lệnh Thread.Sleep(rand.Next(0, RANDOM\_SLEEP\_MAX)) khiến mỗi tiểu trình tạm dừng một khoảng thời gian ngẫu nhiên từ 0 đến 1000 mili-giây giữa các lần in, nhằm mô phỏng việc các thread phải chia sẻ CPU. Nhờ có độ trễ ngẫu nhiên này, kết quả in ra màn hình sẽ không theo thứ tự cố định mà xen kẽ nhau, thể hiện rõ ràng việc các tiểu trình đang chạy song song.

Khi chương trình được thực thi, ta sẽ thấy các dòng như:

A screen shot of a computer

AI-generated content may be incorrect.

Thứ tự xuất hiện của các dòng thay đổi mỗi lần chạy vì hệ điều hành phân bổ CPU cho từng thread một cách ngẫu nhiên. Điều này cho thấy tính bất định (non-deterministic) trong lập trình đa luồng — nghĩa là ta không thể đoán trước thread nào sẽ chạy trước.