ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Khoa Công Nghệ Thông Tin

Đề tài
Tìm hiểu thư viện Boost C++

Sinh viên thực hiện:

- Nguyễn Trần Hậu 1612180
- Nguyễn Duy Thanh 1612628
- Tăng Nguyễn Hoàng Phi 1612493

Giáo viên hướng dẫn:

- Nguyễn Minh Huy
- Hồ Tuấn Thanh
- Trương Toàn Thịnh

Hồ Chí Minh - 2017

1. Mục Lục

2. Giới thiệu

Thư viện Boost là một bộ sưu tập nhiều thư viện C++. Mọi người có thể sử dụng, sửa đổi và phân phối những thư viện này miễn phí, cho mục đích thương mai hoặc phi thương mại. Thư viện Boost là nền tảng độc lập, hỗ trợ nhiều hệ điều hành (Linux, Windows, BSD) và hầu hết các trình biên dịch.

Cộng đồng Boost chịu trách nhiệm cho việc phát triển và phát hành thư viện Boost. Lí tưởng của cộng đồng là phát triển thư viện bổ sung cho thư viện chuẩn C++.

10 thư viện trong Boost được tích hợp trong TR1 (C++ Technical Report). Chuẩn C++11 thêm nhiều thư viện từ Boost so với TR1. Rất nhiều thư viện trong Boost được đề xuất để thêm vào chuẩn C++17.

Tại sao phải sử dụng thư viện Boost? Đó là “năng suất”. Sử dụng thư viện Boost có thể tăng tốc quá trình lập trình, kết quả ít lỗi, tránh trường hợp “phát minh lại bánh xe” cũng như giảm chi phí bảo trì.

3. Kiến trúc

3.1. Quá trình phát triển

Thư viện Boost được phát triển bởi các lập trình viên cá nhân và một số tổ chức. Yêu cầu của Boost là chỉ chấp nhận những thư viện giải quyết vấn đề đang mắc phải, trên chuẩn C++ hiện đại, với chú thích rõ ràng.

Các lập trình viên C++ có thể gợi ý, giới thiệu, viết thư viện mới cho Boost. Tuy nhiên, cần rất nhiều thời gian và công sức. Do đó, người yêu cầu phải thảo luận kĩ với các lập trình viên khác.

Điều kiện để một thư viện được xét vào trong thư viện Boost là dựa vào quá trình nhận xét. Lập trình viên của thư viện đó có thể yêu cầu một đợt nhận xét, thông thường kéo dài 10 ngày. Dựa vào số lượng nhận xét tích cực hay tiêu cực mà thư viện có được chấp nhận hay không.

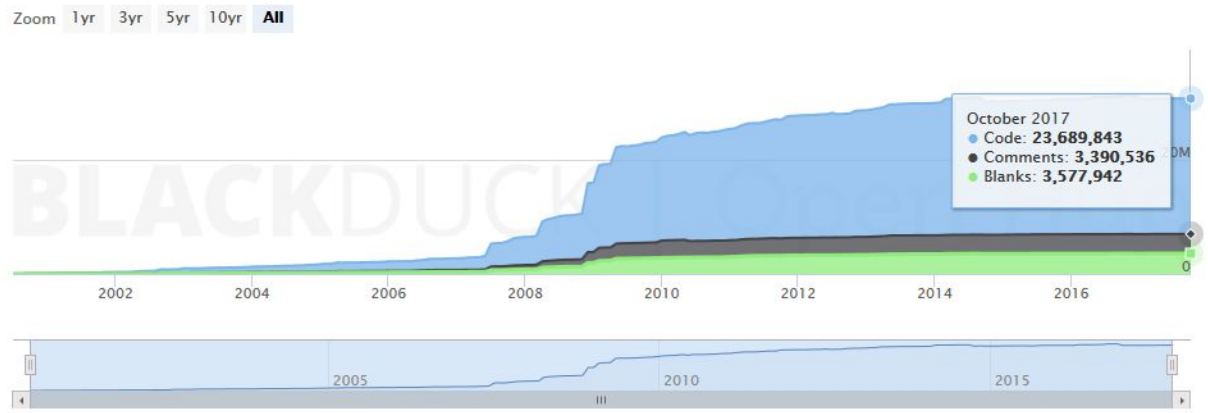
Nếu một thư viện bị từ chối vì lý do kĩ thuật, thư viện đó sau khi được cập nhập có thể yêu cầu nhận xét lại. Tuy nhiên, nếu bị từ chối vì: không giải quyết những vấn đề cụ thể; hay không có cách giải quyết tốt hơn cho một vấn đề cũ; thì khả năng cao là thư viện sẽ bị từ chối một lần nữa.

Vì các thư viện mới có thể được thêm vào Boost bất kì lúc nào nên phiên bản mới của Boost được phát hành liên tục cứ sau 3 tháng. Điều này giúp cho lập trình viên luôn sử dụng phiên bản mới nhất của Boost.

3.2. Tổng quan

- ❏ Boost có hơn một trăm thư viện.
- ❏ Tổng dòng code tính đến tháng 10/2017 là 23,689,736 dòng.

Code, Comments and Blank Lines



- ❏ Ngôn ngữ C++ chiếm 63.4%, HTML chiếm 23.5%, các ngôn ngữ còn lại chiếm 13.1%.

Language Breakdown

Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines	Total Percentage
C++	13,725,927	2,917,768	17.5%	2,796,161	19,439,856	63.4%
HTML	6,605,361	152,492	2.3%	433,586	7,191,439	23.5%
XML	1,912,950	34,934	1.8%	76,741	2,024,625	6.6%
C	363,062	99,688	21.5%	70,120	532,870	1.7%
Autoconf	248,725	319	0.1%	32,029	281,073	0.9%
CSS	200,738	29,603	12.9%	48,903	279,244	0.9%
Python	152,073	93,607	38.1%	55,581	301,261	1.0%
XSL Transformation	130,229	13,313	9.3%	19,937	163,479	0.5%
JavaScript	115,934	2,336	2.0%	7,006	125,276	0.4%

Vì Boost gồm rất nhiều thư viện nên sau đây chỉ kể ra một số thư viện tiêu biểu.

3.2.1. RAIL và quản lý bộ nhớ

RAIL hay Resource Acquisition Is Initialization nghĩa là phải khởi tạo trước khi sử dụng

- Boost.SmartPtr: định nghĩa con trỏ thông minh. Một phần của thư viện đã có trong C++11 thư viện chuẩn. Phần của lại chỉ có trong Boost.
- Boost.PointerContainer: định nghĩa container để lưu các đối tượng khởi tạo bằng new. Trong hàm hủy của container này đã có delete nên không cần sử dụng smart pointer.
- Boost.ScopeExit: sử dụng RAIL cho bất cứ resources nào. Trong khi Smart Ptr và Pointer Container chỉ sử dụng với con trỏ động.
- Boost.Pool: không U liên quan lắm tới RAIL, nhưng thư viện này chứa nhiều xử lý liên quan đến quản lý bộ nhớ. Thư viện này có một vài class có thể cung cấp bộ nhớ cho chương trình nhanh hơn.

3.2.2. Xử lý chuỗi

- Boost.StringAlgo: định nghĩa nhiều thuật toán cho xử lý chuỗi. Ví dụ chuyển chuỗi gồm kí tự thường sang kí tự in hoa.
- Boost.LexicalCast: chuyển số sang chuỗi và ngược lại.
- Boost.Format: cung cấp hàm thay thế std::printf() và an toàn hơn.
- Boost.Regex và Boost.Xpressive: tìm chuỗi với biểu thức chính quy (Regular Expression).
- Boost.Tokenizer: cho phép xử lý thành nhiều chuỗi con từ chuỗi đã cho.
- Boost.Spirit: ứng dụng trong việc viết file cài đặt.

3.2.3. Container

Container: các cấu trúc dữ liệu template

- Boost.Multi-Index: cung cấp container hỗ trợ interface của nhiều container cùng một lúc. Giống như gộp nhiều container với nhau và tính chất từ những container đó.
- Boost.Bimap: dựa trên Multi-Index, cung cấp container tương tự như std::unordered_map, ngoại trừ cả hai đều có thể là chìa khóa.
- Boost.Array: thư viện cung cấp mảng kiểu bất kì với kích thước cố định. Đã được thêm vào C++11.
- Boost.Unordered: cung cấp container sử dụng hash table. Đã được thêm vào C++11.
- Boost.CircularBuffer: cung cấp container với nhiệm vụ chính là ghi đề vào buffer đầu tiên khi buffer đã bị đầy.
- Boost.Heap: cung cấp nhiều loại cài đặt hàng ưu tiên (priority queues).
- Boost.Multi-Array: cài đặt mảng đa chiều dễ hơn.

3.2.4. Cấu trúc dữ liệu

- Boost.Optional: cho phép giá trị trả về lựa chọn cho hàm có thể không có trả về, ví dụ như -1 hay null.
- Boost.Tuple: định nghĩa hàm nhiều giá trị trả về. Đã được thêm vào C++11.
- Boost.Any và Variant: cho phép tạo biến với nhiều kiểu khác nhau.
- Boost.PropertyTree: cấu trúc dữ liệu cây.
- Boost.DynamicBitset: cho phép truy cập vào từng bit của biến bằng toán tử [].
- Boost.Tribool: giống bool nhưng có 3 trạng thái.
- Boost.CompressedPair: tối ưu class trống được kế thừa.

3.2.5. Thuật toán

- Boost.Algorithm: gồm các thuật toán hữu ích.
- Boost.Range: cũng bao gồm thuật toán, nhưng cung cấp khái niệm range, giúp thuật toán chạy nhanh hơn.
- Boost.Graph: cung cấp một số thuật toán cho graph. Ví dụ tìm đường đi ngắn nhất giữa 2 điểm.

3.2.6. Giao tiếp

- Boost.Asio: cung cấp khả năng giao tiếp qua mạng.
- Boost.Interprocess: cung cấp khả năng giao tiếp qua bộ nhớ dùng chung.

3.2.7. Streams và files

- Boost.IOStreams: thư viện nhập xuất.
- Boost.Filesystem: khả năng truy cập và xử lý file, thư mục.

3.2.8. Thời gian

- Boost.DateTime: cung cấp class cho các khoảng thời gian, ví dụ như ngày..
- Boost.Chrono và Boost.Timer: cung cấp khả năng đo thời gian.

3.2.9. Functional Programming

Trong Functional Programming, hàm là đối tượng, có thể làm tham số của hàm khác hoặc được lưu trữ trong container.

- Boost.Phoenix: định nghĩa một số hàm ngắn, không tên như đối tượng.
- Boost.Function: cung cấp một class định nghĩa con trỏ hàm dễ dàng.
- Boost.Bind: cổng giao thức cho phép truyền hàm làm tham số cho một hàm khác mà không cần đúng chữ kí hàm yêu cầu.

- Boost.Ref: truyền tham chiếu của hàm là tham số cho hàm khác.

3.2.10. Lập trình song song

- Boost.Thread: cho phép tạo và kiểm soát thread.
- Boost.Atomic: cho phép truy cập biến chung từ nhiều thread.
- Boost.Lockfree: cung cấp thread-safe container. Container có thể được truy cập từ nhiều thread mà không phải đồng bộ.
- Boost.MPI: hỗ trợ siêu máy tính.

3.2.11. Generic Programming

Generic Programming: lập trình khái quát. Thư viện này cho pháp lập trình khái quát mà không cần hiểu sâu về template meta programming.

- Boost.TypeTraits: cung cấp hàm kiểm tra thông tin của kiểu dữ liệu.
- Boost.EnableIf: sử dụng chung với Boost.TypeTraits. Ví dụ để overload hàm dựa trên kiểu trả về.
- Boost.Fusion: cung cấp container trong đó các thành phần có nhiều kiểu khác nhau.

3.2.12. Language Extensions

Language Extensions: mở rộng cho ngôn ngữ C++.

- Boost.Coroutine: thêm coroutines vào C++.
- Boost.Foreach: mở rộng cho for. Đã được thêm vào C++11.
- Boost.Parameter: cho phép truyền tham số vào hàm theo cặp tên / giá trị giống như Python.

3.2.13. Xử lý lỗi

- Boost.System: cung cấp class để mô tả và phát hiện lỗi. Đã được thêm vào C++11.
- Boost.Exception: gắn dữ liệu vào exceptions sau khi quăng.

3.2.14. Xử lý số học

- Boost.Integer: kiểu số nguyên với chính xác số byte được sử dụng.
- Boost.MinMax: tìm số lớn nhất và nhỏ nhất trong container đồng thời.
- Boost.Random: tạo ngẫu nhiên số.
- Boost.Numeric Conversion: bảo vệ lỗi không lường trước overflow.

3.2.15. Application Libraries

Application Libraries: những thư viện hỗ trợ lập trình ứng dụng độc lập.

- Boost.Log: thư viện để tạo bộ lưu lại quá trình hoạt động.

- Boost.ProgramOptions: định nghĩa và chuẩn hóa các dòng lệnh mà ứng dụng hỗ trợ.
- Boost.Serialization: dễ dàng làm việc tuần tự. Ví dụ hỗ trợ lưu và tải lại ứng dụng từ file.
- Boost.Uuid: hỗ trợ làm việc với UUID.

3.2.16. Mẫu hình thiết kế

- Boost.Flyweight: ứng dụng trong trường hợp nhiều đối tượng khác nhau được sử dụng trong chương trình và cần phải giảm lượng bộ nhớ sử dụng.
- Boost.Signals2: xử lí dễ dàng mẫu hình thiết kế Observer.

4. Cách cài đặt

Donwload file từ trang web http://www.boost.org/users/download/
Vì đa số thư viện của Boost chỉ gồm file Header, muốn sử dụng chỉ cần Include là được. Tuy nhiên, một số thư viện muốn sử dụng phải build lại.

4.1. Windows

1. Giải nén file đã down về, ví dụ tại thư mục D:\boost_1_65_1
2. Trong Visual Studio 2013, sau khi tạo project C++, nhấn chuột phải vào *Project* chọn *Properties*:
    - Trong mục *Configuration* chọn *All Configurations*.
    - Trong *Configuration Properties* > *C/C++* > *General* > *Additional Include Directories*, thêm đường dẫn tới thư mục boost đã giải nén.
    - Trong *Configuration Properties* > *C/C++* > *Precompiled Headers*, chọn *Not Using Precompiled Headers*.
3. Đối với các thư viện đòi hỏi phải build, mở cmd tại thư mục được giải nén
   Chạy boostrap.bat
   Chạy b2

4.2. Unix

1. Giải nén file đã down về, ví dụ tại thư mục ~/boost_1_65_1.
2. Khi compile bằng g++, thêm -I ~/boost_1_65_1 với -I nghĩa là Include thư viện cùng với phía sau là đường dẫn tới boost, ví dụ
   ```
   g++ -I ~/boost_1_65_1 main.cpp
   ```
3. Đối với các thư viện đòi hỏi phải build, di chuyển vào thư mục vừa giải nén
   Chạy ./bootstrap.sh
   Chạy ./b2 install

5. Những sản phẩm ứng dụng thư viện Boost

- ★ Adobe Software Libraries: cung cấp những tính năng cho việc mô hình hóa hiển thị giao diện người dùng và hoạt động bên trong của các ứng dụng. Adobe Software Libraries dựa trên nhiều phần của thư viện Boost bao gồm Boost.Any, Boost.Bind, Boost.Function, Boost.MPL, Boost.Operators, Boost.Range, Boost.StaticAssert, Boost.Thread và Boost.TypeTraits.
- ★ LyX: một ứng dụng xử lý văn bản cho phép người dùng tiếp cận chỉnh sửa văn bản dựa trên cấu trúc của văn bản. Lyx kết hợp sức mạnh và tính linh hoạt của TeX/LaTex với việc sửa dụng giao diện đồ họa. LyX sử dụng Array, Boost.Bind, Boost.Regex, Boost.TypeTraits, Boost.Function và Boost.Signals.
- ★ Regina: một bộ phần mềm toán học cho các nhà địa chất học, Regina tập trung vào cho việc nghiên cứu 3-manifold triangulations và hỗ trợ cho những mặt đơn giản và những cấu trúc góc.
- ★ XEngine: một nền tảng và công cụ vẽ API-independent 3D theo thời gian thực với sự hỗ trợ cửa những kiến trúc lập trình đồ họa và thực thi trên C++.
- ★ LiquidNet: môi trường kinh doanh, đầu tư online hàng đầu Hoa Kì, đứng top 5 công ty phát triển nhất theo Inc Magazine. LiquidNet sử dụng Boost.SharedPointer, Boost.Bind, Boost.Python, Boost.LexicalCast, Boost.Optional, Boost.Any and Boost.Tuple.
- ★ Océ: một trong những nhà cung cấp chính cho việc quản lý văn bản, hệ thống in và các dịch vụ liên quan. Océ phát triển, sản xuất, cung ứng dịch vụ và bán các loại máy in, máy scan, trang thiết bị với ứng dụng duy nhất đi kèm. Thư viện Boost đảm bảo cho ứng dụng tốt hơn và chất lượng hơn, ứng dụng của Océ sử dụng Boost.Assign, Boost.DateTime, Boost.EnableIf, Boost.FileSystem, Boost.Format, Boost.Function, Boost.Iterator, Boost.MultiIndex, Boost.Operators, Boost.Optional, Boost.PointerContainer, Boost.Signals, Boost.SmartPointers, Boost.Thread, and Boost.Variant.
- ★ và còn nhiều phần mềm khác.

6. Boost.Asio

6.1. Giới thiệu

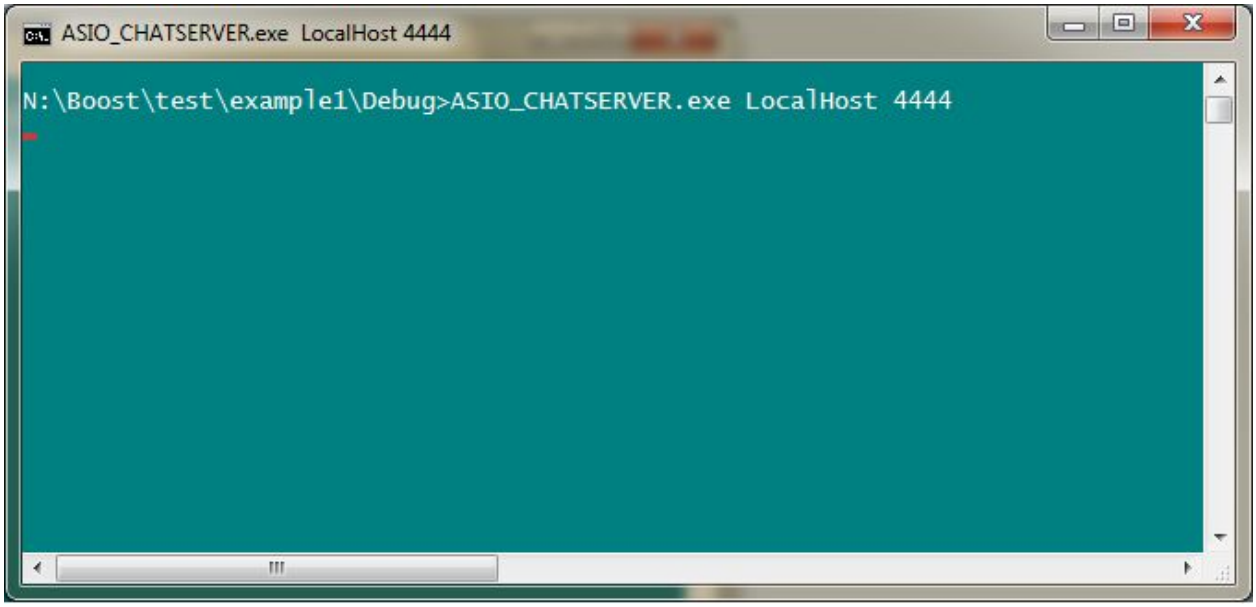
- Boost.Asio là một thư viện cho phép ta lập trình network và I/O tầng thấp. Cung cấp cho những nhà phát triển một mô hình xử lý đồng bộ và không đồng bộ thích hợp, sử dụng C++ hiện nay.
- Boost.Asio được xem như một thư viện networking, ứng dụng nhiều và hiệu quả trong các chương trình, hệ thống client/server hoặc các chương trình Event-driven
- Ban đầu Boost.Asio khi được thiết kế dựa trên mục tiêu sau đây:
    a. Tính linh động.
    b. Khả năng mở rộng tốt.
    c. Tính hiệu quả cao.
    d. Phù hợp với APIs đã được thiết lập.
    e. Dễ sử dụng.
    f. Cơ sở cho sự trừu tượng(bao quát) về sau.

6.2. Demo

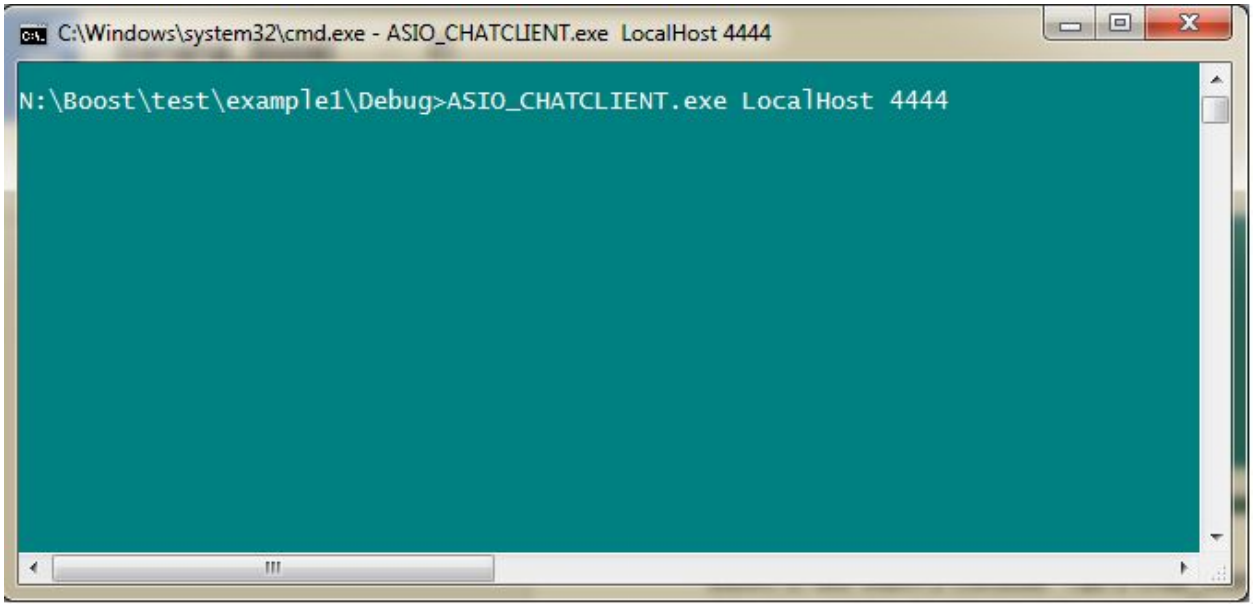
Sau đây sử dụng Boost Asio để demo một chương trình chat client/server trên Console Window. Demo là một ví dụ về xử lý network trong thư viện Boost.

Nguồn:http://www.boost.org/doc/libs/1_65_1/doc/html/boost_asio/examples/cpp11_examples.html

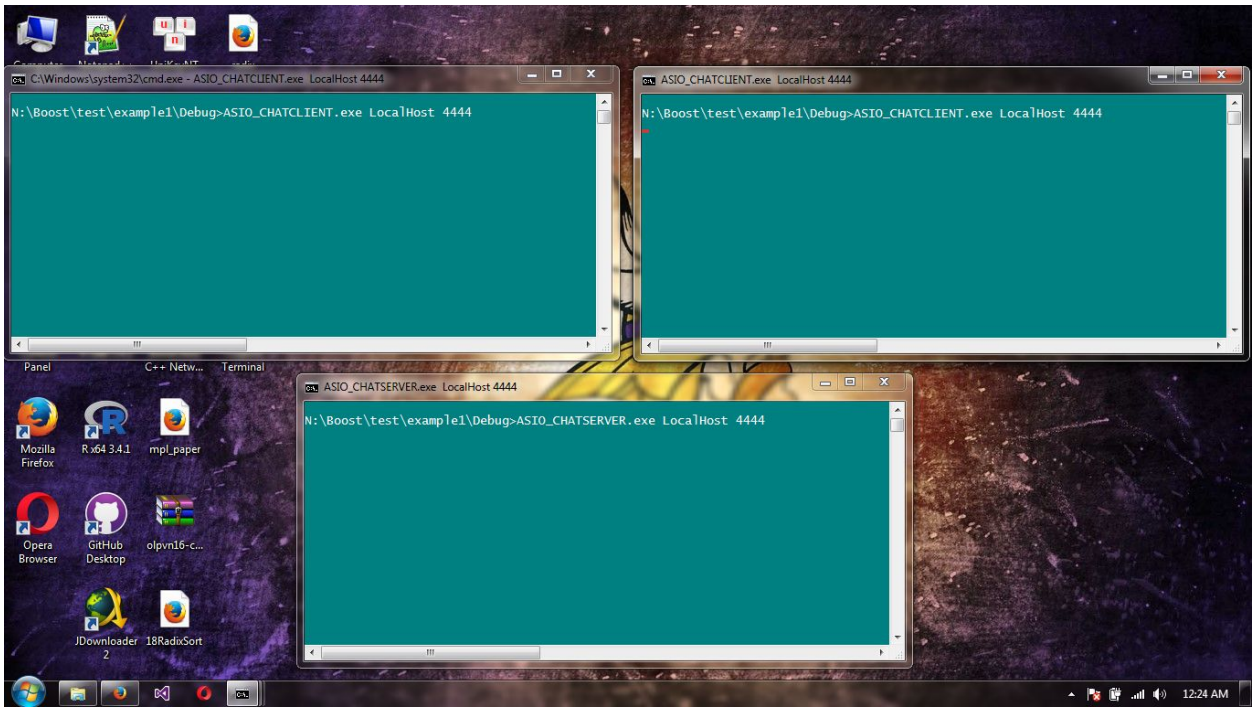
★ Bước 1: Mở 1 Console. Tạo chat_server <port> [<port> ...]
Câu lệnh: ASIO_CHATSERVER.exe LocalHost 4444. Tức là, ta tạo một chat_server host là LocalHost có port 4444.



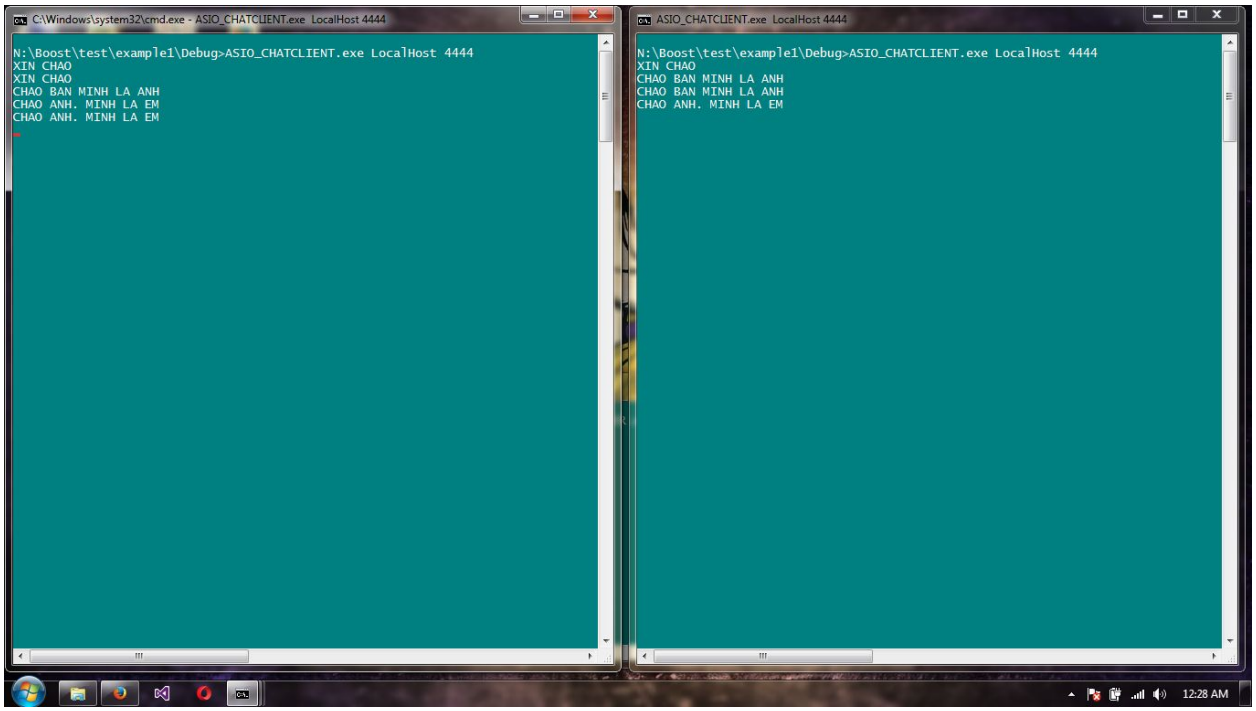
★ Bước 2: Mở thêm 2 console. Tạo 2 chat_client <host> <port>
Câu lệnh: ASIO_CHATCLIENT.exe LocalHost 4444



Kết quả:



★ Bước 3: Ta thực hiện việc chat giữa giữa 2 màn hình console CLIENT.



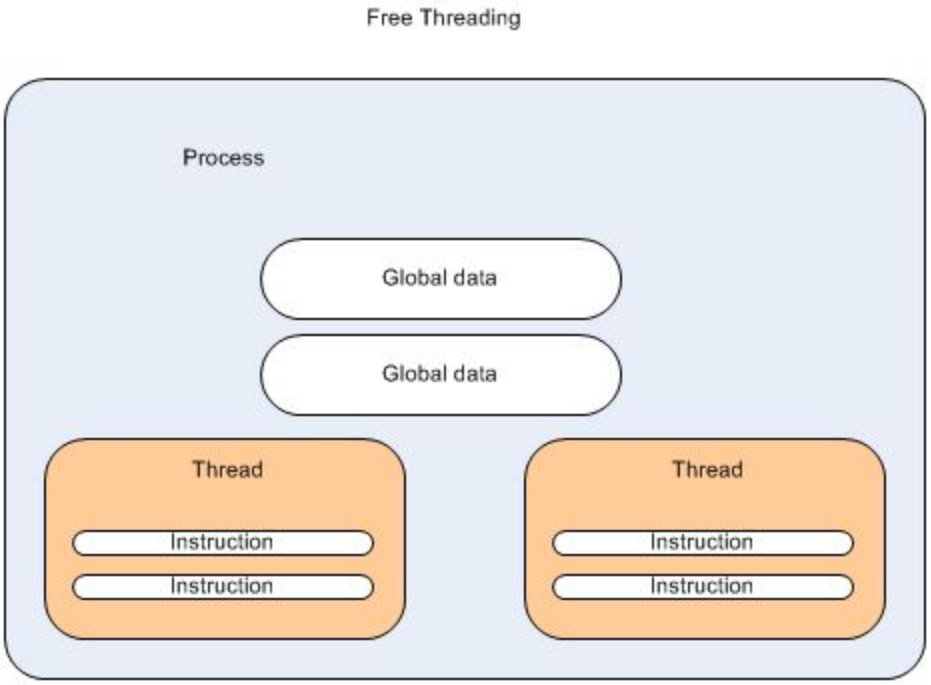
7. Boost.Thread

7.1. Giới thiệu

- Boost.Thread là thư viện cho phép ta sử dụng đa luồng xử lý vùng dữ liệu chung trong C++. Nó cung cấp các lớp và hàm cho phép quản lý các luồng, cùng với việc đồng bộ hóa dữ liệu giữa các luồng

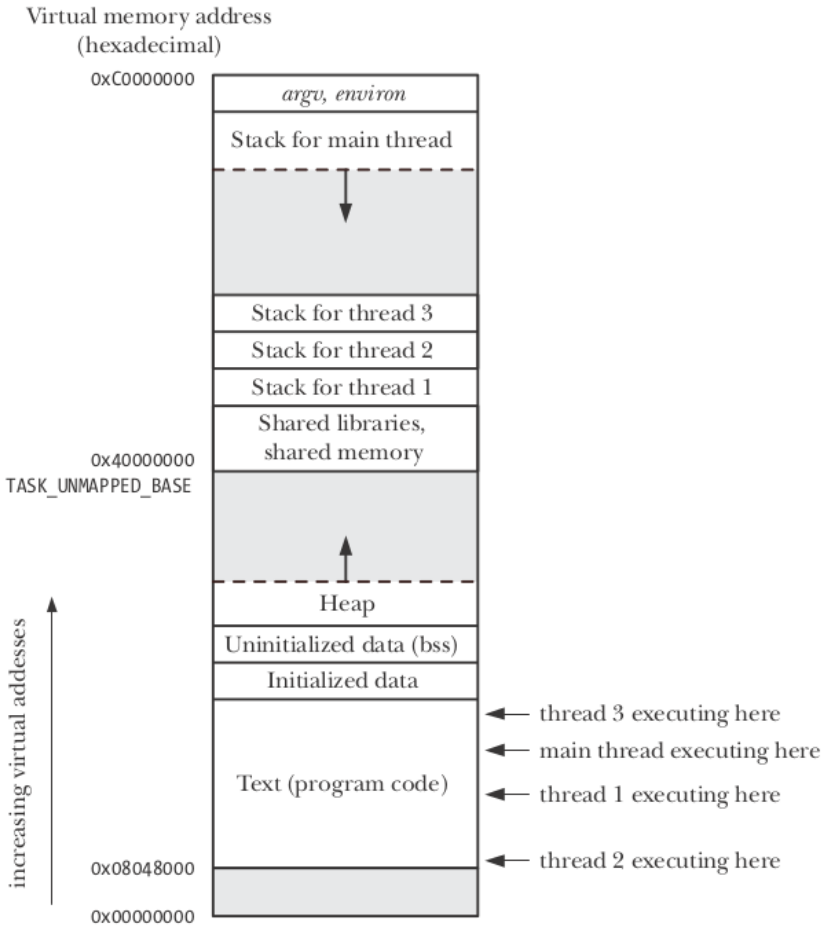
và cung cấp các bản sao dữ liệu cho từng luồng riêng lẻ. Boost.Thread cho phép chương trình thực hiện đồng thời nhiều tác vụ, và giúp quá trình tương tác với người dùng không bị gián đoạn, lập trình song song và là kĩ thuật không thể thiếu trong các ứng dụng về mạng.

- Thư viện Boost.Thread ban đầu được thiết kế và viết bởi Wiliam E. Kempf.



7.2. Vấn đề của việc đa luồng

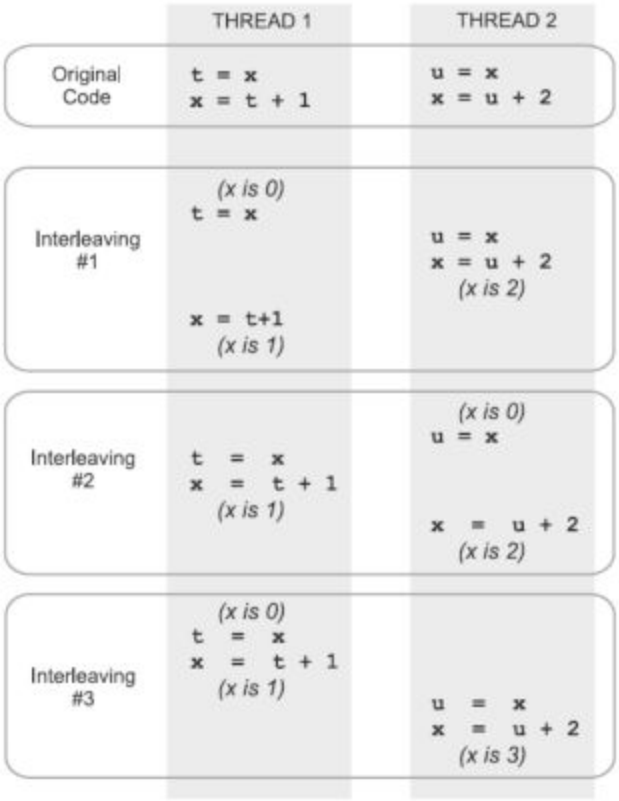
- Quá nhiều Threads



- ❖ Phân vùng một số lượng cố định công việc trong số quá nhiều threads làm cho mỗi thread quá ít công việc đế xử lý.
- ❖ Có quá nhiều luồng thì phần mềm đồng thời phải chịu phí tổn từ việc chia sẻ tài nguyên phần cứng cố định.
- ❖ Giải pháp tốt nhất là hạn chế số lượng các luồng "có thể chạy được" tới số lượng các luồng phần cứng.

❑ Cùng truy cập một vùng dữ liệu để xử lý

- ❖ Ví dụ: Có 2 thread cố gắng gắn giá trị cho 1 biến x cùng chia sẻ vùng nhớ, được khởi tạo giá 0.



- Hình trên là kết quả mà x có thể nhận được. Ta thấy dữ liệu có thể không có được đúng đắn như trình tự. Để xử lý các vấn đề này, Boost.Thread có một định nghĩa là đồng bộ hóa Thread và cần sử dụng nhiều kiểu đối tượng khác nhau và phương thức riêng biệt để xử lý.
- Một vài kiểu đối tượng xử lý đồng bộ hóa Thread:
  - Mutex: là một đối tượng đồng bộ hóa thread mà cho phép chỉ một thread truy cập đế vùng dữ liệu chung tại một thời điểm. Khi một thread cần tuy cập đến vùng dữ liệu chung, thì thread đó trước hết phải “lock” mutex, để đảm bảo chỉ có mỗi thread được truy cập. Nếu có bất kì thread khác muốn truy cập vào vùng dữ liệu đó thì phải chờ đến khi thread đầu tiên “unlock” mutex thì mới có quyền được truy cập vào.
    Ví du:
    mutex.lock(); // chỉ thread hiện tại được truy cập
    std::cout << "Thread " << std::endl;
    mutex.unlock(); // sau khi hàm unlock() kết thúc các thread khác có thể truy cập được.

  - Lock_guard: là một đối tượng được ép kiểu mutex tự động gọi lock() và unlock() trong phương thức khởi tạo và hủy của nó.
    Ví dụ:
    boost::lock_guard<boost::mutex> lock{mutex};
    std::cout << "Thread " << std::endl;

➢ Ngoài ra còn có các đối tượng Atomic, unique_lock, shared_lock, condition_variable_any cung cấp thêm một vài phương thức hữu ích.

7.3. Tại sao lại sử dụng Boost.Thread?

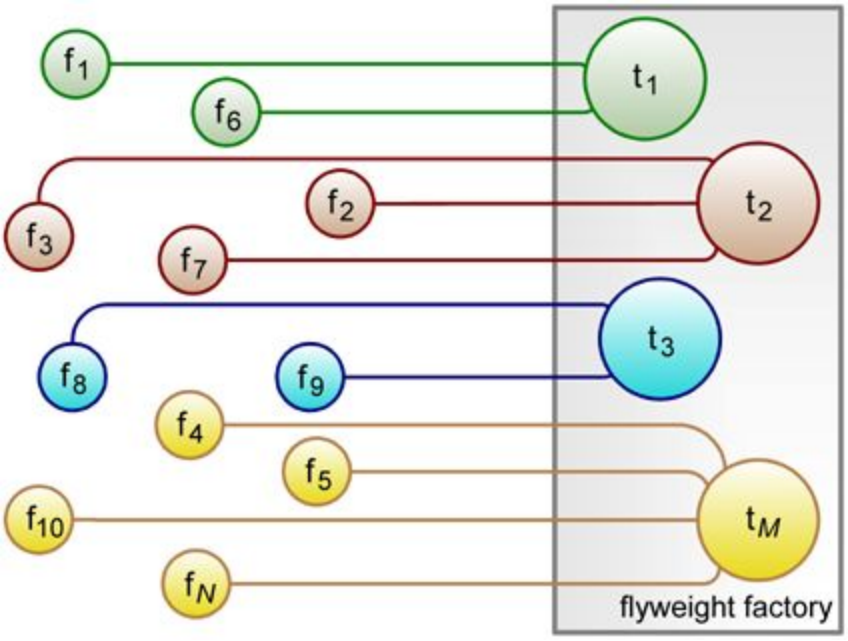
- ❖ Nền tảng độc lập
- ❖ Sử dụng ít tài nguyên hơn vì dùng chung không gian và bộ nhớ
- ❖ Trừu tượng hoá pthreads và WinAPI
- ❖ Là một C++ API hiện đại
- ❖ Cú pháp tương thích với std::thread C++11
- ❖ Thiết kế các chương trình đa nhiệm, mỗi phần của chương trình là một luồng, mỗi luồng sẽ thực thi độc lập một nhiệm vụ.

8. Boost.Flyweight

8.1. Giới thiệu

Boost.Flyweight là thư viện sử dụng mẫu hình thiết kế cho các biến khác nhau giữ giá trị giống nhau. Flyweight giúp tiết kiệm bộ nhớ khi nhiều đối tượng sử dụng chung dữ liệu. Cụ thể dữ liệu dùng chung không ở lưu ở trong đối tượng mà lưu ở chỗ khác, và đối tượng sẽ lưu biến trỏ đến dữ liệu này. Ta có thể dùng con trỏ thay cho flyweight để cài đặt mẫu hình thiết kế này.

Xét một chương trình phải quản lý một lượng lớn các đối tượng với kích thước đáng kể, có khả năng chương trình sẽ tốn bộ nhớ vượt mức cho phép. Khi dữ liệu đối tượng này là hằng, không đổi trong chương trình, kĩ thuật flyweight được sử dụng để tối ưu hóa bộ nhớ.



Giả sử, có N đối tượng giữ M giá trị. Nếu N lớn hơn M, sẽ có nhiều đối tượng có chung giá trị. Có thể giảm bộ nhớ bằng cách sử dụng class trong đối tượng chỉ tới giá trị dùng chung thay vì lưu trực tiếp giá trị trong đối tượng. Giá trị N/M càng lớn, lượng bộ nhớ tiết kiệm được càng cao.

Lợi ích của flyweight:

1. Tăng tốc độ xử lí khi khởi tạo đối tượng, vì không cần phải tạo mọi dữ liệu của đối tượng.
2. Tiết kiệm bộ nhớ vì sử dụng lại dữ liệu đã có chứ không tạo mới.

Ứng dụng đầu tiên của flyweight là chương trình xử lý văn bản. Mỗi kí tự trong văn bản chứa một lượng lớn dữ liệu: mã Unicode, font, kích thước, hiệu ứng, … Nếu mỗi kí tự đều lưu từng giá trị cụ thể, chương trình có thể không xử lý nổi hàng ngàn kí tự. Nhưng nếu ứng dụng flyweight, mỗi kí tự chỉ lưu giá trị chỉ tới dữ liệu; vì lượng thông tin như mã Unicode, font, … là có giới hạn, tức là M nhỏ. Khi đó chương trình có thể xử lý dễ dàng văn bản hàng trăm ngàn chữ trở lên.

8.2. Sử dụng

Để sử dụng Boost.Flyweight ta cần

```
#include <boost\flyweight.hpp>
```

Để dễ gọi ta sử dụng luôn namespace

```
using namespace ::boost;
using namespace ::boost::flyweights;
```

Ví dụ một struct người chơi game bình thường, với thư viện *string*

```
struct user_entry
{
	string first_name;
	string last_name;
	Int age;
};
```

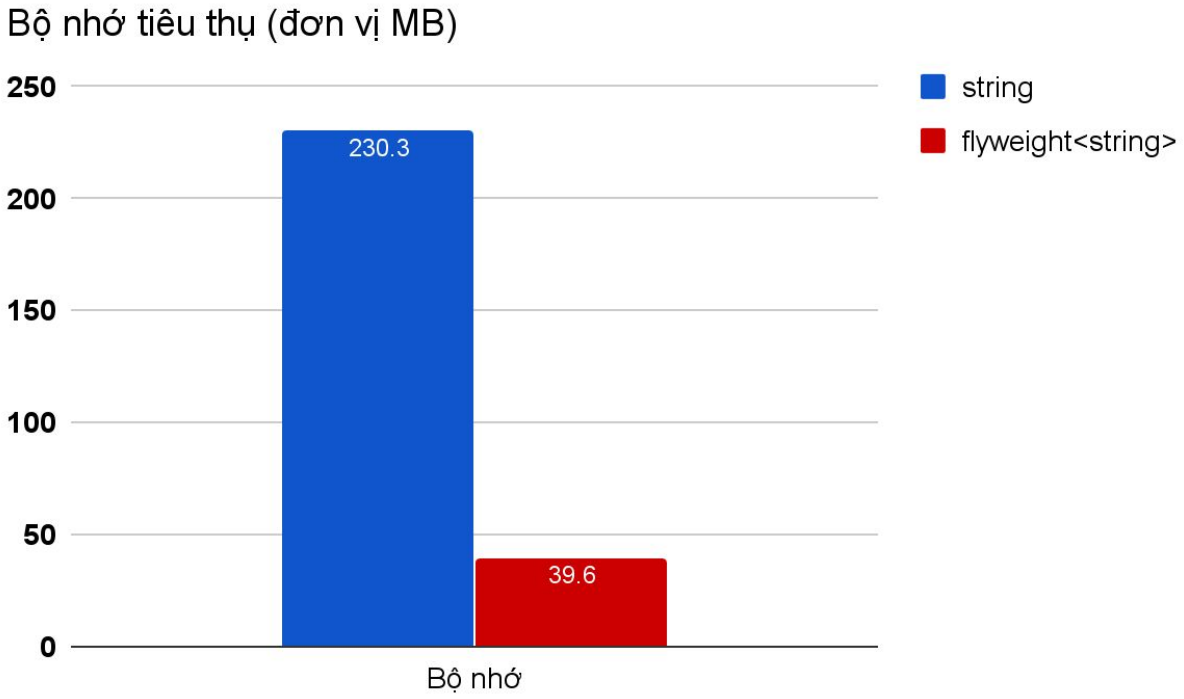
Ta nhận thấy tên có thể trùng, do đó có thể sử dụng flyweight

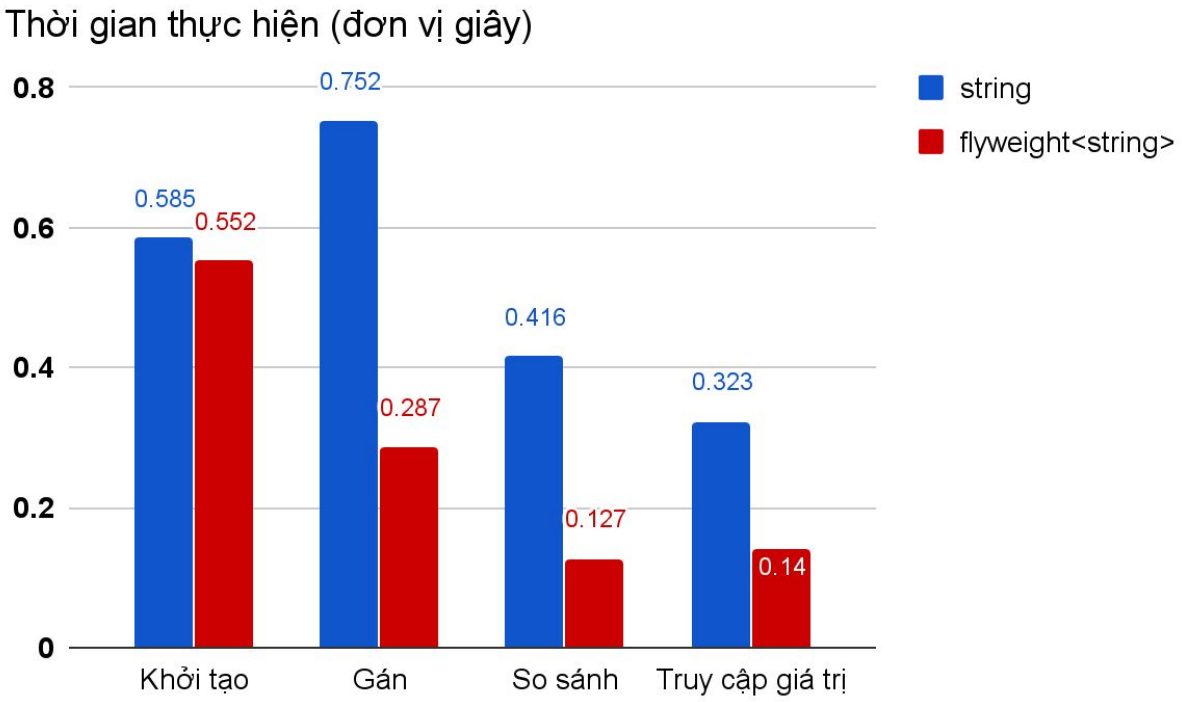
```
struct user_entry
{
	flyweight<string> first_name;
	flyweight<string> last_name;
	int age;
};
```

Quy tắc định nghĩa *flyweight<T>*. T là kiểu biến. T phải thỏa mãn 3 tính chất: gán được, so sánh được, và sử dụng được với Boost.Hash. Ví dụ như kiểu int, float, string.

8.3. Test hiệu suất

Bài test xử lý văn bản sử dụng string trong thư viện chuẩn và flyweight<string>.
Code http://www.boost.org/doc/libs/1_65_1/libs/flyweight/example/perf.cpp.
File text http://www.gutenberg.org/ebooks/2000.txt.utf-8.
Thực hiện test với Visual Studio 2013, Windows 10, Core i5 7200U, 4GB ram





Tài liệu tham khảo

http://www.boost.org/
https://theboostcpplibraries.com/
https://en.wikipedia.org/wiki/Boost_(C%2B%2B_libraries)
https://www.openhub.net/p/boost/analyses/latest/languages_summary
http://think-async.com/Asio/WebHome