CHƯƠNG 7:

LẬP TRÌNH C TRÊN LINUX

1

Nội dung

- Linux và Unix
- Cộng đồng GNU và General Public License
- Lập trình trên Linux
- Chương trình Unix và Linux
- Chương trình Linux đầu tay helloworld.c
- Tìm trợ giúp về trình biên dịch
- Phát triển chương trình với ngôn ngữ C
- Thư viện liên kết trên Linux

2

1. Linux và Unix

- Unix được đánh giá là một HĐH mạnh, ổn định. Trước đây được xem là một HĐH mã nguồn mở; từ khi Sun Microsystems nâng cấp lên thành một sản phẩm thương mại → mất dần tính mở của HĐH Unix.
- Theo dòng lịch sử thì với việc kế thừa và phát huy những tính năng nổi bật của một HĐH đã qua thử thách – UNIX + {*mã nguồn mở + tính ổn định*} ⇨ Linux đã được ủng hộ và được sử dụng bởi cộng đồng trên toàn thế giới

3

- Chương trình viết trên Unix đều có thể chạy tốt trên Linux và ngược lại
- Linux đã được phát triển dựa trên việc tận dụng những ưu điểm và khắc phục những khuyết điểm của các HĐH tựa UNIX
- Linux ngày nay được sử dụng rộng rãi và chính do yếu tố phổ biến trên đã được sự hỗ trợ giúp đỡ của khá nhiều cộng đồng người dùng trên thế giới
- Một số tính năng ***HĐH Linux*** khác so với ***HĐH Unix***:
  - Là hạt nhân cung cấp các chức năng cần thiết tối thiểu của HĐH tựa Unix (*Linux giống Unix gần 98%*)
  - Là một sản phẩm có giá trị do Unix không có phiên bản chạy trên hệ máy PC với kiến trúc Intel

4

2. Cộng đồng GNU và General Public License

- Ngoài HĐH, còn có các chương trình ứng dụng phục vụ cho yêu cầu của người dùng
- Cộng đồng Open Source đã xây dựng nhiều ứng dụng có khả năng chạy được trên Unix/Linux và nhìn chung theo xu hướng hiện đại nhằm lôi kéo người dùng Linux theo phương châm

  "Windows có gì ~ Linux có tương ứng như vậy"

- **GNU** – {*GNU's Not UNIX*} – GNU theo nguyên gốc tiếng Anh là "linh dương đầu bò" <=> là biểu tượng của tổ chức cộng đồng mã nguồn mở
- Tham khảo các phần mềm ứng dụng miễn phí tại địa chỉ http://www.gnu.org/software/software.html
- Khi sử dụng các phần mềm của tổ chức GNU thì cần phải tuân thủ một số quy định của tổ chức trên → Giấy phép chứng nhận GPL còn gọi là *copyleft* thay cho *copyright* cho các chứng nhận bản quyền thương mại

Một số bộ công cụ biên dịch C/C++

gcc	Trình biên dịch C
g++	Trình biên dịch C++
gdb	Trình gỡ lỗi
GNU make	Trình quản lý mã nguồn và quản lý thư viện
GNU Emacs	Trình soạn thảo văn bản

```
$ sudo apt-get update
$ sudo apt-get install gcc
```


3. Lập trình trên Linux

- Nguyên thủy Unix được viết bằng C và phần lớn các ứng dụng trên Unix được viết bằng C
- Ngoài C, có thể sử dụng Pascal, Java hoặc Perl để xây dựng các chương trình cho Linux

Ada	**C**	**C++**
Eiffel	Forth	Fortran
Icon	**Java**	**JavaScript**
Lisp	Modula 2	Modula 3
Oberon	**Objective C**	**Pascal**
Perl	PostScript	Prolog
Python	Scheme	Smalltalk
SQL	Tcl / Tk	UNIX Bourne Shell

4. Chương trình Unix và Linux

- Chương trình trên Unix và Linux tồn tại ở hai dạng:
  - Dạng thực thi (tập tin nhị phân): File chương trình thực thi ~ như tập tin *.exe* của MS-DOS/Windows
  - Dạng thông dịch (tập tin script): File script là những chỉ thị lệnh được thực thi bởi shell hay trình thông dịch nào đó (Perl, Python, Tcl, ...)

Các file script và chương trình nhị phân đều có khả năng và mạnh ngang nhau => lúc thực thi khó phân biệt được đâu là lệnh gọi chương trình nhị phân và đâu là lệnh thực hiện script

Khi chương trình được gọi, Linux sẽ tìm đường dẫn đến nơi chứa tập tin chương trình trong biến môi trường **PATH**. Thông thường, biến này chứa các đường dẫn sau

/bin, **/user/bin**, **/usr/local/bin**

5. Chương trình Linux đầu tiên (helloworld.c)

- Dùng trình soạn thảo bất kỳ (trong Console Mode) để soạn thảo source code như:
  - vi
  - pico
  - cat
- Cách sử dụng:
  - vi helloworld.c
  - cat > helloworld.c

```
#include <stdio.h>
int main () {
   printf("Hello World!\n");
}
```


Thực hiện

- Sử dụng ***gcc*** (hoặc ***cc***) để biên dịch như sau:
-

```
$ gcc –c helloworld.c
$ gcc helloworld.o -o helloworld
$ ./helloworld
Hello World
$
```

Cách khác:

$ gcc filenguon.c

$ ./a.out

6. Tìm trợ giúp về trình biên dịch

- Trong trình ***gcc*** có rất ít tùy chọn (`-o, -c, -g, -l, -L, -I...`).
- Sử dụng lệnh ***man*** để tham khảo thêm một số tuỳ chọn khác.

7. Phát triển chương trình với ngôn ngữ C

- Sử dụng ngôn ngữ C làm ngôn ngữ chính trong chương trình
- Ngôn ngữ lập trình C không phụ thuộc vào HĐH nào

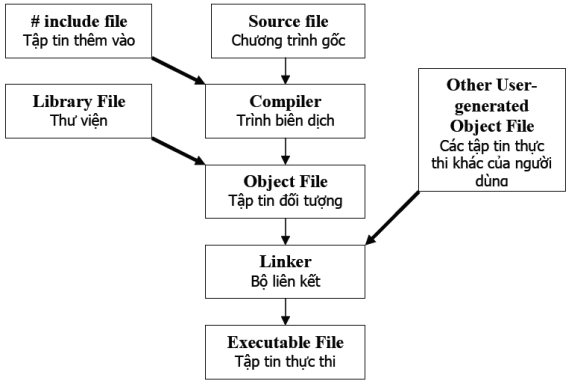
7.1. Chương trình trên Linux

- Cần nắm rõ vị trí đặt tài nguyên để xây dựng chương trình như trình biên dịch, file thư viện, các file header khai báo hàm và cấu trúc dữ liệu...

- sudo apt-get update
- sudo apt-get install gcc

- Trình biên dịch ***gcc*** thường được đặt trong **/usr/bin** hoặc **usr/local/bin**.
- Khi biên dịch, ***gcc*** cần nhiều file hỗ trợ như:
  - các tập tin thư viện liên kết (trong **/lib** hoặc **/usr/local/lib**),
  - các file C header (trong **/usr/include** hoặc **/usr/local/include**)…
- Chương trình sau khi biên dịch có thể đặt bất kỳ đâu trên hệ thống miễn là HĐH có thể tìm thấy trong biến môi trường **PATH** hoặc theo đường dẫn tuyệt đối trong dòng lệnh.

Biên dịch và thi hành chương trình



7.2. Các file header

- File header trong C chứa định nghĩa các hàm, các hằng cùng với những cấu trúc dữ liệu cần cho quá trình biên dịch
- Ví dụ một khai báo header trong C:

```
#include <sys/types.h>
```

- Trình biên dịch sẽ tìm file header có tên là *types.h* trong thư mục **/usr/include/sys**

- Sử dụng tùy chọn ***-I*** để chỉ ra thư mục chứa các file header của riêng mình (khác với thư mục mặc định của hệ thống).
- Ví dụ:

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


7.3. Các file thư viện

- Để tạo ra chương trình thực thi, cần có các file thư viện
- Trong Linux, các file thư viện thường là "*.a*", "*.so*", "*.sa*" và được bắt đầu bằng tiếp đầu ngữ "***lib***".

  Ví dụ: ***libutil.a***, ***libc.so*** *(Đây là tên các thư viện trong Linux)*

- Có 2 loại thư viện liên kết:
  - **Liên kết tĩnh (Static)**
  - **Liên kết động (Dynamic)**
- Khi biên dịch, thông thường chương trình liên kết (***ld***) sẽ tìm thư viện trong 2 thư mục chuẩn **/usr/lib** và **/lib**

8. Thư viện liên kết trên Linux

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

$ `gcc -c helloworld.c`

- Lúc này trình biên dịch chưa tạo ra file thực thi mà tạo ra file đối tượng helloworld.o (file này chứa mã máy của chương trình đã được tổ chức lại. Các file "*.o*" này tương tự như các file "*.obj*" do trình biên dịch C sinh ra trên DOS hoặc Windows)

- Để tạo ra file thực thi, thực hiện bước cuối cùng (gọi linker ***ld***)

$ `gcc helloworld.o -o helloworld`

- Nếu không dùng khoá chuyển "*-c*" thì trình biên dịch sẽ thực hiện cả hai bước trên đồng thời.

`$ gcc helloworld.c`

Sẽ tạo ra file `a.out` và có thể thực thi được file này luôn

8.1. 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 này chạy độc lập với thư viện sau khi biên dịch.

➢ Khi cần sửa chữa hoặc nâng cấp -> **Biên dịch lại**

Ta có 2 file chương trình như sau

- ```
  /* 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 **lib.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 );
```


Chương trình chính triệu gọi 2 hàm trên

```
/* program.c */
#include <stdio.h>
#include "lib.h"
int main (){
        int a, b;
        printf( "Nhap vao a : " );
        scanf( "%d", &a );
        printf("Nhap vào b : " );
        scanf( "%d", &b );
        printf( "Tong %d + %d = %d\n", a, b, cong( a, b ) );
        printf( "Tich %d * %d = %ld\n", a, b, nhan( a, b ) );
        return 0;
}
```


Biên dịch và liên kết với chương trình chính

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

- Sau đó thực thi chương trình

```
$ ./program
```


Các file .o là các tập tin thư viện đối tượng.

Có thể nén lại thành file .a như sau:

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

Thư viện libfoo.a được liên kết lại với chương trình:

```
$ gcc program.o libfoo.a -oprogram
```

-> Hị, chưa chạy được đâu

- Có thể sử dụng tùy chọn -l để chỉ định thư viện khi biên dịch.
- 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 **-o**program **-L**. **-l**foo
  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

8.2. Thư viện liên kết động

8.2.1. Tạo và sử dụng thư viện liên kết động .so

- Thư viện liên kết tĩnh có nhược điểm:
  - Phải "*nhúng*" mã nhị phân kèm theo chương trình khi liên kết, dẫn đến tình trạng tốn không gian đĩa nếu như có chương trình yêu cầu sử dụng một hàm nhiều lần (ví dụ như hàm ***printf()***)
  - Phải biên dịch và/hoặc liên kết lại khi muốn nâng cấp (hoặc bổ sung tính năng mới)

8.2. Thư viện liên kết động

- Thư viện liên kết động khắc phục được hai vấn đề trên do các hàm thư viện liên kết động được nạp trong thời gian thi hành và có thể được dùng chung giữa nhiều tiến trình.
- Để đưa hàm vào thư viện liên kết động
  - Cần dùng tùy chọn ***-fpic*** (hoặc **–fPIC),**

PIC: viết tắt của **P**osition **I**ndependence **C**ode

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

```
$ gcc -c -fPIC cong.c nhan.c
```

- Dùng gcc với tùy chọn `-shared`

```
$ gcc -shared cong.o nhan.o -o libfoo.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 -o program libfoo.so
$ gcc program.c -o program -L. -lfoo
```

- Tham số -L chỉ ra vị trí thư mục để tìm thư viện có từ foo trong tên được chỉ ra ở tham số -l

- File thư viện có hậu tố:
- .a
- .so
- Và tiền tố là lib
- libnm.a / .so

- Có thể tìm tên thư viện
  -lnm
  -L -> chỉ ra thư mục cần tìm thêm

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

Tham số với chương trình main

```
#include <stdio.h>
int main(int argc, char *argv[]) {
   if (argc != 3) {
      printf("Usage:\n %s Integer1 Integer2\n", argv[0]);
   }
   else {
     printf("%s + %s = %d\n",
            argv[1], argv[2], atoi(argv[1]) + atoi(argv[2]));
   }
   return 0;
}
// lưu ý: argv[0] là tên chương trình, tương tự $0 trong shell
```


Sử dụng cách làm thư viện liên kết tĩnh, liên kết động & tham số truyền vào main

- 2 kích thước của ma trận được nhập vào từ bàn phím (dạng tham số dòng lệnh)
- matran.c -> matran.o -> matran
- matran 4 5
- Nhập vào một ma trận -> 1 file .c
- Xuất ra ma trận đó -> 1 file .c
- Sắp xếp các phần tử của ma trận theo thứ tự tăng dần
- Xuất ra ma trận sau khi đã sắp xếp

8.2.2. Thực thi gọi liên kết muộn

- Thư viện liên kết động phải nằm trong /lib, /usr/lib, /usr/local/lib hoặc thư mục với đường dẫn chỉ bởi biến môi trường **LD_LIBRARY_PATH**
- Ví dụ:

```
#export
LD_LIBRARY_PATH=/myprog/lib:/other/lib:.
#./program
```

- Có thể chủ động gọi và nạp các hàm thư viện liên kết động mà không cần nhờ vào HĐH. Đây được gọi là *"hàm liên kết muộn"*
- Ba hàm liên kết muộn chính là:
  - **dlopen()**
  - **dlsym()**
  - **dlclose()**

- Các hàm trên được khai báo như sau:

```
#include <dlfcn.h>
void *dlopen (const char* lib_file, int mode);
void *dlsym (void * handle, const char* funct_name);
int dlclose (void *handle);
```

- Hàm ***dlopen()*** yêu cầu đường dẫn ***lib_file*** tới thư viện ".so" cần nạp.
- Hàm ***dlsym()*** yêu cầu con trỏ đến thư viện nạp và trả về trước đó bởi hàm ***dlopen()***
- Hàm ***dlclose()*** giải phóng thư viện trỏ đến bởi *handle* do ***dlopen()*** nạp trước đó

Gọi chương trình shell từ C

Nếu muốn gọi chương trình shell từ trong chương trình C, ta có thể thực hiện như sau:

```
#include <stdlib.h>
int system(const char *command);
```

Ví dụ gọi pwd

```
system("pwd");
```