Module 2: Low-Level APIs, CLR, Value/Reference Types, Optimization

(Tutorial / Lab)

Subjects: Systems Programming

Minor: For all bachelor programs

Link repository Tutorial 02: link

Lab Objective

Students will:

- Use low-level .NET APIs to interact with memory and data structures
- Understand the performance implications of value vs reference types
- Apply memory and speed optimization strategies
- Explore CLR internals with practical exercises

Question 1 – Understanding Value and Reference Types

1. Create a struct PointStruct with int X, Y
2. Create a class PointClass with the same fields
3. Instantiate both and assign one instance to another
4. Modify the copy and observe changes to the original

Tasks:

- Explain why the changes affect (or do not affect) the original
- Discuss implications for memory and performance

Trả lời:

Link code

Lý do thay đổi *copyStruct* không ảnh hưởng đến *origStruck* là do Struct là kiểu giá trị (Value Type). Khi thực hiện lệnh *PointStruct copyStruct = origStruck;*, hệ thống sẽ tạo một bản sao hoàn toàn mới của dữ liệu, copy từng bit của X và Y sau đó lưu vào biến *copyStruct*. Hai biến này nằm ở hai vùng nhớ độc lập. Nên việc gán *copyStruct.X = 999* chỉ thay đổi dữ liệu trên bản sao, bản gốc *origStruck* vẫn giữ giá trị cũ là 10.

Còn *copyClass* ảnh hưởng đến *origClass* là do Class là kiểu tham chiếu (Reference Type). Khi thực hiện *PointClass copyClass = origClass*, hệ thống chỉ sao chép địa chỉ bộ nhớ (con trỏ) trỏ đến đối tượng trên Heap. Cả *copyClass* và *origClass* cùng trỏ về cùng một đối tượng duy nhất. Nên khi sửa *copyClass.X = 999,* thì có nghĩa là đang sửa trực tiếp vào đối tượng đó, nên khi in *origClass.X*, nó cũng sẽ hiển thị là *999*.

Question 2 – Data Structures & Memory Layout

1. Implement a large array of structs vs a large array of classes (same data)
2. Measure memory usage using GC.GetTotalMemory and timing operations on both arrays

Tasks:

- Compare memory footprint
- Compare access speed
- Explain why differences occur in terms of stack vs heap and CLR handling

Trả lời:

Link code

Kết quả 1 lần chạy ngẫu nhiên:

	Struct Array	Class Array
Memory Footprint	76 MB	305 MB
Access Time	32 ms	45 ms

Vấn đề xuất hiện khi muốn chạy nhiều lần để tính trung bình bộ nhớ và thời gian chạy. Do *GC.GetTotalMemory(true)* không đo được chính xác lượng bộ nhớ sử dụng cho từng lần chạy. Nguyên nhân là vì hàm này chỉ trả về tổng managed heap mà GC đang giữ, trong khi GC không giải phóng heap về hệ điều hành sau mỗi lần đo mà tái sử dụng lại cho các lần cấp phát tiếp theo. Do đó, ở các lần chạy sau, bộ nhớ gần như không tăng, dẫn đến kết quả sai lệch và không thể tính trung bình một cách đáng tin cậy.

Để khắc phục, em sử dụng *GC.GetAllocatedBytesForCurrentThread()* vì hàm này đo số byte thực sự được CLR cấp phát mới trong lần chạy hiện tại, không bị ảnh hưởng bởi việc GC tái sử dụng heap. Nhờ đó, có thể đo chính xác memory allocation của struct và class, cho phép tính trung bình 10 lần một cách ổn định, trong khi vẫn giữ *GC.GetTotalMemory(true)* trong hàm *Run()* để demo, nếu muốn xem benchmark chỉ cần dùng hàm *RunAverage(số lần chạy)*.

Kết quả trung bình 10 lần chạy

	Struct Array	Class Array
Memory Footprint	76,29 MB	305,18 MB
Access Time	27,90 ms	665,00 ms

Memory footprint: Struct sử dụng khoảng 76 MB, trong khi class sử dụng khoảng 305 MB, tức class tiêu tốn bộ nhớ gấp khoảng 4 lần struct. Nguyên nhân là vì struct được lưu inline trực tiếp trong mảng, mỗi phần tử chỉ chứa dữ liệu thực (2 biến int = 8 bytes). Ngược lại, class là reference type, mỗi phần tử trong mảng chỉ là con trỏ, còn dữ liệu thật nằm ở các

object riêng biệt trên heap, kèm theo object header và alignment, làm tổng bộ nhớ tăng rất lớn.

Access speed: Thời gian truy cập struct là 27 ms, trong khi class là 665 ms, chậm hơn hơn 24 lần. Struct nhanh hơn vì dữ liệu được lưu liên tục trong bộ nhớ, CPU có thể tận dụng cache hiệu quả khi duyệt tuần tự. Class chậm do mỗi lần truy cập phải theo con trỏ đến object nằm rải rác trên heap, gây nhiều cache miss và tăng chi phí truy cập bộ nhớ.

Lý do có sự khác biệt: Do cách CLR tổ chức bộ nhớ. Struct là value type nên được lưu inline tại nơi chứa nó, trong trường hợp này toàn bộ struct nằm liên tục trong mảng trên heap, tạo thành một khối bộ nhớ liền mạch. Ngược lại, class là reference type, mỗi phần tử trong mảng chỉ là một reference trỏ tới các object riêng biệt nằm rải rác trên heap. Với CLR, mảng struct chỉ tạo một lần cấp phát lớn, GC quản lý đơn giản và hiệu quả, trong khi mảng class tạo ra hàng triệu allocation nhỏ, khiến GC phải theo dõi từng object, làm tăng overhead, giảm hiệu năng và gây nhiều cache miss. Chính sự khác biệt về memory layout, cache behavior và cơ chế GC handling là nguyên nhân dẫn đến chênh lệch lớn về bộ nhớ và tốc độ.

Question 3 – Just-In-Time (JIT) Compilation

1. Describe how the Just-In-Time (JIT) compiler works in .NET
2. Explain when JIT compilation occurs during program execution

Tasks:

- Illustrate with a small code snippet how IL is converted to native code at runtime
- Discuss performance implications of JIT compilation

Trả lời:

Link code

Cách JIT hoạt động: Khi bạn biên dịch code C#, nó trở thành ngôn ngữ trung gian (IL - Intermediate Language). Khi chạy chương trình, JIT Compiler sẽ dịch mã IL này sang mã máy (Native Code) tương thích với cấu trúc phần cứng hiện tại.

Thời điểm xảy ra: Xảy ra tại thời điểm chạy (Runtime), cụ thể là khi một phương thức (method) được gọi lần đầu tiên. Những lần gọi sau sẽ dùng lại mã máy đã dịch.

Hiệu năng:

- *Khởi động:* Chậm hơn một chút do tốn thời gian dịch.
- *Chạy:* Rất nhanh (gần bằng C++) vì là mã máy thuần túy và có thể được tối ưu hóa theo CPU cụ thể của máy đang chạy.

Question 4 – Compiler vs Interpreter

1. Compare a compiler and an interpreter
2. Give one example of each in the context of programming languages

Tasks:

- Discuss advantages and disadvantages of each approach
- Explain how .NET uses a hybrid approach with IL and JIT compilation

Trả lời:

So sánh:

- **Compiler:** Dịch toàn bộ mã nguồn sang mã máy trước khi chạy (ví dụ: C++). Chạy nhanh nhưng thời gian build lâu.

- **Interpreter:** Dịch và chạy từng dòng lệnh một khi chương trình đang chạy (ví dụ: Python, JavaScript cũ). Khởi động nhanh nhưng chạy chậm hơn do phải dịch liên tục.

Cách tiếp cận của .NET (Hybrid):

- C# được Compiler thành mã IL nằm trong file .exe/.dll.
- Khi chạy, JIT sẽ dịch IL sang mã máy thật (Native Code). Điều này kết hợp ưu điểm: code C# có thể chạy trên nhiều nền tảng nhưng vẫn đạt hiệu năng cao (nhờ JIT ra Native code).

Question 5 – Membership Card Printing Program

1. Write a C# program that prints a membership card for a CRM system
2. Ensure the program automatically installs required fonts on the computer if they are not present

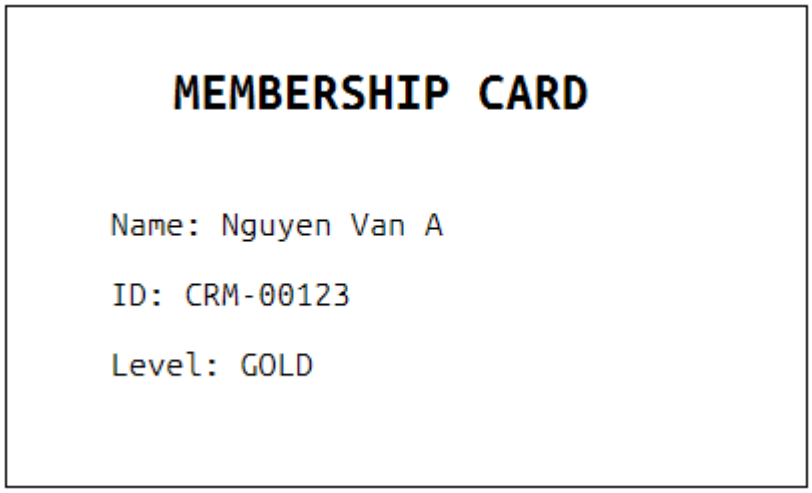
Tasks:

- Demonstrate the program printing a sample card with name, ID, and membership level
- Explain how your program handles font installation and system dependencies
- Discuss any potential security or permission issues when installing fonts programmatically

Trả lời:

Link code

Card được vẽ bằng System.Drawing trên một Bitmap, sau đó lưu vào thư mục Question_05 trong thư mục output của chương trình. Sau khi xuất file, chương trình tự động mở File Explorer để hiển thị vị trí ảnh đã tạo.



Chương trình kiểm tra font theo thứ tự sau:

1. Kiểm tra font đã tồn tại trong hệ thống hay chưa bằng *InstalledFontCollection*.
2. Nếu font chưa tồn tại:
    - Nếu không có quyền *Administrator:* sử dụng *PrivateFontCollection* để load font trực tiếp từ file .ttf mà không cài vào hệ thống.

    - Nếu có quyền *Administrator:* copy font vào thư mục hệ thống *Windows Fonts* và đăng ký font thông qua *Registry*.
3. Khi vẽ nội dung, chương trình ưu tiên dùng font private nếu đã load, nếu không sẽ dùng font hệ thống.

Việc cài font vào hệ thống yêu cầu quyền *Administrator*, vì chương trình phải ghi vào:

- Thư mục hệ thống *Windows\Fonts*
- Registry HKEY_LOCAL_MACHINE

Nếu chạy dưới quyền người dùng thông thường, thao tác này có thể bị từ chối hoặc gây lỗi bảo mật. Ngoài ra, việc chỉnh sửa Registry tiềm ẩn rủi ro nếu ứng dụng độc hại lợi dụng để ghi dữ liệu trái phép. Vì vậy, chương trình sử dụng PrivateFontCollection như một giải pháp an toàn, tránh can thiệp hệ thống khi không đủ quyền, đồng thời giảm rủi ro bảo mật và tăng tính ổn định khi triển khai trên nhiều môi trường khác nhau.