TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN CUỐI KÌ MÔN CHUYÊN ĐỀ CÔNG NGHỆ PHẦN MỀM

Tìm hiểu và viết demo về đa luồng trong Java, C# - So sánh sự khác biệt giữa 2 ngôn ngữ

Người hướng dẫn: **ThS. VŨ ĐÌNH HỒNG**

Người thực hiện: **NGUYỄN TRUNG TÍNH – 51603330**

NGUYỄN TUẤN HUY – 51603144

Lớp **: 160503101**

Khoá **: 20**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2019

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN CUỐI KÌ MÔN CHUYÊN ĐỀ CÔNG NGHỆ PHẦN MỀM

Tìm hiểu và viết demo về đa luồng trong Java, C# - So sánh sự khác biệt giữa 2 ngôn ngữ

Người hướng dẫn: **ThS. VŨ ĐÌNH HỒNG**

Người thực hiện: **NGUYỄN TRUNG TÍNH – 51603330**

NGUYỄN TUẤN HUY – 51603144

Lớp **: 160503101**

Khoá **: 20**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2019

LỜI CẢM ƠN

Chúng em xin chán thành cảm ơn sự hướng dẫn nhiệt tình của thầy Vũ Đình Hồng và thầy Mai Văn Mạnh đã giúp chúng em có thêm kiến thức về đa luồng cũng như lập trình đa luồng trên hai nền tảng Java và C#.

ĐỒ ÁN ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là sản phẩm đồ án của riêng chúng tôi và được sự hướng dẫn của ThS Vũ Đình Hồng. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày tháng năm

Tác giả

(ký tên và ghi rõ họ tên)

Nguyễn Trung Tính

Nguyễn Tuấn Huy

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

TÓM TẮT

Đa luồng hay còn được gọi là Multithreading. Một chương trình đa luồng luôn có hai tiến trình trở lên chạy song song nhau, mỗi tiến trình đó người ta gọi là một luồng (thread). Luồng là đơn vị nhỏ nhất có thể thực hiện được 1 công việc riêng. Một ứng dụng ngoài luồng chính có thể có các luồng khác thực thi đồng thời. Đa luồng giúp cho các tác vụ được xử lý độc lập giúp công việc được hoàn thành nhanh chóng. Vậy đa luồng có thể hiểu đơn giản là quá trình xử lý nhiều thread song song nhau và thực hiện các nhiệm vụ khác nhau cùng một lúc.

MỤC LỤC

DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT

CÁC CHỮ VIẾT TẮT

JVM	Java Virtual Machine
I/O	Input/Output

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

DANH MỤC HÌNH

DANH MỤC BẢNG

CHƯƠNG 1 – KIẾN THỨC VỀ LUỒNG VÀ ĐA LUỒNG

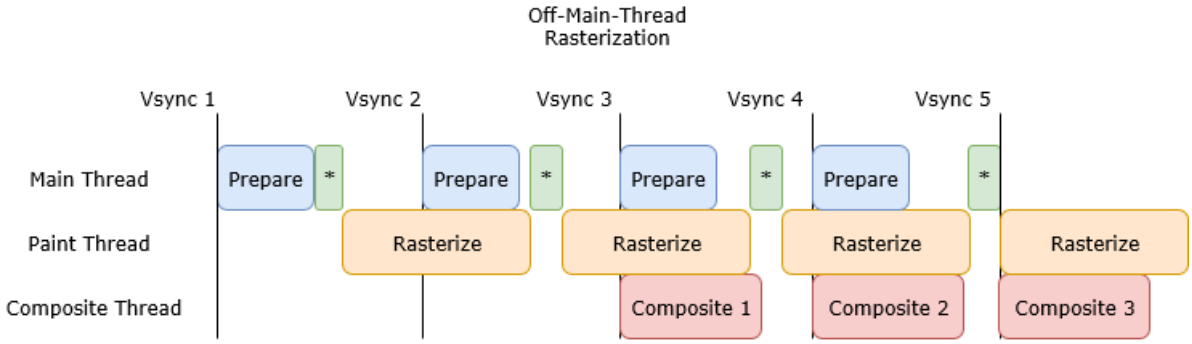
1.1 Giới thiệu

1.1.1 Thread (luồng)

Về cơ bản thì luồng (thread) được hiểu là một tiến trình con (sub-process), là một đơn bị xử lý nhỏ nhất của máy tính có thể thực hiện một công việc riêng biệt.

1.1.2 Multi-thread (đa luồng)

Là một tiến trình thực hiện nhiều luồng đồng thời. Một ứng dụng ngoài luồng chính có thể có các luồng khác thực thi đồng thời làm ứng dụng chạy nhanh và hiệu quả hơn.



Hình 1: Ví dụ về đa luồng

VD: Trình duyệt web hay các chương trình chơi nhạc là 1 ví dụ điển hình về đa luồng.

- Khi duyệt 1 trang web, có rất nhiều hình ảnh, CSS, javascript… được tải đồng thời bởi các luồng khác nhau.
- Khi play nhạc, chúng ta vẫn có thể tương tác được với nút điều khiển như: Play, pause, next, back … vì luồng phát nhạc là luồng riêng biệt với luồng tiếp nhận tương tác của người dùng

1.1.3 Đa nhiệm (Multitasking)

Hình 2: Đa nhiệm trong máy tính

Là khả năng chạy đồng thời một hoặc nhiều chương trình cùng một lúc trên một hệ điều hành. Hệ điều hành quản lý việc này và sắp xếp lịch phù hợp cho các chương trình đó. Ví dụ, trên hệ điều hành Windows chúng ta có làm việc đồng thời với các chương trình khác nhau như: Microsoft Word, Excel, Media Player, …

1.1.3.1 Có 2 loại đa nhiệm

- Đa nhiệm dựa trên tiến trình (Process) – Đa tiến trình (Multiprocessing)
  - Mỗi tiến trình có địa chỉ riêng trong bộ nhớ, tức là mỗi tiến trình phân bổ vùng nhớ riêng biệt
  - Tiến trình khá nặng
  - Sự tương tác giữa các tiến trình tốn chi phí cao
  - Việc chuyển đổi từ tiến trình này sang tiến trình khác đòi hỏi thời gian để đăng ký việc lưu và tải lại bộ nhớ, các danh sách cập nhật,…
- Đa nhiệm dựa trên luồng (Thread) – Đa luồng (MultiThreading)

- Các luồng chia sẻ không gian địa chỉ ô nhớ giống nhau
- Luồng khá nhẹ
- Sự tương tác giữa các luồng tốn ít chi phí

1.1.4 Ưu điểm và nhược điểm của đa luồng

1.1.4.1 Ưu điểm

- Luồng là độc lập vì vậy nó không ảnh hưởng đến luồng khác nếu ngoại lệ xảy ra trong một luồng duy nhất.
- Không chặn người sử dụng vì các luồng là độc lập và có thể thực hiện nhiều công việc cùng một lúc.
- Mỗi luồng có thể dùng chung và chia sẻ nguồn tài nguyên trong quá trình chạy, nhưng thực hiện một cách độc lập.
- Có thể thực hiện nhiều hoạt động với nhau để tiết kiệm thời gian. Ví dụ một ứng dụng có thể tách thành : luồng chính chạy giao diện người dùng, luồng phụ gửi kết quả xử lý đến luồng chính.

1.1.4.2 Nhược điểm

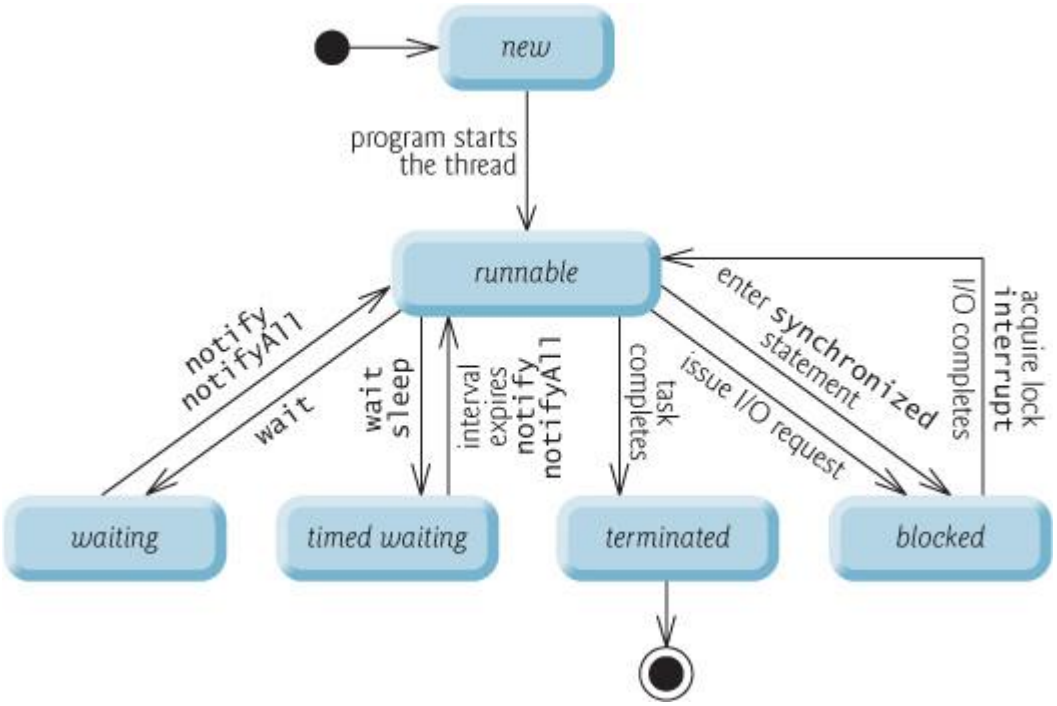
- Càng nhiều luồng thì xử lý càng phức tạp
- Xử lý vấn đề tranh chấp bộ nhớ, đồng bộ dữ liệu khá phức tạp
- Cần phát hiện, tránh các luồng chết (deadlock), luồng chạy mà không làm gì trong ứng dụng.

CHƯƠNG 2: SO SÁNH SỰ KHÁC BIỆT VỀ ĐA LUỒNG TRONG C# VÀ JAVA

2.1 Đa luồng trong Java

2.1.1 Vòng đời (các trạng thái) của một Thread trong Java

Vòng đời của thread trong java được JVM (Java Virtual Machine) kiểm soát. Java định nghĩa các trạng thái của luồng trong các thuộc tính static của lớp Thread.State



Hình 3: Vòng đời của một thread trong Java

- **new:** Trạng thái khi luồng được khởi tạo bằng phương thức khởi tạo của lớp Thread nhưng chưa được start(). Trong trạng thái này, luồng được tạp ra nhưng chưa được cấp phát tài nguyên và cũng chưa chạy.
- **runnable:** sau khi gọi phương thức start() thì luồng được cấp tài nguyên và lập lịch của CPU của luồng cũng có hiệu lực. Ở đây dùng runnable chứ không phải running vì luồng không luôn luôn chạy mà tùy vào hệ thống mà có sự điều phối CPU.

- **waiting:** ở trạng thái này luồng sẽ chờ cho đến khi có luồng khác đánh thức nó
- **timed_waiting:** luồng chờ trong một thời gian nhất định, hoặc là có luồng khác đánh thức nó
- **blocked:** là một trạng thái "not runnable", là trạng thái khi luồng còn sống nhưng không được chọn để chạy. Nó đang chờ một monitor để unlock một đối tượng mà nó cần
- **terminate:** một luồng ở trong trạng thái terminated hoặc dead khi phương thức run của nó thoát

2.1.2 Cách tạo luồng trong Java

Trong Java ta có thể tạo ra một luồng bằng một trong hai cách sau : tạo một đối tượng của lớp được **extend** từ **class Thread** hoặc **implements** từ **interface Runnable**

2.1.2.1 Tạo luồng bằng cách extend từ lớp Thread

Để tạo luồng từ việc kế thừa từ lớp Thread, ta thực hiện như sau

- Khai báo một lớp mới kế thừa từ lớp Thread
- Override lại phương thức run của lớp này, những gì được viết trong phương thức run sẽ thực thi khi luồng bắt đầu chạy. Sau khi luồng hoàn thành tất cả các lệnh thì luồng cũng tự hủy.
- Tạo một đối tượng của lớp vừa khai báo
- Gọi phương thức start() của đối tượng này để bắt đầu thực thi luồng

2.1.2.2 Tạo luồng bằng cách implement từ Interface Runnable

Để tạo luồng thực hiện từ Interface Runnable, ta thực hiện các công việc sau

- Khai báo một lớp mới implements từ Interface Runnable
- Hiện thực phương thức run() ở lớp này, những gì trong phương thức run() sẽ thực thi khi luồng bắt đầu chạy. Sau khi tất cả các câu lệnh được thực thi thì phương thức run() cũng tự hủy.

- Tạo một đối tượng của lớp vừa khai báo (vd: thr1)
- Tạo một thể hiện của lớp Thread bằng phương thức khởi tạo: Thread(Runnable target)
    - Runnable target: đối tượng được implements từ Interface Runnable
    - Vd: Thread thr1 = new Thread(thr1)
- Gọi phương thức start() của đối tượng thr1

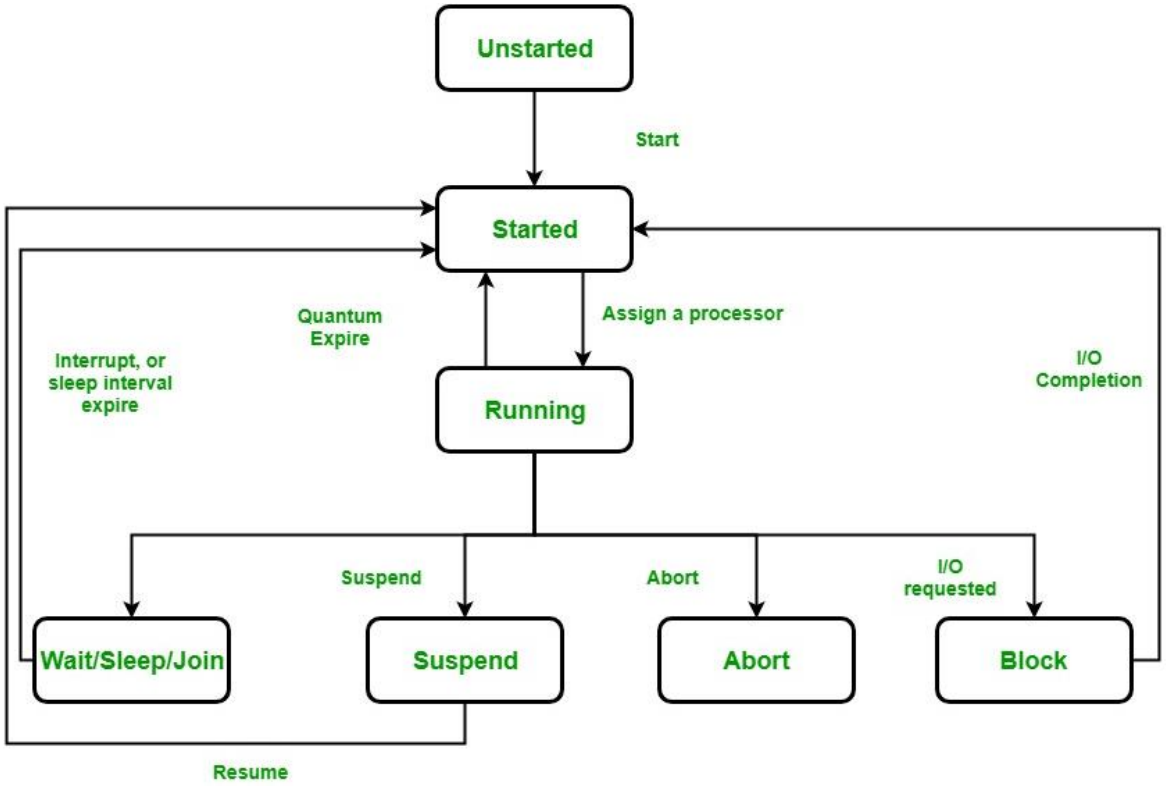
2.1.2.3 Khi nào cần dùng implements từ Interface Runnable

- Đây là cách hay được sử dụng bởi vì nó không yêu cầu phải tạo một lớp kế thừa từ lớp Thread. Trong trường hợp ứng dụng thiết kế yêu cầu sử dụng đa thừa kế thì chỉ có interface mới có thể giải quyết vấn đề
- Trong các trường hợp còn lại ta có thể kế thừa từ lớp Thread

2.2 Đa luồng trong C#

2.2.1 Vòng đời của Thread trong C#

Vòng đời của một Thread bắt đầu khi một đối tượng của lớp System.Threading.Thread được tạo và kết thúc khi Thread đó được kết thúc hoặc hoàn thành việc thực thi.



Hình 4: Vòng đời của một Thread trong C#

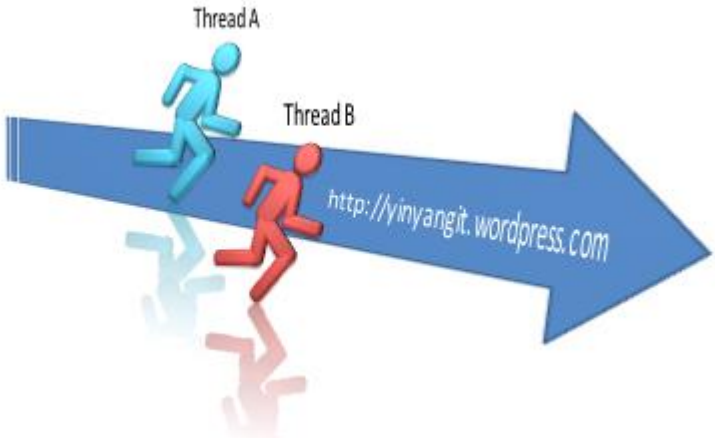
Dưới đây là các trạng thái đa dạng trong vòng đời của một Thread trong C#

- **Unstarted State:** Nó là tình huống khi thể hiện của Thread được tạo, nhưng phương thức Start chưa được gọi
- **Ready State**: Nó là tình huống khi Thread đó sẵn sàng để chạy và đợi CPU cycle.
- **Not Runnable State**: Một Thread là không thể thực thi (not executable), khi:

- Phương thức Sleep đã được gọi
- Phương thức Wait đã được gọi
- Bị chặn bởi hoạt động I/O

– **Dead State**: Nó là tình huống khi Thread hoàn thành sự thực thi hoặc bị hủy bỏ

2.2.2 Tạo và thực thi một Thread



Hình 5: Tạo và thực thi một thread

.Net cung cấp lớp Thread trong namespace System.Threading cùng với những phương thức cần thiết để giúp lập trình viên sử dụng một cách đơn giản và hiệu quả. Để tạo một thread mới bạn làm theo các bước sau:

– Tạo phương thức (gọi là phương thức callback) sẽ thực thi ghi thread được gọi: Phương thức này phải không có tham số hoặc chỉ có một tham số là kiểu object và kiểu trả về là void. Bước này có thể bỏ qua vì ta có thể sử dụng sử dụng anonymous method hoặc lambda expression để tạo đoạn mã lệnh thực thi in-line cùng với lệnh khởi tạo thread.

– Tạo đối tượng Thread và truyền một delegate ThreadStart chứa phương thức sẽ thực thi vào constructor của Thread.

– Chạy thread: Gọi phương thức Start() của đối tượng thread vừa tạo

2.2.3 Property ThreadState và ThreadPriority

2.2.3.1 ThreadState

Thuộc tính ThreadState cho thấy trạng thái hiện tại của thread. Mỗi một lời gọi phương thức của thread sẽ làm thay đổi giá trị thuộc tính này như Unstarted, Running, Suspended, Stopped, Aborted,….

2.2.3.2 ThreadPriority

Thuộc tính này xác định mức độ ưu tiên mà thread sẽ được thực thi so với các thread khác. Mỗi thread khi được tạo ra mang giá trị priority là Normal. Các giá trị mà thuộc tính có thể có bao gồm: Lowest, BelowNormal, Normal, AboveNormal và Highest.

2.3 So sánh giữa C# và Java

2.3.1 Về cú pháp:

Nội dung	Java	C#
Làm một thread sleep	Thread.sleep(1000);	Thread.sleep(1000);
Tạo một thread	Tạo một lớp extends lớp Thread hoặc implements Interface Runnable	Viết một phương thức protected
Start một thread	<Tên thread>.start();	<Tên thread>..start();
Dừng một thread	<Tên thread> = null;	<Tên thread>.Abort();
Làm gián đoạn một thread	<Tên thread>.stop();	<Tên thread>.Interrupt();

Bảng 1: So sánh luồng Java và C#

Về cơ bản việc sử dụng luồng hay đa luồng trong C# và Java đều khá tương đồng nhau, chỉ có một ít sự khác biệt đó là cách thức để tạo dựng một thread.

2.3.2 Về tốc độ xử lý

2.3.2.1 Ví dụ 1:

Trong ví dụ này, ta viết chương trình bằng Java và C# tạo 7000 thread. Trong đó, ta xét 100 trường hợp đại diện. Thời gian cần thiết để tạo và bắt đầu được đo và ghi lại trong bảng kết quả sau:

	Số lượng threads	Thể hiện	χ/s
C#	7000	100	0,722
Java	7000	100	0,616

Bảng 2: Bảng kết quả ví dụ 1

2.3.2.2 Ví dụ 2:

Ví dụ này mô phỏng đơn giản về sự cạnh tranh giữa các luồng trên một tài nguyên phân tán, trong trường hợp này là số dư tài khoản ngân hàng. Mỗi trong số bốn luồng được tạo sẽ cố gắng tăng số dư tài khoản ngân hàng, sau đó họ sẽ cố gắng giảm số dư. Vì tại bất kỳ thời điểm nào, một tài khoản ngân hàng chỉ có thể được truy cập bởi một luồng, điều đó sẽ xảy ra, trong khi tài khoản ngân hàng đang bị thao túng bởi một luồng, một luồng khác có thể bắt đầu thực hiện nhiệm vụ của mình trên cùng một tài khoản, không phải là một tình huống mong muốn, ta sẽ cho các các luồng được đồng bộ hóa. Trong trường hợp cụ thể này, chúng tôi sẽ sử dụng khóa từ khóa synchronization trong .Net và synchronized trong Java . Chương trình sẽ được kiểm tra bằng 300 lần lặp và kết quả thu được sẽ được hiển thị trong bảng kết quả:

	Thể hiện	Số luồng	Thể hiện/s	Test/s
C#	300	4	0,018	5,395
Java	300	4	0,008	3,014

Bảng 3: Bảng kết quả đồng bộ hóa luồng

Dựa vào hai ví dụ trên, ta thấy được rằng tốc độ xử lý luồng ở Java hơn hẳn C#

TÀI LIỆU THAM KHẢO

Tiếng Anh

1. https://www.javatpoint.com/multithreading-in-java (1/12/2019)
2. https://docs.oracle.com/javase/tutorial/essential/concurrency/ (1/12/2019)
3. https://www.tutorialspoint.com/csharp/csharp_multithreading.htm (1/12/2019)
4. https://www.codeproject.com/Articles/1083/Multithreaded-Programming-Using-C (1/12/2019)