

Bài 2: LẬP TRÌNH C TRÊN LINUX

I. Lý thuyết

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 -otest
```

- 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 -otest /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 -otest -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 -o test -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**.

4. Thư viện liên kết trên Linux

- Hình thức đơn giản nhất của thư viện là tập hợp các tập tin **.o** do trình biên dịch tạo ra ở bước biên dịch với tùy chọn **-c**. Ví dụ

```
$ gcc -c helloworld.c
```

trình biên dịch chưa tạo ra tập tin thực thi mà tạo ra tập tin đối tượng **helloworld.o**. Tập tin này chứa các mã máy của chương trình đã được sắp xếp lại. Nếu muốn tạo ra tập tin thực thi, chúng ta gọi trình biên dịch thực hiện bước liên kết:

```
$ gcc helloworld.o -o helloworld
```

Trình biên dịch sẽ gọi tiếp trình liên kết **ld** tạo ra định dạng tập tin thực thi cuối cùng. Ở đây, nếu chúng ta không sử dụng tùy chọn **-c**, trình biên dịch sẽ thực hiện cả hai bước đồng thời.

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.

Ví dụ sử dụng liên kết tĩnh:

```
/* cong.c */
int cong( int a, int b )
{
    return a + b;
}
```

```
/* nhan.c */
long nhan( int a, int b )
{
    return a * b;
}
```

Thực hiện biên dịch để tạo ra hai tập tin thư viện đối tượng **.o**

```
$ gcc -c cong.c nhan.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 */
int cong( int a, int b );
long nhan( int a, int b );
```

Cuối cùng, tạo ra chương trình chính **program.c** triệu gọi hai hàm này.

```
/* program.c */
```

```

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

int main ()
{
    int a, b;
    printf( "Nhap vào a : " );
    scanf( "%d", &a );
    printf("Nhap vào b : " );
    scanf( "%d", &b );
    printf( "Tổng %d + %d = %d\n", a, b, cong( a, b ) );
    printf( "Tich %d * %d = %ld\n", a, b, nhan( a, b ) );
    return ( 0 );
}

```

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

```

$ gcc -c program.c
$ gcc program.o cong.o nhan.o -oprogram

```

Sau đó thực thi chương trình

```

$ ./program

```

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 **cong.o** và **nhan.o** vào **libfoo.a**

```

$ ar cvr libfoo.a cong.o nhan.o

```

Sau khi đã có được thư viện **libfoo.a**, chúng ta liên kết lại với chương trình theo cách sau:

```

$ gcc program.o -oprogram libfoo.a

```

Chúng ta có thể sử dụng tùy chọn **-l** để chỉ định thư viện khi biên dịch thay cho cách trên. Tuy nhiên **libfoo.a** không nằm trong thư mục thư viện chuẩn, cần phải kết hợp với tùy chọn **-L** để chỉ định đường dẫn tìm kiếm thư viện trong thư mục hiện hành. Dưới đây là cách biên dịch:

```

$ gcc program.c -oprogram -L -lfoo

```

Chúng ta có thể sử dụng lệnh **nm** để xem các hàm đã biên dịch sử dụng trong tập tin chương trình, tập tin đối tượng **.o** hoặc tập tin thư viện **.a**. Ví dụ:

```

$ nm cong.o

```

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.

- Tạo thư viện liên kết động:

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).

Ví dụ: biên dịch lại 2 tập tin **cong.c** và **nhan.c**

```

$ gcc -c -fpic cong.c nhan.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 cong.o nhan.o -olibfoo.so
```

Nếu tập tin **libfoo.so** đã có sẵn trước thì không cần dùng đến tùy chọn **-o**

```
$ gcc -shared cong.o nhan.o libfoo.so
```

Bây giờ chúng ta đã có thư viện liên kết động **libfoo.so**. Biên dịch lại chương trình như sau:

```
$ gcc program.c -oprogram -L. -lfoo
```

- Sử dụng thư viện liên kết động:

Khi Hệ Điều Hành nạp chương trình **program**, nó cần tìm thư viện **libfoo.so** ở đâu đó trong hệ thống. Ngoài các thư mục chuẩn, Linux còn tìm thư viện liên kết động trong đường dẫn của biến môi trường **LD_LIBRARY_PATH**. Do **libfoo.so** đặt trong thư mục hiện hành, không nằm trong các thư mục chuẩn nên ta cần đưa thư mục hiện hành vào biến môi trường **LD_LIBRARY_PATH**:

```
$ LD_LIBRARY_PATH=.:
```

```
$ export LD_LIBRARY_PATH
```

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 program
```

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

```
$ ./program
```

- Một khuyết điểm của việc sử dụng thư viện liên kết động đó là thư viện phải tồn tại trong đường dẫn để Hệ Điều Hành tìm ra khi chương trình được triệu gọi. Nếu không tìm thấy thư viện, Hệ Điều Hành sẽ chấm dứt ngay chương trình cho dù các hàm trong thư viện chưa được sử dụng. Ta có thể chủ động nạp và gọi các hàm trong thư viện liên kết động mà không cần nhờ vào Hệ Điều Hành bằng cách gọi hàm liên kết muộn.

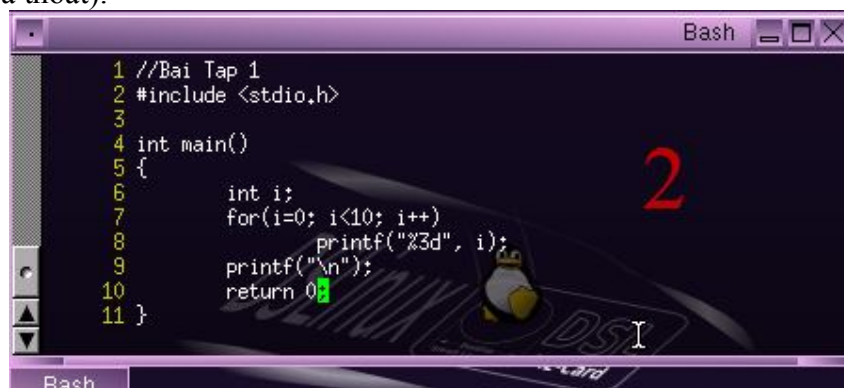
II. Nội dung bài thực hành số 2

1. Viết chương trình C in ra màn hình các số nguyên từ 0 đến 9

Bước 1: Mở chương trình soạn thảo: **\$vi thuchanh.c**

Bước 2: Viết chương trình:

- Khởi đầu vào màn hình **vi** bạn đang ở chế độ xem (view). Muốn chỉnh sửa nội dung file bạn nhấn phím **Insert**. Dòng trạng thái cuối màn hình đổi thành **--INSERT--** cho biết bạn đang trong chế độ soạn thảo (bạn cũng có thể nhấn phím **i** hoặc **a** thay cho phím **Insert**).
- Nhấn **Enter** nếu bạn muốn sang dòng mới. Nhấn các phím mũi tên để di chuyển con trỏ và thay đổi nội dung file. Muốn ghi lại nội dung file sau khi soạn thảo, bạn nhấn **Esc** để trở về chế độ lệnh và nhấn **:w**. Muốn thoát khỏi **vi** bạn nhấn **:q** (hoặc **:wq** để lưu và thoát).



```
Bash
1 //Bai Tap 1
2 #include <stdio.h>
3
4 int main()
5 {
6     int i;
7     for(i=0; i<10; i++)
8         printf("%3d", i);
9     printf("\n");
10    return 0;
11 }
```

Bước 3: Biên dịch chương trình thành tập tin đối tượng: `$gcc -c thuchanh.c`

Bước 4: Biên dịch tập tin đối tượng thành tập tin thực thi: `$gcc thuchanh.o -o thuchanh`

->Lưu ý: Có thể gom bước 3 và 4 bằng câu lệnh: `$gcc thuchanh.c -o thuchanh`

Bước 5: Thực thi chương trình bằng lệnh: `$. /thuchanh`



2. Viết chương trình cộng và nhân 2 số nguyên sử dụng thư viện liên kết tĩnh:

```
$ vi cong.c
int cong(int a, int b)
{
    return a + b;
}
```

```
$ vi nhan.c
int nhan(int a, int b)
{
    return a * b;
}
```

```
$ vi program.c
#include <stdio.h>

int main()
{
    int a, b;
    printf("\nNhap a:");
    scanf("%d",&a);
    printf("Nhap b:");
    scanf("%d",&b);
    printf("\nTong cua hai so la: %d",cong(a,b));
    printf("\nTich cua hai so la: %d\n",nhan(a,b));
    return 0;
}
```

```
$ gcc -c cong.c nhan.c
$ ar cvr libfoo.a cong.o nhan.o
$ gcc program.c -o program -L. -lfoo
$ ./program
```

3. Viết chương trình cộng và nhân 2 số nguyên sử dụng thư viện liên kết động:

```
$ vi cong.c
int cong(int a, int b)
{
    return a + b;
}
```

```
$ vi nhan.c
int nhan(int a, int b)
{
    return a * b;
}
```

```
$ vi program.c
#include <stdio.h>

int main()
{
    int a, b;
    printf("\nNhap a:");
    scanf("%d",&a);
    printf("Nhap b:");
    scanf("%d",&b);
    printf("\nTong cua hai so la: %d",cong(a,b));
    printf("\nTich cua hai so la: %d\n",nhan(a,b));
    return 0;
}
```

```
$ gcc -c -fpic cong.c nhan.c
$ gcc -shared cong.o nhan.o -o libfoo.so
$ gcc program.c -o program -L. -lfoo
$ LD_LIBRARY_PATH=.:
$ export LD_LIBRARY_PATH
$ ./program
```

4. Bài tập thêm

4.1. Viết chương trình nhập, xuất mảng số nguyên(sử dụng thư viện liên kết động).

4.2. Tạo thư mục `/home/dsl/lib`

- Chép thư viện `libfoo.so` tạo được ở câu 4.1 vào thư mục vừa tạo.
- Biên dịch và chạy lại chương trình.