C++ dlopen mini HOWTO

IsottonAaron

aaron (at) isotton.com

도운이

김경태

nullnull4u (at) hanmail.net

`dlopen` API를 이용하여 동적으로 C++ 함수와 클래스를 적재하는 방법을 설명한다.

고친 과정

고침 1.002002-06-19고친이 AI

copyright와 license부분을 앞쪽으로 옮겼다. 용어에 관한 부분을 추가했다. 약간 변화가 있었다.

고침 0.972002-06-19고친이 JYG

문장과 문법을 약간 가다듬었다.

고침 0.962002-06-12고친이 AI

참고 문헌을 추가했다. 외부 함수와 변수들의 설명을 수정했다.

고침 0.952002-06-11고친이 AI

아주 약간 개선.

고침 0.9 2002-06-10고친이 AI

초안 작성.

1. 소개

Unix C++ 프로그래머 사이에서 자주 발생하는 질문은 `dlopen`API를 이용하여 C++ 함수와 클래스를 적재하는 방법에 관한 것입니다.

사실 이것은 항상 간단한 것만은 아니기 때문에,약간의 설명이 필요합니다. 이 mini HOWTO 에서 그것에 관한 내용을 다루고 있습니다.

이 문서를 이해하기 위해서는 C, C++,그리고`dlopen` API 에 대해서 어느 정도 알고 있어야 할 것입니다.

이 HOWTO의 원문은 http://www.isotton.com/howtos/C++-dlopen-mini-HOWTO/ (http://www.isotton.com/howtos/C++-dlopen-mini-HOWTO/)에 있습니다.

1.1. Copyright and License

1.2. 경고(Disclaimer)

이 문서의 내용으로 인한 책임은 지지 않습니다. 여기에 쓰여진 개념이나 예제 그리고 정보를 사용하여 발생한 문제에 대한 책임은 모두 당신의 책임입니다. 여기에는 당신의 시스템에 피해를 줄 수 있는 오류나 부정확한 것들이 있을수도 있습니다. 주의하여 주시기 바라며, 저는 여기에 어떠한 책임도 지지 않을 것입니다.

모든 저작권은 구체적으로 언급하지 않았다면, 그것들 각각의 소유자가 가지고 있습니다. 이 문서에 있는 용어의 사용이 어느 등록상표나 서비스 마크의 효력에 영향을 끼치는 것으로 간주해서는 안 됩니다. 특별한 제품이나 브랜드를 지명하는 것이 상품등의 추천으로 보여서는 안 됩니다.

1.3. 도와 주신분

이 문서에서, 나는 이 분들께 감사하게 되어 기쁘게 생각합니다.

- Joy Y Goodreau 씨<`joyg (at) us.ibm.com`> 는 교정에 도움을 주셨습니다.
- D. Stimitis 씨<`stimitis (at) idcomm.com`> 는 formatting 과 name mangling에 대한 몇가지 이슈를 지적해주셨고, `extern "C"`에 대한 몇가지 난해한 점을 지적해주셨습니다.

1.4. Feedback

이 문서에 대한 반응은 언제나 환영입니다. 당신이 추가했으면 하는 것, 의견, 비판을 다음 주소로 보내주시기 바랍니다 <`aaron@isotton.com`>.

1.5. 이 문서에서 쓰인 용어들

`dlopen` API

`dlclose`, `dlerror`, `dlopen`,그리고 `dlsym` 함수는 `dlopen(3)` 매뉴얼 페이지에서 설명하고 있습니다.

이 글에서“`dlopen`”이라고 썼을 때는, `dlopen`함수 하나를 지칭하는 것이고, “`dlopen` API” 라고 썼을 때에는, **API** 전체를 지칭하는 것입니다.

2. 문제

프로그램이 실행될때, 라이브러리를 적재해야 할 때가 가끔씩 있을 것입니다. 당신이 프로그램에 들어가는 플러그인이나 모듈을 만들고 있을때 이러한 일은 종종 발생합니다.

C언어에서, 동적으로 라이브러리를 적재하는 것은 매우 간단합니다. (`dlopen`, `dlsym` 그리고 `dlclose`를 호출하는 것만으로 충분합니다) C++에서는 약간 더 복잡합니다. C++ 라이브러리를 동적으로 적재하는 것이 어려운 이유중에 일부분은name mangling 때문이고, 일부분은 `dlopen` API가 C를 염두에 두고 만들어졌기 때문에 class를 적재하는 적당한 방법을 제공하지 못하기 때문입니다.

C++에서 라이브러리를 적재하는 방법에 대해 설명하기 전에, name mangling에 대해서 자세히 살펴보고, 문제를 분석해 봅시다. 비록 당신이 name mangling에 관심이 없더라도, 나는 당신이 그것에 대한 설명을 읽기를 바랍니다. 왜냐하면 그것은 당신이 왜 문제가 발생하고 어떻게 해결해야 하는지를 이해하는데 도움을 주기 때문입니다.

2.1. Name Mangling

모든 C++프로그램(혹은 라이브러리나 Object 파일)에서, 모든 non-static 함수는 이진 파일에서 **symbol**로 표현됩니다. 이러한 symbol들은 프로그램(혹은 라이브러리나 Object 파일)에서, 유일하게 함수를 확인하는, 특수한 문자열입니다.

C에서는, symbol의 이름은 함수의 이름과 동일합니다. `strcpy`의 symbol은 `strcpy`입니다. C에서는 두개의 non-static 함수가 같은 이름을 가질 수 없기 때문에 이것이 가능합니다.

C++은 오버로딩을 허용하고 (같은 이름을 가지지만 인자가 다른 함수를 정의할 수 있다), C가 가지고 있지 않은 여러가지 특징들 — Class, member function, exception specifications — 을 가지고 있기 때문에, 그냥 단순히 함수 이름을 symbol 이름으로 쓸 수는 없습니다. 이 문제를 해결하기 위해서, C++에서는 **name mangling**(이름 엉망으로 만들기)이라는 것을 사용합니다. 이것은 함수이름과 모든 필요한 정보 모두를(인자의 크기나 갯수와 같은) 컴파일러만이 알아볼 수 있는 이상한 문자열로 바꿔버립니다. 예를 들어 `foo`라는 함수가 있다면, name mangling에 의해 `foo@4%6^`로 바뀔 것입니다.

name mangling과 관련하여 생기는 문제점의 하나는 C++ 표준 (현재는 [ISO14882])이 그 방법을 정의해 놓지 않았다는 점입니다. 이것은 모든 컴파일러들마다 자기만의 방법 으로 name mangling을 한다는 것을 의미합니다. 어떤 컴파일러는 name mangling 알고리즘이 버전에 따라 다르기도 합니다. (g++ 2.x 와 3.x에서 뚜렷하게 드러납니다.) 비록 당신이 특정한 컴파일러가 어떻게 이름을 바꾸는지 이해했다고 해도(그래서 `dlsym`을 통해 그 함수를 적재할 수 있게 된다고 해도), 그것은 아마 그 특정한 컴파일러에서만 효과가 있을 것이고, 다음 버전의 컴파일러에서는 이미 사용할 수 없을지도 모릅니다.

2.2. Classes

`dlopen` API 의 또다른 문제는, 이들이함수를 적재하는 것만을 지원하고 있다는 점입니다. 하지만 C++에서는 라이브러리가 종종 당신이 프로그램에서 쓰고자 하는 Class를 노출시키고

는 합니다. 분명히, 클래스를 사용하기 위해서 당신은 그것의 인스턴스를 만들어야 하지만, 그것은 쉽지 않습니다.

3. 해결 방법

3.1. extern "C"

C++에는 `extern "C"`라는, C binding으로 함수를 정의하는 특별한 키워드가 있습니다. `extern "C"`로서 선언된 함수는 C처럼 함수의 이름을 symbol의 이름으로 사용합니다. 이러한 이유로, 멤버함수가 아닌 함수들만이 `extern "C"`로서 선언될 수 있고, 이러한 함수들은 오버로딩을 할 수 없습니다.

이런 심한 제한이 있지만, `extern "C"`함수는 C 함수처럼 `dlopen`을 써서 동적으로 적재할 수 있기 때문에 매우 유용합니다.

이것은 `extern "C"`로 선언된 함수가 C++ 코드를 포함할 수 없다는 것을 의미하는 것이아닙니다. 이런 함수는 어느 종류의 인자라도 받을 수 있고, C++의 특징을 쓸 수 있습니다.

3.2. 함수를 적재하는 법

C++에서 함수는 `dlsym`을 통해 C처럼 적재됩니다. 당신이 적재하고자 하는 함수는 symbol의 이름이 엉망으로 되지 않도록(name mangling이 이루어지지 않도록), `extern "C"`로서의 자격을 갖춰야 할 것입니다.

예 **1.** 함수를 적재하기

main.cpp:

```
#include <iostream>
#include <dlfcn.h>


int main() {
    using std::cout;
    using std::cerr;

    cout << "C++ dlopen demo\n\n";

    // open the library
    cout << "Opening hello.so...\n";
    void* handle = dlopen("./hello.so", RTLD_LAZY);

    if (!handle) {
```

```
        cerr << "Cannot open library: " << dlerror() << '\n';
        return 1;
    }

    // load the symbol
    cout << "Loading symbol hello...\n";
    typedef void (*hello_t)();
    hello_t hello = (hello_t) dlsym(handle, "hello");
    if (!hello) {
        cerr << "Cannot load symbol 'hello': " << dlerror() <<
            '\n';
        dlclose(handle);
        return 1;
    }

    // use it to do the calculation
    cout << "Calling hello...\n";
    hello();

    // close the library
    cout << "Closing library...\n";
    dlclose(handle);
}
```

hello.cpp:

```
#include <iostream>

extern "C" void hello() {
    std::cout << "hello" << '\n';
}
```

`hello`라는 함수는 `hello.cpp`에서 `extern "C"`로 선언되었습니다. 이것은 `main.cpp`에서 `dlsym`을 사용하여 적재할 수 있습니다. 함수는 `extern "C"` 로서의 자격을 갖추어야 합니다. 그렇지 않다면 우리는 hello 함수의 symbol의 이름을 알 수 없을테니까요.

주의

extern "C"의 선언에는 두가지의 다른 형태가 있습니다: 하나는 위에서 썼던extern "C"의 방법이고, 또 다른 하나는 extern "C" { … }형태로 중괄호 사이에 선언이 들어가는 방법입니다. 첫번째(inline)형태는 extern 연결과 C 언어 연결을 갖습니다. 두번째 형태는 C언어 연결에만 영향을 끼칩니다. 따라서 아래의 두 가지 선언 방법은 동일한 것입니다.

```
extern "C" int foo;
extern "C" void bar();

extern "C" {
     extern int foo;
     extern void bar();
 }
```

extern과 extern이 아닌 함수의 선언에는 차이가 없기때문에, 당신이 어떠한 변수도 선언하지 않는 한 문제가 없습니다. 당신이 변수를 선언한다면,다음의 두 선언 방법은 같지 않다는 것을 명심해야 합니다.

```
extern "C" int foo;
extern "C" {
    int foo;
}
```

좀 더 자세한 설명을 원하시면, paragraph 7에 주의를 기울여 [ISO14882]의 7.5를 읽어보시거나, [STR2000]의 paragraph 9.2.4.를 참조해주십시오.

외부 변수로 무언가를 하기 전에, see also부분에 적혀 있는 문서들을 정독해 주시기 바랍니다.

3.3. 클래스를 적재하는 법

클래스를 적재하는 것은 좀 더 어렵습니다. 왜냐하면 우리는 클래스의인스턴스를 필요로 하지, 함수에 대한 포인터를 필요로 하는게 아니기 때문입니다.

클래스가 실행파일에 정의되어 있지 않은데다가,(몇몇 상황에서는) 클래스의 이름조차도 알 수 없기 때문에, 우리는 new 를 이용하여 클래스의 인스턴스를 생성할 수 없습니다.

이것은 다형성(polymorphism) 을 통해 해결할 수 있습니다. 우리는 기반 클래스, 가상의 멤버를 가지고 있는 인터페이스클래스를 실행파일내에 선언하고, 실제 구현부의 클래스를 모듈에 선언합니다. 일반적으로 인터페이스 클래스는 추상적입니다. (클래스가 순수가상함수(pure virtual function)을 가지고 있을때 클래스가 추상적이라고 합니다.)

클래스의 동적인 적재는 일반적으로 플러그인 — 명확하게 정의된 인터페이스를 보여주어야 하는 — 에 쓰이기 때문에, 우리는 어쨌거나 인터페이스 클래스와 거기서 파생된 구현부

클래스를 정의해야 합니다.

다음으로,모듈 안에**Class factory function**이라는 두개의 도움을 주는 함수를 추가로 선언해야 합니다. 이 함수 중 하나는 클래스의 인스턴스를 만들고 그것의 포인터를 반환하는 역할을 하고, 또 다른 하나의 함수는 factory에서 만들어진 함수의 포인터를 받아 그것(클래스의 인스턴스)를 파괴하는 역할을 합니다. 이 두 함수는 `extern "C"`의 자격을 가지고 있어야 합니다.

클래스를 모듈에서 쓰기 위해서,두개의 factory function을 우리가 hello함수를 적재했던 것처럼`dlsym`을 사용하여 적재하십시오.그럼 우리는 우리가 원하는 만큼의 인스턴스를 생성할수도 있고, 파괴할 수도 있습니다.

예 **2.** 클래스를 적재하는 법

여기서 우리는 일반적인다각형을 인터페이스로 하고, 삼각형을 구현부분으로 할 것입니다.

main.cpp:

```
#include "polygon.hpp"
#include <iostream>
#include <dlfcn.h>

int main() {
    using std::cout;
    using std::cerr;

    // load the triangle library
    void* triangle = dlopen("./triangle.so", RTLD_LAZY);
    if (!triangle) {
        cerr << "Cannot load library: " << dlerror() << '\n';
        return 1;
    }

    // load the symbols
    create_t* create_triangle = (create_t*) dlsym(triangle, "create");
    destroy_t* destroy_triangle = (destroy_t*) dlsym(triangle, "destroy");
    if (!create_triangle || !destroy_triangle) {
        cerr << "Cannot load symbols: " << dlerror() << '\n';
        return 1;
    }

    // create an instance of the class
    polygon* poly = create_triangle();

    // use the class
    poly->set_side_length(7);
        cout << "The area is: " << poly->area() << '\n';

    // destroy the class
    destroy_triangle(poly);
```

```
    // unload the triangle library
    dlclose(triangle);
}
```

polygon.hpp:

```
#ifndef POLYGON_HPP
#define POLYGON_HPP

class polygon {
protected:
    double side_length_;

public:
    polygon()
        : side_length_(0) {}

    void set_side_length(double side_length) {
        side_length_ = side_length;
    }

    virtual double area() const = 0;
};

// the types of the class factories
typedef polygon* create_t();
typedef void destroy_t(polygon*);

#endif
```

triangle.cpp:

```
#include "polygon.hpp"
#include <cmath>

class triangle : public polygon {
public:
    virtual double area() const {
        return side_length_ * side_length_ * sqrt(3) / 2;
    }
};


// the class factories

extern "C" polygon* create() {
    return new triangle;
}

extern "C" void destroy(polygon* p) {
    delete p;
```

```
}
```

클래스를 적재할때 주의해야 할 점이 몇가지 있습니다:

- 당신은 인스턴스를 생성하는 함수와 파괴하는 함수를 모두 제공해야 합니다. 또 당신이 인스턴스를 파괴할때에는 실행파일 내에서 `delete`를 이용해서 지우지 마시고 ,항상 모듈로 넘겨서 주시기 바랍니다. 이는 C++ 에서 `new`와 `delete` 가 오버로딩 될 수 있기 때문입니다. 이 경우에 서로 맞지 않는 `new`와 `delete` 가 호출될 수 있고, 그렇게 되면 메모리 누수나 segmentation fault가 일어날 수도 있습니다.이것은 모듈과 실행파일을 링크할때 서로 다른 표준 라이브러리를 사용했을 경우에도 적용됩니다.
- 인터페이스 클래스의 소멸자는 구현부 클래스에 소멸자가 없는경우를 제외하고는 가상(virtual)이어야 합니다. 그렇지 않으면 그것은 호출되지 않을 것이고, 메모리 누수나 segmentation fault가 일어날 수도 있습니다.

4. See Also

- `dlopen(3)`의 매뉴얼 페이지를 보십시오. `dlopen`API의 목적과 용도를 설명합니다.
- James Norton 씨가 Linux Journal (http://www.linuxjournal.com/)에 기고했던 글 *Dynamic Class Loading for C++ on Linux* (http://www.linuxjournal.com/article.php?sid=3687)
- `extern "C"`, 상속, 가상함수, `new` 와 `delete`에 대해 나와있는 당신이 즐겨보는 C++ reference. 전 [STR2000]을 추천합니다.
- [ISO14882]
- Program Library HOWTO (http://www.dwheeler.com/program-library)에서는 당신에게 정적, 공유,동적적재 라이브러리와 그러한 것들을 만드는 방법을 설명해 줄 것입니다. (번역된 문서는 http://kldp.org/HOWTO/html/Program-Library-HOWTO/ (http://kldp.org/HOWTO/html/Program-Library-HOWTO/) 에서 볼 수 있다.)

서지사항

ISO14482 *ISO/IEC 14482-1998 — The C++ Programming Language*. http://webstore.ansi.org/에서 PDF로 이용가능합니다.

STR2000 Stroustrup Bjarne *The C++ Programming Language*, Special Edition. ISBN 0-201-70073-5. Addison-Wesley.