

SE04 - Làm việc với thư viện trên Linux

i. Thư viện

- Thư viện là một tệp chứa các đoạn code (và dữ liệu) đã được biên dịch, sẽ được tích hợp vào một chương trình. Việc này giúp tiết kiệm thời gian xây dựng chương trình, biên dịch lại nhanh hơn và cập nhật dễ dàng hơn.
- Cấu trúc thư viện trong C/C++:
 - Các file header chứa các khai báo hàm và các định nghĩa cần thiết để truy cập các chức năng của thư viện. File này có extension “.h” trong C và được prefix ‘c’ trong C++.
 - Các file đối tượng chứa các hàm và dữ liệu đã được biên dịch, thường được tổng hợp thành file thư viện lưu trữ (.a) nếu là thư viện tĩnh và file đối tượng chia sẻ (.so) nếu là thư viện chia sẻ.
- C++ hỗ trợ nạp chồng phương thức, sẽ có trường hợp nhiều hàm có cùng nhưng khác đối số. Lúc này trình biên dịch sẽ sửa lại tên symbol của hàm khi biên dịch file đối tượng, kỹ thuật này gọi là “Name Mangling”. Có thể ngăn trình biên dịch thực hiện “Name Mangling” bằng cách thêm *extern “C”* vào trước các khai báo cần chặn “Name Mangling”. Ví dụ:

```
extern "C"
{
    int add(int a, int b);
    int sub(int a, int b);
    int mul(int a, int b);
    int divi(int a, int b);
}
```

- Thư viện có thể chia làm:
 - Static library - Thư viện tĩnh
 - Shared library – Thư viện chia sẻ, có thể được chia thành 2 dạng:
 - Dynamically link library
 - Dynamically load library

ii. Static library - Thư viện tĩnh

- Thư viện tĩnh là một tập hợp các tập tin đối tượng thông thường, các thư viện tĩnh thường có hậu tố là .a.
- Thư viện tĩnh cho phép khả năng liên kết tới các chương trình mà không cần phải biên dịch lại. Thư viện thường hữu ích đối với nhà phát triển thư viện muốn cho phép người dùng sử dụng thư viện mà không làm lộ mã nguồn thư viện.
- Sử dụng thư viện liên kết tĩnh với Netbeans IDE projects:
 - Đặt đường dẫn đến thư mục chứa file header (.h) của thư viện vào mục Project Properties => Build => C/C++ Compiler => General => Include Directories. Ngoài ra còn có thể đặt đường dẫn đến file header vào mục Project Properties => Build => C/C++ Compiler => General => Include Headers.
 - Đặt đường dẫn đến file thư viện (.a) vào mục Project Properties => Build => Linker => Libraries.

- Ví dụ tạo thư viện tĩnh:
 - Có file header `math_static.h` và file mã nguồn `math_static.cpp`:
 - File `math_static.cpp`

```

math_static.cpp
~/NetBeansProjects

1 int add(int a, int b){
2     return (a+b);
3 }
4
5 int sub(int a, int b){
6     return (a-b);
7 }
8
9 int mul(int a, int b){
10    return (a*b);
11 }
12
13 int divi(int a, int b){
14     return (a/b);
15 }

```

- File `math_static.h`

```

math_static.h
~/NetBeansProjects

1 int add(int a, int b);
2 int sub(int a, int b);
3 int mul(int a, int b);
4 int divi(int a, int b);

```

- Tạo file đối tượng `math_static.o` từ file `math_static.cpp` sử dụng `g++`:

```

home@home-pc:~/NetBeansProjects$ g++ -c math_static.cpp
home@home-pc:~/NetBeansProjects$ ls
duckdee_SE01.zip  file_io_2          lu112889rh0k.tmp  pointer           SE04.odt
duckdee_SE02.zip  find-text-in-file  math_static.cpp   SE01.odt
duckdee_SE03.zip  find_text_in_text  math_static.h     SE02.odt
file_IO           find_text_length   math_static.o     SE03.odt

```

- Tạo một file thư viện tĩnh `math_static.a` từ file đối tượng trên:

```

home@home-pc:~/NetBeansProjects$ ar rs math_static.a math_static.o
ar: creating math_static.a
home@home-pc:~/NetBeansProjects$ ls
duckdee_SE01.zip  file_io_2          lu112889rh0k.tmp  math_static.o    SE03.odt
duckdee_SE02.zip  find-text-in-file  math_static.a     pointer          SE04.odt
duckdee_SE03.zip  find_text_in_text  math_static.cpp   SE01.odt
file_IO           find_text_length   math_static.h     SE02.odt

```

- Để sử dụng thư viện tĩnh vừa tạo, ta thêm `#include <math_static.h>` vào đầu chương trình:

```

math_demo.cpp
~/NetBeansProjects
1 #include <math_static.h>
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     int x, y;
8     printf("Enter two numbers\n");
9     cin >> x >> y;
10
11     cout << x << " + " << y << " = " << add(x, y) << "\n";
12     cout << x << " - " << y << " = " << sub(x, y) << "\n";
13     cout << x << " * " << y << " = " << mul(x, y) << "\n";
14
15     if(y == 0){
16         cout << "Denominator is zero so can't perform division\n";
17         return 1;
18     }else{
19         cout << x << " / " << y << " = " << div(x, y) << "\n";
20     }
21
22     return 0;
23 }

```

- Tạo file đối tượng `math_demo.o` từ `math_demo.cpp`:

```

home@home-pc:~/NetBeansProjects$ g++ -I . -c math_demo.cpp
home@home-pc:~/NetBeansProjects$ ls
duckdee_SE01.zip  find-text-in-file  math_demo.o      pointer
duckdee_SE02.zip  find_text_in_text  math_static.a    SE01.odt
duckdee_SE03.zip  find_text_length   math_static.cpp  SE02.odt
file_IO           lu112889rh0k.tmp  math_static.h    SE03.odt
file_io_2         math_demo.cpp      math_static.o    SE04.odt

```

Tham số `-I` yêu cầu `g++` tìm kiếm các file header trong đường dẫn theo sau.

- Liên kết `math_demo.o` với `math_static.a` để tạo file thực thi `math_demo`.

```

home@home-pc:~/NetBeansProjects$ g++ -static -o math_demo math_demo.o math_static.a
home@home-pc:~/NetBeansProjects$ ls
duckdee_SE01.zip  find-text-in-file  math_demo.cpp    math_static.o  SE04.odt
duckdee_SE02.zip  find_text_in_text  math_demo.o      pointer
duckdee_SE03.zip  find_text_length   math_static.a    SE01.odt
file_IO           lu112889rh0k.tmp  math_static.cpp  SE02.odt
file_io_2         math_demo          math_static.h    SE03.odt

```

- Kiểm tra file thực thi có được liên kết tĩnh không sử dụng lệnh `file`:

```

home@home-pc:~/NetBeansProjects$ file math_demo
math_demo: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically link
ed, BuildID[sha1]=0951f7ad94c963ebb44fbc91749c7ded57f3e15a, for GNU/Linux 3.2.0, not
stripped

```

- Kiểm tra dung lượng file thực thi

```

-rwxrwxr-x 1 home home 2404192 Apr 10 17:34 math_demo

```

- Chạy file thực thi `math_demo`:

```

home@home-pc:~/NetBeansProjects$ ./math_demo
Enter two numbers
8
9
8 + 9 = 17
8 - 9 = -1
8 * 9 = 72
8 / 9 = 0

```

iii. Dynamically link library - Thư viện liên kết động

- Thư viện liên kết động là thư viện được tải bởi chương trình khi bắt đầu chạy. Khi một thư viện được cài đặt đúng cách, tất cả các chương trình bắt đầu sau đó đều tự động sử dụng nó.
- Các vấn đề khi sử dụng thư viện liên kết động:
 - Không tìm thấy thư viện: hệ thống không thể tìm thấy thư viện trong các thư mục hệ thống tiêu chuẩn. Xử lý bằng cách kiểm tra thư viện đã đặt vào các đường dẫn trên chưa hoặc thêm đường dẫn đến thư viện vào thư mục `/etc/ld.so.conf.d/`.
 - Phiên bản không tương thích: thư viện được biên dịch với một phiên bản khác của trình biên dịch. Xử lý bằng cách biên dịch lại thư viện với phiên bản phù hợp.
 - Kiến trúc không tương thích: thư viện được biên dịch cho kiến trúc khác với kiến trúc của hệ thống. Xử lý bằng cách cài đặt kiến trúc phù hợp với thư viện hoặc biên dịch lại thư viện cho phù hợp với kiến trúc hệ thống đang sử dụng.
 - Lỗi liên kết: chương trình không thể liên kết đến thư viện. Xử lý bằng cách biên dịch lại chương trình với các tham số phù hợp với thư viện sử dụng.
- Sử dụng thư viện liên kết động trên Linux:

- Ta có thể đặt thư viện vào các thư mục hệ thống tiêu chuẩn, sau đó chạy lệnh `ldconfig`:
`/usr/lib` chứa các thư viện là một phần của hệ thống lõi.

- `/usr/local/bin` chứa các thư viện được quản trị viên hoặc phần mềm bên thứ ba cài đặt vào

- `/lib` chứa các thư viện cần thiết để hệ thống khởi động và hoạt động bình thường

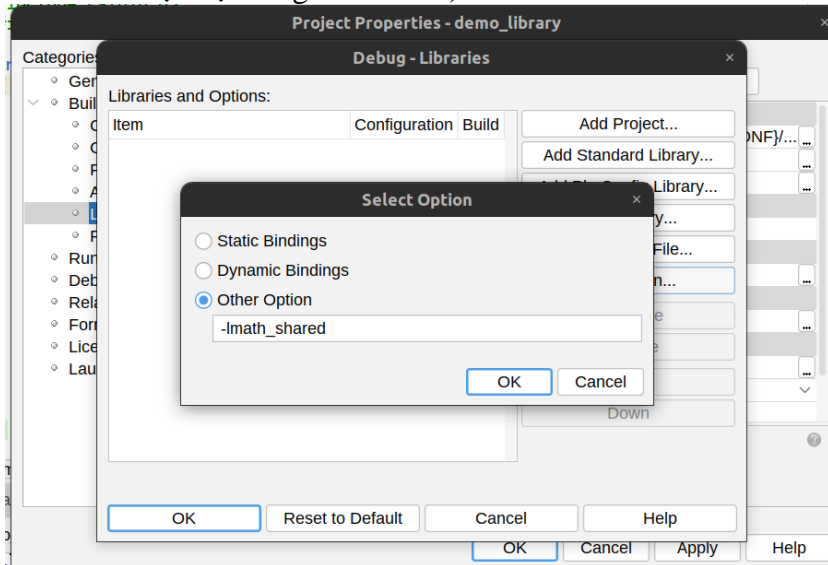
- `/lib64` tương tự với `/lib` nhưng chứa các thư viện dành cho kiến trúc 64-bit

Ngoài ra, mỗi distro của Linux sẽ có thêm các đường dẫn riêng tùy vào phiên bản. Ví dụ trên Ubuntu còn sử dụng đường dẫn `/usr/lib/x86_64-linux-gnu` để chứa các thư viện dùng cho kiến trúc x86_64.

- Ngoài các đường dẫn trên, người dùng còn có thể chỉ định thêm đường dẫn yêu cầu hệ điều hành tìm kiếm thư viện bằng cách thêm tập tin với nội dung là đường dẫn đến thư mục mới vào thư mục `/etc/ld.so.conf.d/`.
- Các cách trên yêu cầu thực hiện dưới quyền của super user dẫn đến trong một số hệ thống có thể không thực hiện được do không được cấp quyền super user, ta có thể sử dụng một số cách sau.
- Biên dịch chương trình với tham số `-L` cùng với đường dẫn đến thư mục chứa file thư viện
- Sử dụng biến môi trường `LD_LIBRARY_PATH` khi chạy chương trình để yêu cầu chương trình tìm kiếm thư viện tại các đường dẫn chứa trong biến `LD_LIBRARY_PATH`
- Thêm biến môi trường `LD_LIBRARY_PATH` chứa các đường dẫn đến thư mục chứa thư viện vào tệp cấu hình shell của người dùng hiện tại (`~/.bashrc` đối với Bash)
- Sử dụng thư viện liên kết động với Netbeans IDE project
 - Đặt đường dẫn đến thư mục chứa file header (.h) của thư viện vào mục Project Properties => Build => C/C++ Compiler => General => Include Directories. Ngoài ra còn có thể đặt

đường dẫn đến file header vào mục Project Properties => Build => C/C++ Compiler => General => Include Headers.

- Đặt đường dẫn đến file thư viện (.so) vào mục Project Properties => Build => Linker => Libraries.
- Ngoài ra còn có thể tạo option tại mục Project Properties => Build => Linker => Libraries hoặc thêm vào Project Properties => Build => Linker => Libraries => Complilatin Line => Additional Options với cấu trúc -l[tên thư viện] (chỉ hoạt động khi đã đặt thư viện vào thư mục hệ thống tiêu chuẩn).



- Ví dụ tạo và sử dụng thư viện liên kết động
 - Tạo file header math_shared.h và file mã nguồn math_shared.cpp với nội dung giống với ví dụ ở phần thư viện tĩnh.
 - Tạo file đối tượng từ math_shared.cpp:

```
home@home-pc:~/NetBeansProjects/math_shared$ g++ -Wall -fPIC -c math_shared.cpp
home@home-pc:~/NetBeansProjects/math_shared$ ls
math_shared.cpp  math_shared.h  math_shared.o
```

Tham số -fPIC yêu cầu g++ tạo code độc lập với vị trí (position-independent code).

- Tạo thư viện liên kết động libmath_shared.so

```
home@home-pc:~/NetBeansProjects/math_shared$ g++ -shared -o libmath_shared.so \math_shared.o
home@home-pc:~/NetBeansProjects/math_shared$ ls
libmath_shared.so  math_shared.cpp  math_shared.h  math_shared.o
```

- Để sử dụng thư viện liên kết động vừa tạo, ta thêm #include <math_dynamic.h> vào đầu chương trình.

```

math_demo.cpp
~/NetBeansProjects/math_shared
Save
1 #include <math_shared.h>
2 #include <iostream>
3
4 using namespace std;
5
6 int main()
7 {
8     int x, y;
9     printf("Enter two numbers\n");
10    cin >> x >> y;
11
12    cout << x << " + " << y << " = " << add(x, y) << "\n";
13    cout << x << " - " << y << " = " << sub(x, y) << "\n";
14    cout << x << " * " << y << " = " << mult(x, y) << "\n";
15
16    if(y == 0){
17        cout << "Denominator is zero so can't perform division\n";
18        return 1;
19    }else{
20        cout << x << " / " << y << " = " << divi(x, y) << "\n";
21    }
22
23    return 0;
24 }

```

- Đặt file libmath_static.so vào thư mục hệ thống tiêu chuẩn. Ví dụ đặt tại thư mục /lib/x86_64-linux-gnu sau đó chạy lệnh ldconfig.

```

home@home-pc:~/NetBeansProjects/math_shared$ sudo cp libmath_shared.so /lib/x86_64-linux-gnu
home@home-pc:~/NetBeansProjects/math_shared$ sudo ldconfig

```

Lệnh ldconfig tạo các liên kết và bộ đệm cần thiết đến thư viện liên kết động mới nhất tìm thấy trên thư mục thư viện tiêu chuẩn.

- Tạo file đối tượng math_demo.o từ file math_demo.cpp

```

home@home-pc:~/NetBeansProjects/math_shared$ g++ -I . -c math_demo.cpp
home@home-pc:~/NetBeansProjects/math_shared$ ls
libmath_shared.so  math_demo.cpp  math_demo.o  math_shared.cpp  math_shared.h  math_shared.o

```

- Tạo file thực thi, với tham số -l ám chỉ đến tên thư viện liên kết động vừa tạo.

```

home@home-pc:~/NetBeansProjects/math_shared$ g++ -o math_demo math_demo.o -lmath_shared
home@home-pc:~/NetBeansProjects/math_shared$ ls
libmath_shared.so  math_demo.cpp  math_shared.cpp  math_shared.o
math_demo          math_demo.o    math_shared.h

```

- Sử dụng lệnh ldd để kiểm tra thư viện liên kết động được sử dụng

```

home@home-pc:~/NetBeansProjects/math_shared$ ldd math_demo
linux-vdso.so.1 (0x00007fff15177000)
libmath_shared.so => /lib/x86_64-linux-gnu/libmath_shared.so (0x00007fc7adb96000)
libstdc++.so.6 => /lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007fc7ad800000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fc7ad400000)
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007fc7adaaf000)
/lib64/ld-linux-x86-64.so.2 (0x00007fc7adbb1000)
libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007fc7ada8f000)

```

- Kiểm tra dung lượng file thực thi

```

-rwxrwxr-x 1 home home 16744 Apr 10 19:24 math_demo

```


- Chạy file thực thi math_demo.

```
home@home-pc:~/NetBeansProjects/math_shared$ ./math_demo
Enter two numbers
5
3
5 + 3 = 8
5 - 3 = 2
5 * 3 = 15
5 / 3 = 1
```

iv. Dynamically Loaded Libraries – DLL

- Dynamically Loaded Libraries là các thư viện không được tải vào lúc khởi động chương trình.
- Trong Linux, các DLL được xây dựng dưới dạng tệp đối tượng tiêu chuẩn hoặc thư viện liên kết động tiêu chuẩn. Sự khác biệt chính là các DLL không được tải tự động vào thời điểm liên kết hoặc khởi động chương trình, mà có một API để mở thư viện, tra cứu ký hiệu, xử lý lỗi và đóng thư viện. Người dùng C sẽ cần khai báo file header <dlfcn.h> để sử dụng API này.
- Một số hàm được hỗ trợ bởi dlfcn.h
 - Hàm: void * dlopen(const char *filename, int flag);
 - Dùng để mở thư viện và chuẩn bị để sử dụng.
 - Tham số truyền vào:

filename: con trỏ đến chuỗi chứa tên thư viện được mở.

flag: là một trong các giá trị sau

- | | |
|----|---|
| RT | • Xử lý các ký hiệu không xác định khi code từ thư viện được thực |
| L | thi |
| D | |
| - | |
| L | |
| A | |
| Z | |
| Y | |
| RT | • Xử lý tất cả các ký hiệu không xác định trước khi dlopen() trả về |
| L | và thất bại nếu không thể thực hiện |
| D | |
| - | |
| N | |
| O | |
| W | |
| RT | • Các ký hiệu bên ngoài được xác định trong thư viện sẽ được cung |
| L | cấp cho các thư viện được tải sau đó |
| D | |
| - | |
| G | |
| L | |
| O | |
| B | |
| A | |
| L | |

v.

- Kết quả trả về: Một handle tương ứng với thư viện vừa mở được sử dụng bởi các hàm khác trong dlfcn.h nếu thành công. Ngược lại, hàm trả về con trỏ NULL

- Hàm: void * dlsym(void *handle, char *symbol);
 - Dùng để tìm kiếm giá trị của một ký hiệu trong thư viện đã mở tương ứng với “handle”
 - Tham số truyền vào:

handle: kết quả trả về từ hàm dlopen(), tương ứng với thư viện cần sử dụng.

symbol: chuỗi chỉ định ký hiệu cần tìm kiếm.

- Giá trị trả về: Con trỏ đến địa chỉ mà ký hiệu được tải trên memory nếu thành công. Ngược lại, trả về con trỏ NULL nếu thất bại.

- Hàm: int dlclose(void* handle);
 - Dùng để đóng thư viện tương ứng với handle.
 - Tham số truyền vào

handle: kết quả trả về từ hàm dlopen(), tương ứng với thư viện cần đóng.

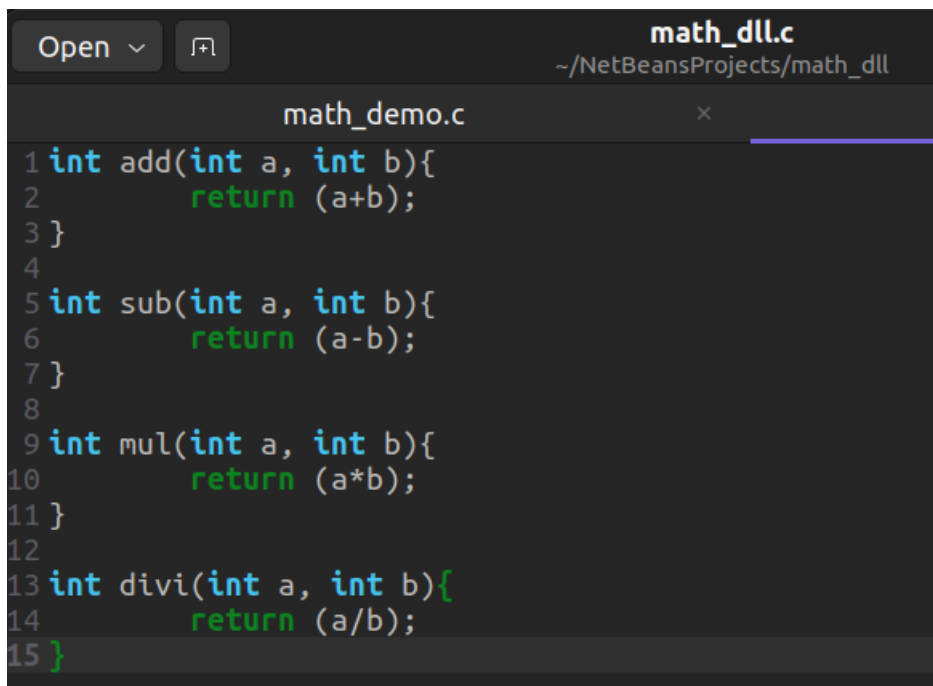
- Giá trị trả về: Trả về 0 nếu đóng thư viện thành công. Ngược lại trả về một giá trị khác 0.

- Sử dụng DLL với Netbeans IDE project:
 - Nếu đã đặt thư viện vào thư mục hệ thống tiêu chuẩn thì chỉ cần tạo option tại mục Project Properties => Build => Linker => Libraries hoặc thêm vào Project Properties => Build => Linker => Libraries => Complilatin Line => Additional Options với nội dung -ldl.
 - Đặt đường dẫn đến file thư viện (.so) vào mục Project Properties => Build => Linker => Libraries.
- Ví dụ sử dụng DLL:
 - Tạo file math_dll.c và math_demo.c với nội dung sau:
 - math_dll.h:


```
Open  math_demo.c  Save  ~/NetBeansProjects/math_dll
math_demo.c  math_dll.c
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <dlfcn.h>
4
5 //define function pointer type name math_f
6 typedef int (*math_f) (int, int);
7
8 int main()
9 {
10     const char *error;
11     void *handle;
12     math_f add, sub, mul, divi;
13
14     //Load library
15     handle = dlopen ("libmath_dll.so", RTLD_LAZY);
16
17     //Search for function by symbol in loaded library
18     add = (math_f) dlsym(handle, "add");
19     sub = (math_f) dlsym(handle, "sub");
20     mul = (math_f) dlsym(handle, "mul");
21     divi = (math_f) dlsym(handle, "divi");
22
23     int x, y;
24     printf("Enter two numbers\n");
25     scanf("%d%d",&x,&y);
26
27     printf("\n%d + %d = %d", x, y, add(x, y));
28     printf("\n%d - %d = %d", x, y, sub(x, y));
29     printf("\n%d * %d = %d", x, y, mul(x, y));
30
31     if(y==0){
32         printf("\nDenominator is zero so can't perform division\n");
33         exit(0);
34     }else{
35         printf("\n%d / %d = %d\n", x, y, divi(x, y));
36         return 0;
37     }
38     dlclose(handle);
39     return 0;
40 }
```

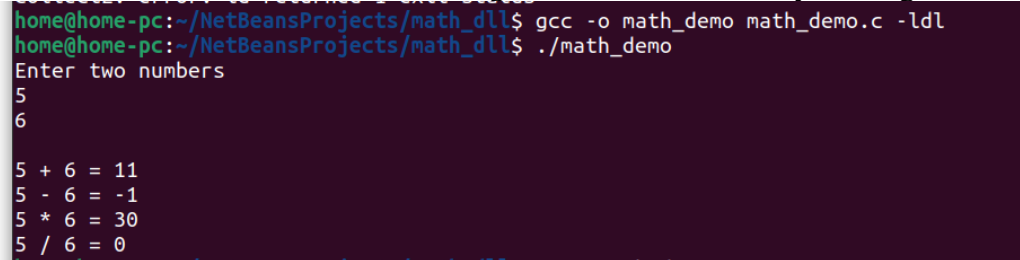
vi.

- math_dll.c:



```
math_demo.c
1 int add(int a, int b){
2     return (a+b);
3 }
4
5 int sub(int a, int b){
6     return (a-b);
7 }
8
9 int mul(int a, int b){
10    return (a*b);
11 }
12
13 int divi(int a, int b){
14     return (a/b);
15 }
```

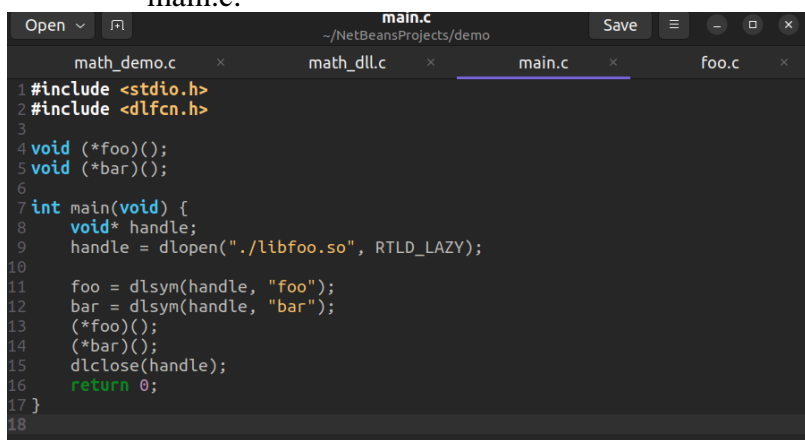
- Sau khi tạo thư viện liên kết động từ file math_dll.c, chạy chương trình math_demo:



```
home@home-pc:~/NetBeansProjects/math_dll$ gcc -o math_demo math_demo.c -ldl
home@home-pc:~/NetBeansProjects/math_dll$ ./math_demo
Enter two numbers
5
6

5 + 6 = 11
5 - 6 = -1
5 * 6 = 30
5 / 6 = 0
```

- Ví dụ khác nhau khi sử dụng các flag khác nhau của hàm dlopen()
 - Tạo 2 file main.c và thư viện libfoo.so:
 - main.c:



```
main.c
1 #include <stdio.h>
2 #include <dlfcn.h>
3
4 void (*foo)();
5 void (*bar)();
6
7 int main(void) {
8     void* handle;
9     handle = dlopen("./libfoo.so", RTLD_LAZY);
10
11     foo = dlsym(handle, "foo");
12     bar = dlsym(handle, "bar");
13     (*foo)();
14     (*bar)();
15     dlclose(handle);
16     return 0;
17 }
18
```

- foo.c:

```

1 #include <stdio.h>
2
3 void foo() {
4     printf("Hello from foo!");
5 }
6
7 void bar();

```

- Trong trường hợp load thư viện tải thư viện libmath_dll.so trong chương trình math_demo.c ta sử dụng các flag khác nhau:
 - Sử dụng flag RTLD_LAZY sẽ khiến chương trình xảy ra lỗi Segmentation faults tại dòng 12 và bị crash do cố truy cập vào con trỏ bar giữ giá trị NULL.
 - Sử dụng flag RTLD_NOW sẽ khiến chương trình xuất ra lỗi; symbol lookup error: ./libfoo.so: undefined symbol: bar; do hàm bar chưa được định nghĩa.

vii. So sánh

	Static Libraries	Shared Libraries	Dynamically Load Libraries
Ưu điểm	<ul style="list-style-type: none"> • Các chương trình sử dụng thư viện tĩnh thông thường sẽ chạy rất nhanh bởi chúng không mất thời gian để mở thư viện ra mà dịch • Các chương trình có thể chạy độc lập mà không cần file đính kèm • Dễ thực hiện 	<ul style="list-style-type: none"> • Kích thước file thực thi nhỏ • Dễ nâng cấp chương trình, ứng dụng. Khi nâng cấp/ thay đổi, chỉ cần thay file .so cũ bằng file .so mới 	<ul style="list-style-type: none"> • Kích thước file thực thi nhỏ • Có thể đóng gói và đưa vào chương trình khác • Dễ nâng cấp chương trình, ứng dụng. Khi nâng cấp/ thay đổi, chỉ cần thay file .so cũ bằng file .so mới • Tiết kiệm bộ nhớ và cải thiện hiệu suất cũng như bộ nhớ do thư viện chỉ được tải lên khi cần.
Khuyết điểm	<ul style="list-style-type: none"> • Do khi chạy, các chương trình sẽ copy toàn bộ thư viện, kích thước file thực thi lớn • Khi thay đổi/ nâng cấp thư viện cần biên dịch lại toàn bộ chương trình 	<ul style="list-style-type: none"> • Chương trình, ứng dụng sẽ không thể chạy khi thiếu 1 file .dll nhất định • Chương trình sử dụng thư viện động thường chạy chậm hơn những chương trình sử dụng thư viện tĩnh 	<ul style="list-style-type: none"> • Khó thực hiện • Chương trình chạy chậm hơn do các tệp được tải lên tại thời điểm xử lý
Ngữ cảnh sử dụng	<ul style="list-style-type: none"> • Khi code của thư viện tồn tại trong 	<ul style="list-style-type: none"> • Khi cần cập nhật các thư viện mà vẫn cho 	<ul style="list-style-type: none"> • Khi triển khai các plugin hoặc modun

cùng một hệ thống với chương trình sử dụng nó.

- Trong các thiết bị nhúng chuyên dụng sẽ không được cập nhật chương trình.
- Trong các trường hợp trên, với cùng dung lượng lưu trữ, hiệu suất của hệ thống sẽ nhanh hơn so với sử dụng thư viện động.

phép hỗ trợ các chương trình cũ muốn sử dụng các phiên bản cũ hơn, không tương thích ngược với thư viện cũ.

- Khi cần cập nhật tính năng mà không phải biên dịch lại toàn bộ hệ thống.
- Thường được sử dụng trong các trường hợp chương trình cần truy cập các thư viện hoặc modul bên ngoài không có sẵn tại thời điểm biên dịch.
- Được sử dụng khi thư viện được dùng bởi nhiều chương trình khác nhau.

do DLL cho phép chèn tải đến khi cần.

- Khi triển khai các trình thông dịch muốn thỉnh thoảng biên dịch code thành mã máy và sử dụng phiên bản đã biên dịch mà không phải dừng chương trình.
- Thường được sử dụng khi chương trình cần tải các thư viện hoặc modul theo yêu cầu, chẳng hạn như trong kiến trúc plug-in thêm hoặc khi xử lý các chương trình lớn sử dụng nhiều thư viện.
- Được sử dụng khi thư viện chỉ được dùng bởi một vài phần của chương trình hoặc dưới một số tình huống nhất định.