**CPP.Assignment 12 – Advance features in C++**

**Exercise 1:**

Em đang làm mô phỏng lại shared\_ptr mà chưa đúng được tính chất của nó.

Trong class em có sử dụng 2 con trỏ và 1 biến đếm:

+ 1 con trỏ m\_ptr để trỏ đến đối tượng. ( dùng shallow copy)

+ 1 con trỏ m\_refs để chia sẻ reference count (dùng shallow copy)

+ 1 biến đếm m\_ins để đếm số thực thể thực sự đang trỏ đến (instances).

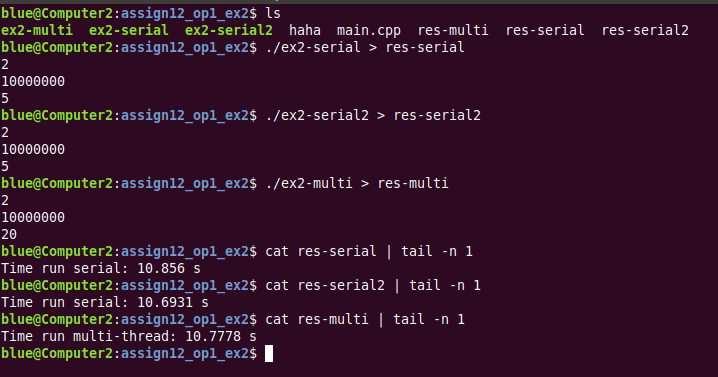
**Exercise 2:**

Bài toán em lập trình chạy cả cho một luồng và đa luồng, test với a = 2, b = 10 000 000, n = 20. Đa luồng em chia đoạn [a, b] thành n đoạn bằng nhau và cho mỗi một luồng chạy một đoạn. Kết quả (cả về thời gian và kết qua in ra đều khá giống nhau). Em luôn kỳ vọng đa luồng phải nhanh hơn một luồng, em đã cố gắng cải thiện nhưng chưa được.

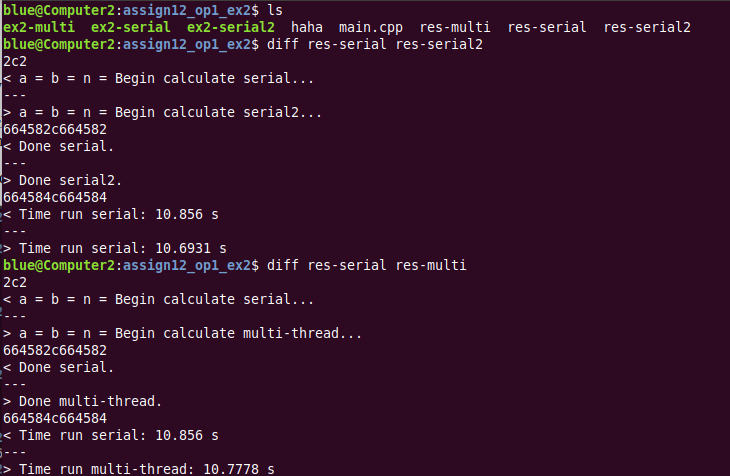
Trên Ubuntu lệnh biên dịch đa luồng là:

g++ main.cpp -lpthread

Kết quả: Thời gian tương đương nhau: Khoảng 10.7s



Kết quả ghi vào file cũng giống nhau cả về số lượng và thứ tự:



**Exercise 3:**

Em tạo thư viện động (dynamic library hay shared library (on Linux)) bởi vì tính linh hoạt của nó. Sau khi liên kết mà không có thay đổi gì trong chương trình main, chúng ta có thể tùy chỉnh (update) implementation của các hàm trong thư viện và chỉ cần build lại thư viện, đặt đúng chỗ cũ là những update này sẽ được chương trình main tải vào khi khởi chạy (không cần re-build hoặc re-link)

Tuy nhiên, ở đây có sự khác nhau khi tổ chức để link với hàm main giữa các hàm thường và hàm template, vì vậy em tách bài ra thành 2 mục riêng cho mỗi loại.

+ **Với hàm thường**, ta phải xác định kiểu trước, muốn thực hiện cho kiểu nào, ta phải overload lại cho kiểu đó.

Chương trình có 3 files: main.cpp để test và 2 file định nghĩa thư viện (tuan): tuan.h và tuan.cpp

Điều đặc biệt ở đây là chúng ta thậm chí còn không cần thêm dòng #include “tuan.h” vào trong tệp tuan.cpp. (bởi vì định nghĩa hàm sẽ được load vào lúc chuẩn bị chạy)

Trên Ubuntu, để build thư viện và link với hàm main, chúng ta cần thực hiện các lệnh:

*g++ -c -fPIC -o tuan.o tuan.cpp* // build file *tuan.cpp* ra object file *tuan.o*

*g++ -shared -o libtuan.so tuan.o* // build shared library (*libtuan.so*) từ objec file *tuan.o*

*g++ -L. -o main main.cpp -ltuan* // link hàm *main* với shared library *tuan* (libtuan.so)

Kiểm tra link thành công chưa:

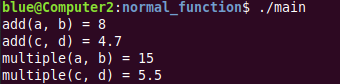


Kết quả libtuan.so đã nằm trong danh sách thư viện liên kết với hàm main.

Cuối cùng là cài đặt biến môi trường (em chọn đây là cách dễ nhất) để hàm main có thể tìm kiếm shared library và load vào trước khi chạy chương trình:

*export LD\_LIBRARY\_PATH=.:$LD\_LIBRARY\_PATH*

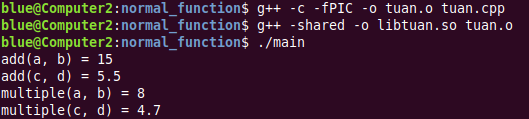
Kết quả chạy:



Hoàn toàn không thay đổi hàm main, thay đổi implementation của hàm cộng thành nhân, và nhân thành implementation của hàm cộng (chỉ để chứng minh dù không ý nghĩa lắm):

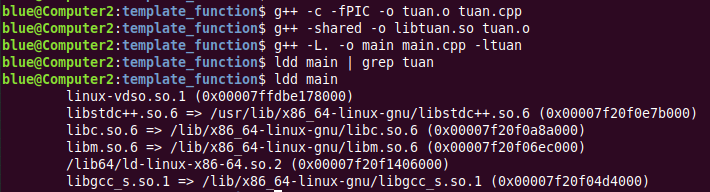
Chỉ build lại thư viện (2 lệnh đầu) và chạy lại chương trình:

Kết quả chạy: Update của triển khai hai hàm được load vào mà không cần biên dịch lại hàm main.

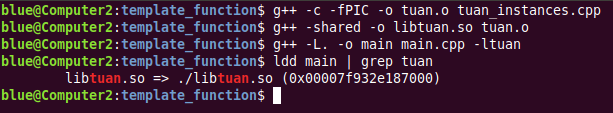


+ Đối với function template, vấn đề là template được triển khai tại thời điểm biên dịch, và chỉ được triển khai thành normal function để biên dịch khi được sử dụng (ở đây là trong hàm main), nên nếu tách rời khai báo và định nghĩa function template thành 2 tệp như trên thì:

* Do 2 tệp main.cpp và tuan.cpp được biên dịch độc lập, template được triển khai khi biên dịch main.cpp (chỉ có khai báo hàm), trong khi biên dịch tệp tuan.cpp thì template không được triển khai thành định nghĩa hàm tương ứng dẫn đến lỗi khi liking.
* Do đó, phải gộp 2 nội dung 2 tệp ở đâu đó:
  + Không thể trong main.cpp sẽ phải biên dịch lại mỗi khi tuan.cpp được cập nhật.
  + Cũng không thể #include “tuan.cpp” trong tệp tuan.h vì tuan.h được #include trong main.cpp, khi đó hàm main sẽ có đủ mọi thành phần cần thiết và trở thành một phiên bản cố định, thực tế là nó không link với shared library nữa:

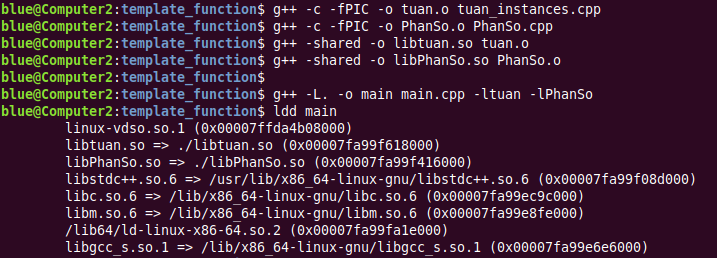


* + Vậy chỉ có thể #include “tuan.h” trong tệp tuan.cpp, nhưng lại gặp lỗi linking ở trên. Để giải quyết vấn đề này, chúng ta buộc phải khởi tạo cho function template một cách tường minh cho kiểu ta muốn ta muốn sử dụng (explicitly instantiate template functions/classes) với từ khóa tempalate và thay thế bằng kiểu chúng ta muốn sử dụng.
  + Có thể đặt khởi tạo này trong file tuan.cpp, nhưng để dễ theo dõi và cập nhật khi muốn sử dụng template cho đối tượng mới, chúng ta có thể tạo một file thứ 4, tuan\_instances.cpp #include 2 file tuan.h và tuan.cpp, cùng các khởi tạo tường minh cho template (chỉ việc thêm từ khóa template vào trước prototype của normal function ở phần trước, nhưng lợi thế ở của template ở đây là chúng ta chỉ cần 4 khai báo như vậy cho một kiểu đối tượng mới là đã có thể sử dụng.



Chỉ có file đầu vào cho lệnh đầu tiên được thay đổi (từ tuan.cpp thành tuan\_instances.cpp) và kết quả là đã linking thành công shared library với hàm main. Kết quả không có gì thay đổi.

Tạo thêm lớp PhanSo có ghi đè phép toán cộng và nhân, và khai báo khởi tạo tường minh cho 2 phép toán với đối tượng PhanSo ở file tuan\_instances.cpp. Biên dịch thành shared library libPhanSo.so. Kết quả chúng ta có hai shared library liên kết với main và có thể thoải mái tùy chỉnh implementation.



Ta thấy làm main đã được link với 2 thư viện libtuan.so và libPhanSo.so (thực tế là nếu không sử dụng hai hàm add và multiple của template thì thư viện libtuan.so sẽ không được liên kết).

