HỆ ĐIỀU HÀNH

TUẦN 4

TẠO CÁC THƯ VIỆN LIÊN KẾT TĨNH, ĐỘNG

Hướng dẫn làm bài

1. Chương trình trên Linux

- Để có thể viết chương trình trên Linux, chúng ta cần phải nắm rõ 1 số vị trí tài nguyên để xây dựng chương trình như trình biên dịch, tập tin thư viện, các tập tin tiêu đề (header), các tập tin chương trình sau khi biên dịch, …
- Trình biên dịch **gcc** thường được đặt trong thư mục **/usr/bin** hoặc **/usr/local/bin** (kiểm tra bằng lệnh **which gcc**). Tuy nhiên, khi biên dịch, **gcc** cần đến rất nhiều tập tin hỗ trợ nằm trong những thư mục khác nhau như những tập tin tiêu đề (header) của C thường nằm trong thư mục **/usr/include** hay **/usr/local/include**. Các tập tin thư viện liên kết thường được **gcc** tìm trong thư mục **/lib** hoặc **/usr/local/lib**. Các thư viện chuẩn của **gcc** thường đặt trong thư mục **/usr/lib/gcc-lib**.

Chương trình sau khi biên dịch ra tập tin thực thi (dạng nhị phân) có thể đặt bất cứ vị trí nào trong hệ thống.

2. Các tập tin tiêu đề (header)

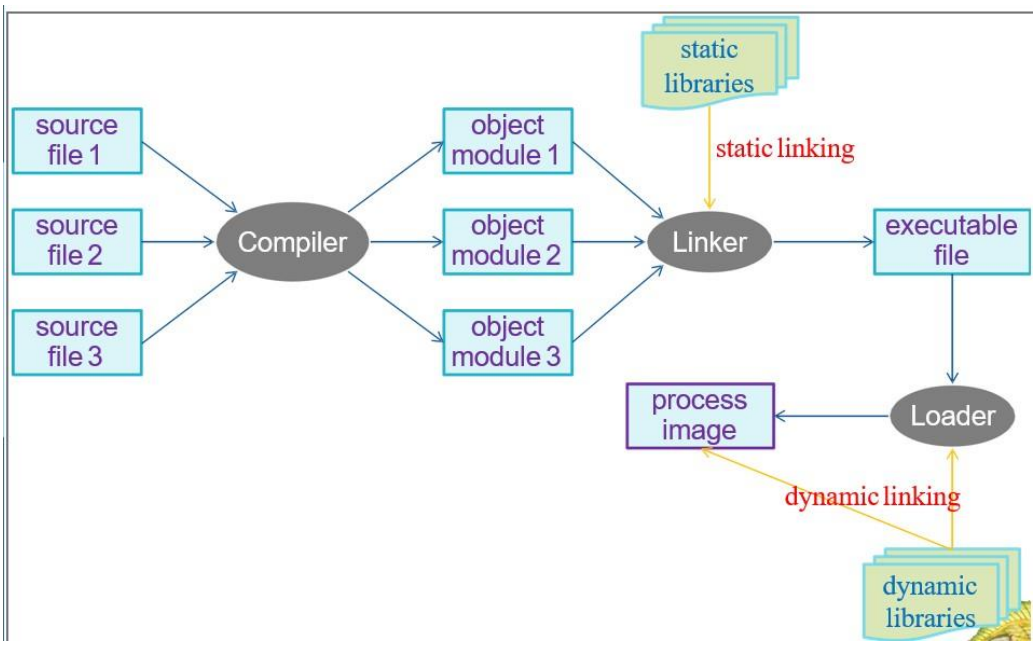
- Các tập tin tiêu đề trong C thường định nghĩa hàm và khai báo cần thiết cho quá trình biên dịch. Hầu hết các chương trình trên Linux khi biên dịch sử dụng các tập tin tiêu đề trong thư mục **/usr/include** hoặc các thư mục con bên trong thư mục này, ví dụ: **/usr/include/sys**. Một số khác được trình biên dịch dò tìm mặc định như **/usr/include/X11** đối với các khai báo hàm lập trình đồ họa X-Window, hoặc **/usr/include/g++-2** đối với trình biên dịch GNU **g++**.

Tuy nhiên, nếu chúng ta có các tập tin tiêu đề của riêng mình trong một thư mục khác thư mục mặc định của hệ thống thì chúng ta có thể chỉ rõ tường minh đường dẫn đến thư mục khi biên dịch bằng tùy chọn **–I**, ví dụ:

```
$ gcc –I/usr/mypro/include test.c –o test
```

- Khi chúng ta sử dụng một hàm nào đó của thư viện hệ thống trong chương trình C, ngoài việc phải biết khai báo nguyên mẫu của hàm, chúng ta cần phải biết hàm này được định nghĩa trong tập tin tiêu đề nào. Trình **man** sẽ cung cấp cho chúng ta các thông tin này rất chi tiết. Ví dụ, khi dùng **man** để tham khảo thông tin về hàm **kill()**, chúng ta sẽ thấy rằng cần phải

khai báo 2 tập tin tiêu đề là **types.h** và **signal.h**.



3. Các tập tin thư viện

- Các tập tin tiêu đề của C chỉ cần thiết để trình biên dịch bắt lỗi cú pháp, kiểm tra kiểu dữ liệu của chương trình và tạo ra các tập tin đối tượng. Muốn tạo ra chương trình thực thi, chúng ta cần phải có các tập tin thư viện. Trong Linux, các tập tin thư viện tĩnh của C có phần mở rộng là **.a**, **.so**, **.sa** và bắt đầu bằng tiếp đầu ngữ **lib**. Ví dụ **libutil.a** hay **libc.so** là tên các thư viện liên kết trong Linux.
- Linux có hai loại liên kết là liên kết tĩnh (**static**) và liên kết động (**dynamic**). Thư viện liên kết động trên Linux thường có phần mở rộng là **.so**, chúng ta có thể dùng lệnh **ls /usr/lib** hoặc **ls /lib** để xem các thư viện hệ thống đang sử dụng. Khi biên dịch, thông thường trình liên kết (**ld**) sẽ tìm thư viện trong 2 thư viện chuẩn **/usr/lib** và **/lib**.
- Để chỉ định tường minh một thư viện nào đó, chúng ta làm như sau:

```
$ gcc test.c –o test /usr/lib/libm.a
```

Bởi vì thư viện bắt buộc phải có tiếp đầu ngữ **lib** và có phần mở rộng là **.a** hoặc **.so**, trình biên dịch cho phép chúng ta sử dụng tùy chọn **`-l`** ngắn gọn như sau:

```
$ gcc test.c –o test -lm
```

Chúng ta sẽ thấy rằng **gcc** sẽ mở rộng **–l** thành tiếp đầu ngữ **lib** và tìm **libm.a** hoặc **libm.so** trong thư mục chuẩn để liên kết.

- Mặc dù vậy, không phải lúc nào thư viện của chúng ta cũng phải nằm trong thư viện của Linux. Nếu thư viện của chúng ta nằm ở một thư mục khác, chúng ta có thể chỉ định **gcc** tìm kiếm trực tiếp với tùy chọn **–L** như sau:

```
$ gcc test.c -otest -L/usr/myproj/lib -ltool
```

Lệnh trên cho phép liên kết với thư viện **libtool.a** hoặc **libtool.so** trong thư mục **/usr/myproj/lib**.

a) Thư viện liên kết tĩnh

Thư viện liên kết tĩnh là các thư viện khi liên kết trình biên dịch sẽ lấy toàn bộ mã thực thi của hàm trong thư viện đưa vào chương trình chính. Chương trình sử dụng thư viện liên kết tĩnh chạy độc lập với thư viện sau khi biên dịch xong. Nhưng khi nâng cấp và sửa đổi, muốn tận dụng những chức năng mới của thư viện thì chúng ta phải biên dịch lại chương trình.

- Là tập hợp các file object tạo thành một file đơn nhất
- Tương tự file .LIB trên windows
- Khi chỉ định một liên kết ứng dụng với một static library thì linker sẽ tìm trong thư viện đó để trích những file object mà bạn cần. Sau đó, linker sẽ tiến hành liên kết các file object đó vào chương trình của bạn.

Xây dựng thư viện liên kết tĩnh

Giả sử ta có 2 file hello1.c và hello2.c như sau:

hello1.c

```c
#include <stdio.h>
void hello1(int i)
{
        printf("Hello parameter 1 = %d \n",i);
}
```

hello2.c

```c
#include <stdio.h>
void hello2(int i)
{
        printf("Hello parameter 2 = %d \n",i);
}
```

- Biên dịch và tạo file object **.o**

```
$ gcc -c hello1.c hello2.c
```

- Để một chương trình nào đó gọi được các hàm trong thư viện trên, chúng ta cần tạo một tập tin header **.h** khai báo các nguyên mẫu hàm để người sử dụng triệu gọi:

```
/* lib.h */
void hello1(int i);
void hello2(int i);
```

- Cuối cùng tạo ra chương trình chính main.c triệu gọi 2 hàm này.

```
#include “lib.h”
...
```

Ví dụ hàm main:

```
#include <stdio.h>
#include <stdlib.h>
#include "lib.h"

int main(int argc, char *argv[])
{
        /* code */
        int i = atoi(argv[1]);
        int j = atoi(argv[2]);
        hello1(i);
        hello2(j);
        return 0;
}
```

- Chúng ta biên dịch và liên kết với chương trình chính như sau:

$ gcc –c main.c

$ gcc main.o hello1.o hello2.o -o main

- Sau đó thực thi chương trình

$ ./main

Ở đây **.o** là các tập tin thư viện đối tượng. Các tập tin thư viện **.a** là chứa một tập hợp các tập tin **.o**. Tập tin thư viện **.a** thực ra là 1 dạng tập tin nén được tạo ra bởi chương trình **ar**. Chúng ta hãy yêu cầu **ar** đóng **hello1.o** và **hello2.o** vào **libh.a**

Tạo thư viện *liên kết tĩnh* **libh.a** từ 2 file **hello1.c** và **hello2.c**

- Dùng lệnh **ar** để tạo thư viện tĩnh tên **libh.a**

```
$ ar cr libh.a hello1.o hello2.o
```

- Dùng lệnh **nm** để xem kết quả

```
$ nm libh.a
```

- Dùng lệnh **file** để xem **libh.a** là file gì

```
$ file libh.a
```

```
vunguyen@vunguyen:~/lab3/f1$ gcc -c hello1.c hello2.c
vunguyen@vunguyen:~/lab3/f1$ ar cr libh.a hello1.o hello2.o
vunguyen@vunguyen:~/lab3/f1$ nm libh.a

hello1.o:
0000000000000000 T hello1
                 U printf

hello2.o:
0000000000000000 T hello2
                 U printf
vunguyen@vunguyen:~/lab3/f1$ file libh.a
libh.a: current ar archive
```

Tạo **main.c** có sử dụng *hàm* trong thư viện **libh.a**

main.c

```
#include <stdio.h>
int main(int argc, char ** argv)
{
        int i = atoi(argv[1]);
        int k = atoi(argv[2]);
        hello1(i);
        hello2(k);
}
```

- Biên dịch *không link* đến thư viện liên kết tĩnh

```
$ gcc -c main.c
$ gcc -o main.o
```

```
vunguyen@vunguyen:~/lab3/f1$ gcc -o main.out main.o
main.o: In function `main':
main.c:(.text+0x50): undefined reference to `hello1'
main.c:(.text+0x5f): undefined reference to `hello2'
collect2: error: ld returned 1 exit status
```

→ báo lỗi

- Biên dịch *có link* đến thư viện liên kết tĩnh

```
vunguyen@vunguyen:~/lab3/f1$ gcc -o main.out main.o libh.a
vunguyen@vunguyen:~/lab3/f1$ ./main.out
Segmentation fault (core dumped)
vunguyen@vunguyen:~/lab3/f1$ ./main.out 5 6
Hello parameter 1 = 5
Hello parameter 2 = 6
```

→ đúng

b) Thư viện liên kết động

Khuyết điểm của thư viện liên kết tĩnh là nhúng mã nhị phân kèm theo chương trình khi biên dịch, do đó tốn không gian đĩa và khó nâng cấp. Thư viện liên kết động được dùng để giải quyết vấn đề này. Các hàm trong thư viện liên kết động không trực tiếp đưa vào chương trình lúc biên dịch và liên kết, trình liên kết chỉ lưu *thông tin tham chiếu* đến các hàm trong thư viện liên kết động. Vào lúc chương trình nhị phân thực thi, Hệ Điều Hành sẽ nạp các chương trình liên kết cần tham chiếu vào bộ nhớ. Như vậy, nhiều chương trình có thể sử dụng chung các hàm trong một thư viện duy nhất.

- Tương tự thư viện dạng .DLL của windows Thư mục chứa thư viện chuẩn **/usr/lib, /lib**

Tạo thư viện liên kết động

- Tạo thư viện liên kết động **libd.a** từ 2 file **hello1.c** và **hello2.c**
- Khi biên dịch tập tin đối tượng để đưa vào thư viện liên kết động, chúng ta phải thêm tùy chọn – **fPIC** (PIC- Position Independence Code – mã lệnh vị trí độc lập).

```
$ gcc -c -fPIC hello1.c hello2.c
```

- Để tạo ra thư viện liên kết động, chúng ta không sử dụng trình **ar** như với thư viện liên kết tĩnh mà dùng lại **gcc** với tùy chọn **-shared**.

```
$ gcc -shared -o libd.a hello1.o hello2.o
```

```
vunguyen@vunguyen:~/lab3/f1$ gcc -c -fPIC hello1.c hello2.c
vunguyen@vunguyen:~/lab3/f1$ gcc -shared -fPIC -o libd.a hello1.o hello2.o
```

- Để sử dụng được thư viện liên kết động, trước hết chúng ta phải đưa thư viện vào thư mục **/lib** sử dụng lệnh **copy**

```
$ sudo cp libd.a /lib
$ gcc -c main.c
$ gcc -o main.out2 main.o libd.a
```

- Kiểm tra xem Hệ Điều Hành có thể tìm ra tất cả các thư viện liên kết động mà chương trình sử dụng hay không:

```
$ ldd main.out2
```

- Rồi chạy chương trình sử dụng thư viện liên kết động này:

```
$ ./main.out2 2 3
```

```
vunguyen@vunguyen:~/lab3/f1$ gcc -o main.out2 main.o libd.a
vunguyen@vunguyen:~/lab3/f1$ ./main.out2 2 3
Hello parameter 1 = 2
Hello parameter 2 = 3
```

Bài tập thực hành

Bài 1: Viết chương trình tính tổng **add(int a, int b)**, hiệu **sub(int a, int b)** của 2 số a và b được *truyền tham số từ dòng lệnh* bằng cách sử dụng các thư viện từ câu a. và câu b.

a. Liên kết tĩnh từ 2 tập tin add.c và sub.c (**ar**)

b. Liên kết động từ 2 tập tin add.c và sub.c (**-fPIC** + **-shared**)

c. Sử dụng thư viện ở a. và b., viết hàm **main** truyền vào hai số nguyên và dấu phép tính "+" hay "-", và in ra kết quả tương ứng. Chương trình báo lỗi nếu các đối số truyền vào không đúng theo qui tắc.

```
>  ./main.out 1 + 2
>  Ket qua: 3
>  ./main.out 3 - 4
>  Ket qua: -1
>  ./main.out 1 * 3
>  Doi so truyen vao khong dung
```

Bài 2: Viết chương trình nhập, xuất mảng số nguyên (sử dụng thư viện *liên kết động*).

Bài 3: Tạo thư mục **`/home/dsl/lib`**

- Chép thư viện được tạo ở Bài 2 vào thư mục vừa tạo.
- Biên dịch và chạy lại chương trình. Cho nhận xét.

HẾT