CHUYÊN ĐỀ HỆ ĐIỀU HÀNH LINUX

Tuần 9: Tạo makefile đơn giản

GVLT: NGUYỄN Thị Minh Tuyền

Nội dung

1. Module trong C
2. Biên dịch thủ công
3. Cây phụ thuộc
4. Biên dịch tự động

Nội dung

1. **Module trong C**
2. **Biên dịch thủ công**
3. **Cây phụ thuộc**
4. **Biên dịch tự động**

Thư viện và module

- Thư viện
  - tập hợp của phần mềm (đã biên dịch trước hoặc trực tiếp mã nguồn)
- Module
  - tổ chức phần mềm mà mỗi module đã đóng gói một số chức năng bên trong một thư viện

`#include <math.h>` → thực hiện các thao tác toán học

Phân chia thành các module C: để làm gì ?

- phân chia dự án thành những phần nhỏ, rõ ràng hơn ;
- phát triển độc lập từng phần ;
- tái sử dụng dễ dàng ở chương trình khác ;
- không cần biên dịch lại những phần không thay đổi.

Tạo một module C

- Một module = 1 file .c [+ 1 file .h]
- File header .h : là giao diện, chứa :
  - các khai báo hàm, các hằng số và macro
  - các định nghĩa kiểu (struct, enum)

  → tất cả những gì ta muốn truy cập vào module, đặt vào file header
- File .c : là file cài đặt, chứa: (chứa các định nghĩa hàm hoặc chi tiết cài đặt)
  - biến toàn cục trong module
  - định nghĩa hàm hoặc chi tiết cài đặt của hàm

Public vs private

- Hàm hay dữ liệu public :
  - được sử dụng ở các module khác ;
  - được tài liệu hoá, có phiên bản ;
  - là một phần của API.
- Hàm hay dữ liệu private:
  - chỉ sử dụng cho bản thân module đó ;
  - có thể thay đổi mà không cần đưa cảnh báo;
  - việc sử dụng bên ngoài có thể gây nguy hiểm.

Module C và public/private

- File .h chứa các hàm API public :
  - các hằng số và macro public
  - các khai báo kiểu public
  - các biến ngoại public
  - các khai báo hàm public
- File .c cũng chứa cài đặt public, và các dữ liệu private :
  - các hằng số, macro và kiểu private
  - các biến toàn cục của module
  - thân hàm public và private

Ví dụ [1]

File max.h :

```
#ifndef MAX_H
#define MAX_H

    int max (int a, int b);

#endif // MAX_H
```

File max.c :

```
#include "max.h"

int max (int a, int b)
{
    if (a > b) return a;
    return b;
}
```

`#ifndef .. #define .. #endif` : là một chỉ thị (directive), gọi là include guard

Mục tiêu : bảo vệ chương trình khỏi việc tạo vòng lặp khi include hoặc include dư thừa một khai báo.

Ví dụ [2]

File point.h :

```
#ifndef POINT_H
#define POINT_H
typedef struct {
    int x, y;
} Point ;

void display_point (Point p);
#endif // POINT_H
```

File point.c :

```
#include <stdio.h>
#include "point.h"
void display_point (Point p) {
  printf ("(%d,%d)\n",
            p.x, p.y);
}
```

Sự khác nhau giữa #include <..> et #include "..." ?

Ví dụ [3]

File `principal.c` :

```
#include <stdio.h>
#include "max.h"
#include "point.h"
int main () {
    Point p = { 3, 4 };
   display_point (p);
    printf ("max = %d\n", max (p.x, p.y));
    return 0;
}
```

Nội dung

1. Module trong C
2. **Biên dịch thủ công**
3. Cây phụ thuộc
4. Biên dịch có điều kiện

Biên dịch thủ công

- Mỗi file `.c` sẽ tạo ra một file `.o`
- Để sinh ra các file này: sử dụng option `-c`

```
gcc -Wall -c max.c
gcc -Wall -c point.c
gcc -Wall -c principal.c
```

- Theo bất cứ thứ tự nào.

Xem nội dung một file .o : lệnh nm


```
$ nm max.o
0000000000000000 T _max

$ nm point.o
0000000000000000 T _display_point
                 U _printf


$ nm principal.o
                 U _display_point
0000000000000000   T _main
                 U _max
                 U _printf
```

Sinh ra file thực thi

- Với `gcc` :

```
gcc max.o point.o principal.o -o exo1
```

- Tên file thực thi: mặc định `a.out`
- Bên trong, `gcc` gọi lệnh `ld` (được gọi là trình liên kết)
- `ld` thông báo các lỗi:
  - tham chiếu không xác định
  - tham chiếu nhiều lần

Các ký hiệu trong file thực thi

```
$ nm exo1
0000000100000000 T __mh_execute_header
0000000100000ef0 T _display_point
0000000100000f20 T _main
0000000100000ec0 T _max
                 U _printf
                 U dyld_stub_binder
```

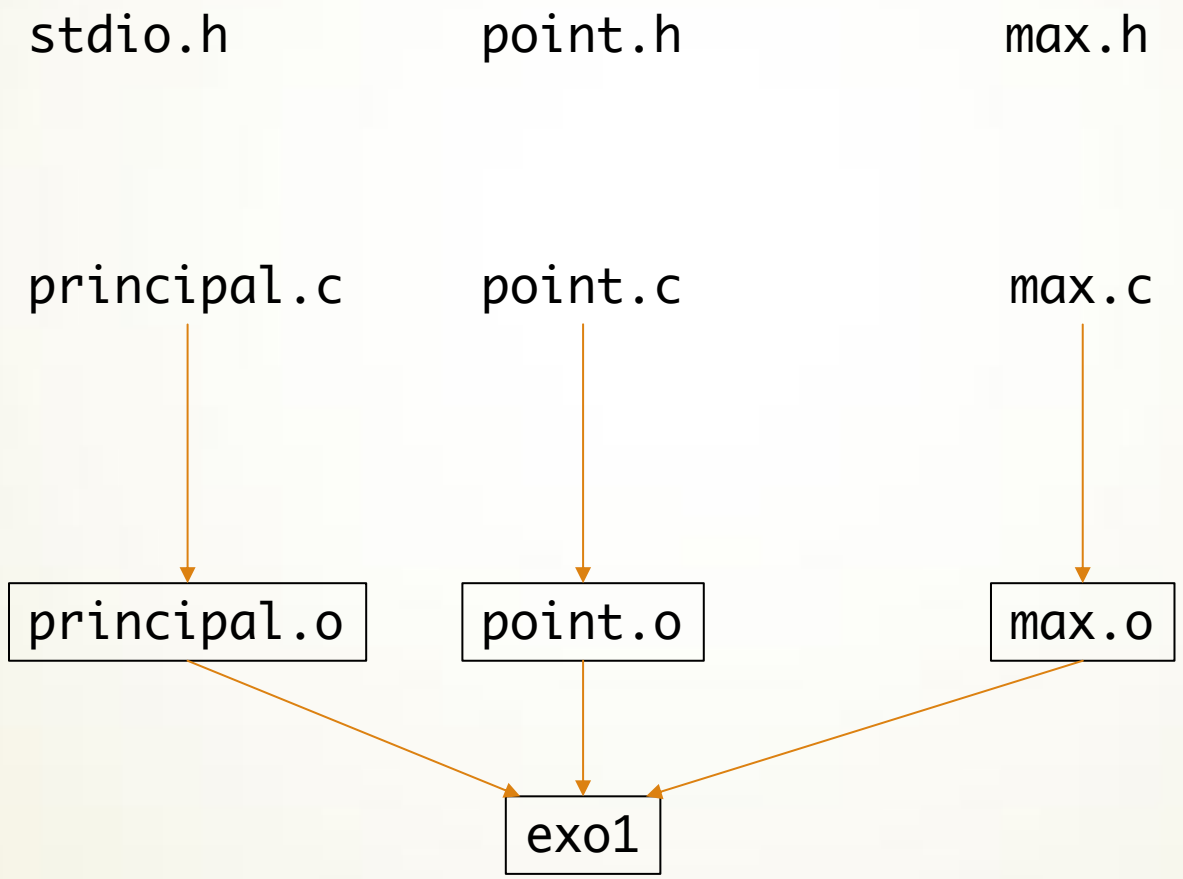
Thư viện đã được biên dịch sẵn

- Thư viện = file chứa một hoặc nhiều `.o`
- Thư viện tĩnh: Liên kết trực tiếp vào file thực thi đầu ra
  - được cài đặt với chương trình như là một phần của file thực thi
- Thư viện chia sẻ: liên kết động tại thời điểm thực thi
  - được cài đặt trước
  - được sử dụng cho các ứng dụng trên hệ điều hành
  - file thực thi được đặt tách biệt với thư viện

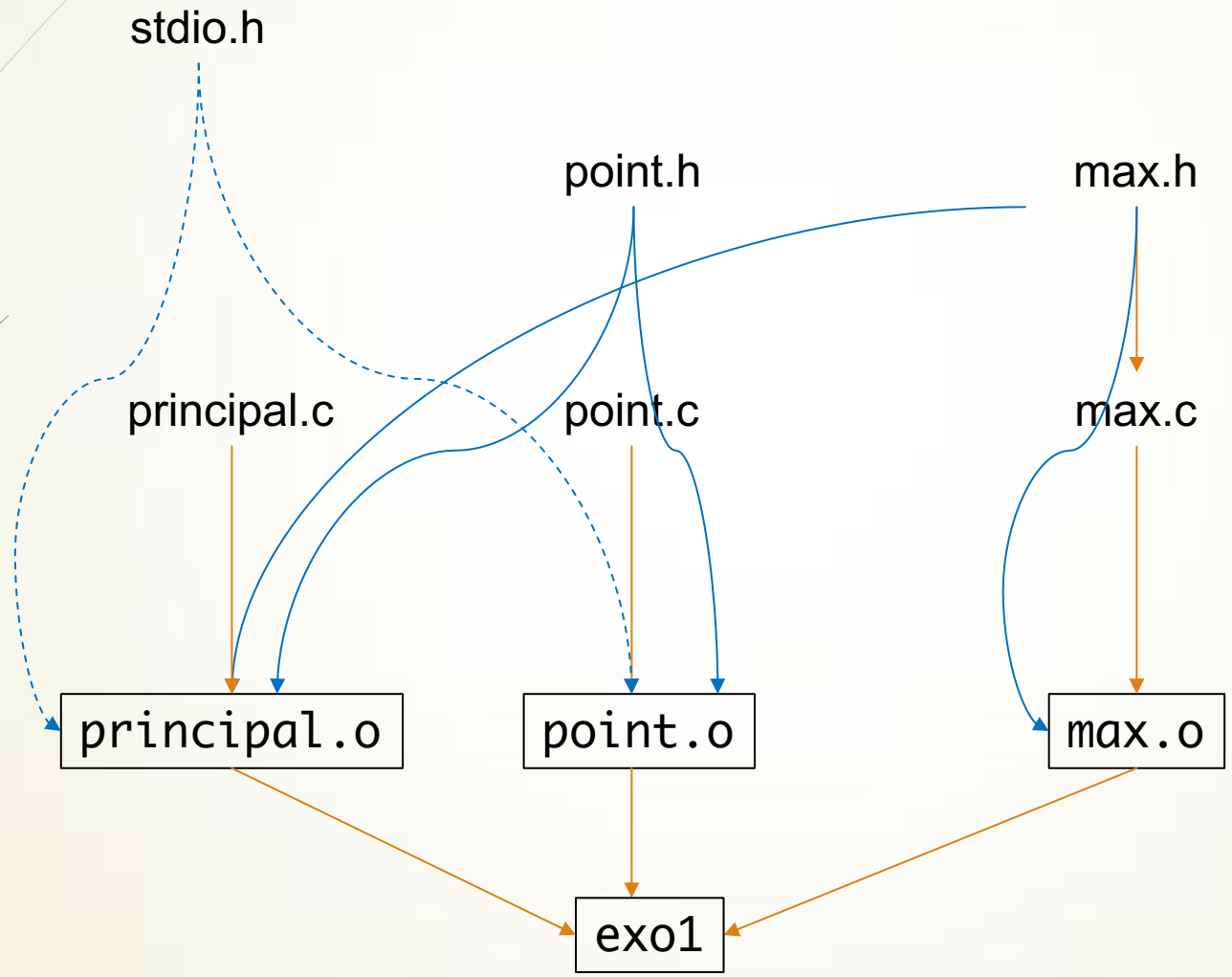
Nội dung

1. Module trong C
2. Biên dịch thủ công
3. **Cây phụ thuộc**
4. Biên dịch có điều kiện

Cây phụ thuộc

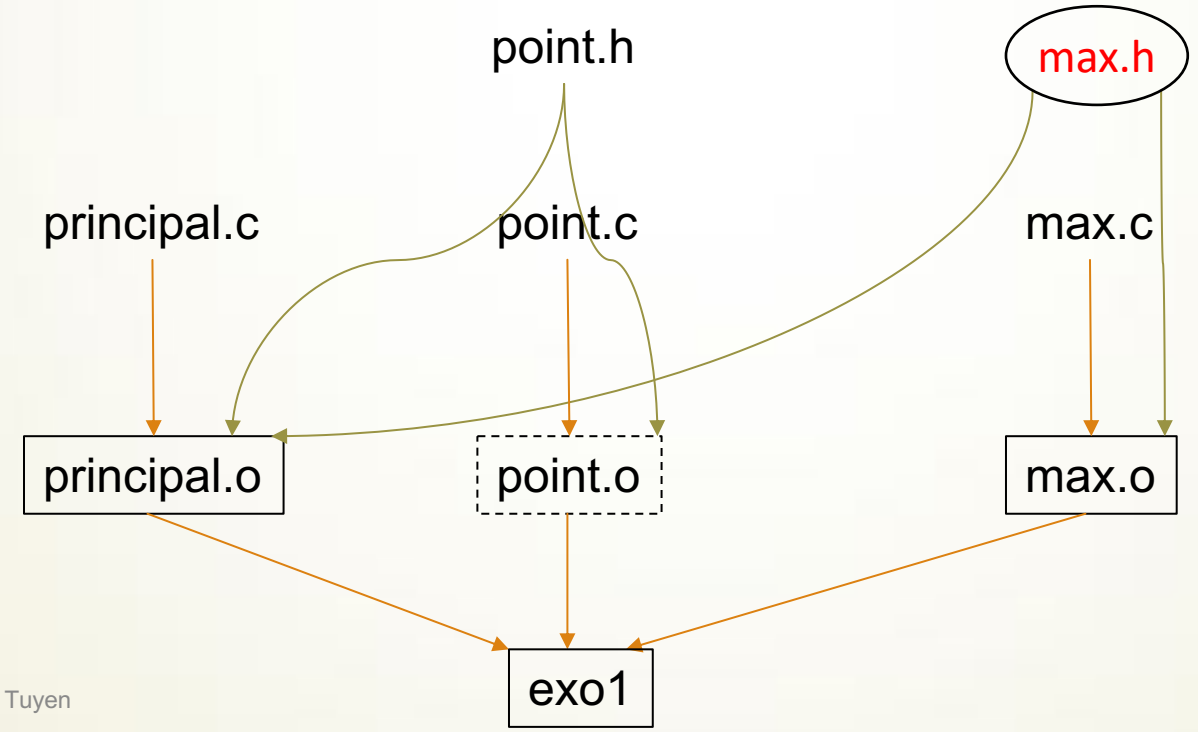


Các phụ thuộc với file .h



Sử dụng cây phụ thuộc

- Nếu ta cập nhật file `max.h`, ta phải
  - Biên dịch lại `principal.c` và `max.c`, sau đó thực thi lại
  - nhưng không cần biên dịch lại `point.c`



Nội dung

1. Module trong C
2. Biên dịch thủ công
3. Cây phụ thuộc
4. **Biên dịch tự động**

Biên dịch thủ công

- Việc biên dịch có thể gây nhàm chán
  - GCC có rất nhiều cờ
  - nhiều lệnh riêng rẽ
  - phải build nhiều target khác nhau
  - nhiều kiến trúc hỗ trợ
  - nhiều file nguồn

  → biên dịch thủ công không thể thực hiện với dự án lớn hoặc các team lớn
- Việc biên dịch thủ công có thể gây ra
  - các vấn đề về sự nhất quán của dự án
  - lãng phí thời gian khi biên dịch

  → dễ gây ra lỗi khi biên dịch

Phần mềm quản lý việc build chương trình

- Build management software (Build Automation)
- Cung cấp một phương thức đơn giản cho việc sinh ra các các file thực thi chương trình.
- Tự động hoá quá trình tiền xử lý, assembling,

GNU Make [1]

- GNU Toolset thực hiện các thao tác bằng cách sử dụng make
  - tiền xử lý,
  - assembling,
  - biên dịch,
  - liên kết,
  - cấp phát bộ nhớ

GNU Make [2]

- GNU Make là một công cụ điều khiển việc phát sinh file thực thi và những file "không phải mã nguồn" của một chương trình từ các file mã nguồn của chương trình.
- Make là một Build Management Software
  - Xác định file nào cần được biên dịch và biên dịch lại trong một dự án được định nghĩa sẵn makefile
  - Hỗ trợ việc phát sinh các phụ thuộc và các file khác trong quá trình build chương trình.
  - Được sử dụng rộng rãi và free.


GNU Toolchain
Linker File
*.s
*.a
*.c
*.h
GNU Make
Executable
Source Files

Makefile

- Một hoặc nhiều file được sử dụng để báo cho make cách build một dự án
  - Được triệu gọi từ dòng lệnh
- Makefile chứa các đích (target) và các quy tắc (rule)
  - Đây là những công thức (recipe) để mô tả cách build một file thực thi hoặc file không phải mã nguồn nào đó.
- Các file thực thi có những phụ thuộc
  - Yêu cầu cần thiết cho một recipe cụ thể
  - Những phụ thuộc này có thể được phát sinh tự động bởi make

File Makefile

- File Makefile là một file text
  - đặt trong cùng thư mục
  - mô tả các phụ thuộc với các quy tắc/luật khi biên dịch
  - được sử dụng bởi lệnh make
- Cú pháp :

```
target :prerequisites ...
        recipe
        ...
        ...
```

Ví dụ

- File Makefile :

```
# My Makefile

exo1 : principal.o point.o max.o
    gcc principal.o point.o max.o -o exo1
principal.o : principal.c point.h max.h
    gcc -Wall -c principal.c
point.o : point.c point.h
    gcc –Wall -c point.c
max.o : max.c max.h
    gcc -Wall -c max.c
```

Việc sử dụng makefile

- Để sinh ra file thực thi, thực hiện lệnh : `make`
- Mục đích:
  - Tìm kiếm `Makefile` hoặc `makefile` trong thư mục hiện hành ;
  - tìm kiếm đích (target) đầu tiên (ở đây là exo1)
  - sau đó thực hiện đệ quy :
    - đối với mỗi phụ thuộc, tìm quy tắc tương ứng ;
    - nếu phụ thuộc mới hơn target, ta cần xây dựng lại bằng lệnh.

Ví dụ [1]

- Ta cập nhật file `max.h`, sau đó thực hiện lệnh `make`
- Target đầu tiên : `exo1`
- Đến các target :
  - `principal.o`
    - `principal.c`, `point.h`, `max.h` không phải là target
    - `max.h` mới hơn `principal.o` → biên dịch lại `principal.o`
  - `point.o`
    - `point.c` và `point.h` không phải là target
    - không thay đổi
  -
```
exo1 : principal.o point.o max.o
    gcc principal.o point.o max.o -o exo1
```
  - `principal.o : principal.c point.h max.h`
  - `gcc -Wall -c principal.c`
  - `point.o : point.c point.h`
  - `gcc -Wall -c point.c`
  -
```
max.o : max.c max.h
    gcc -Wall -c max.c
```

Ví dụ [2]

- Đến target (tiếp theo)
  - max.o
    - max.c và max.h không phải là target
    - max.h mới hơn max.o → biên dịch lại max.o
- principal.o và max.o mới hơn exo1 → biên dịch lại exo1
- Dừng.

```
exo1 : principal.o point.o max.o
    gcc principal.o point.o max.o -o exo1
principal.o : principal.c point.h max.h
    gcc -Wall -c principal.c
point.o : point.c point.h
    gcc -Wall -c point.c
max.o : max.c max.h
    gcc -Wall -c max.c
```

Lựa chọn target

- Ta có thể yêu cầu thực hiện một hoặc nhiều target :
- ```
  $ make exo1
  $ make point.o max.o
  ```
- Nếu tất cả đã được cập nhật, make sẽ không làm gì cả :
- ```
  $ make exo1
  ```
- ```
  make: `exo1' is up to date.
  ```
  ```

Tách riêng các phụ thuộc .h


```
exo1 : principal.o point.o max.o
    gcc principal.o point.o max.o -o exo1
principal.o : principal.c
        gcc -Wall -c principal.c
point.o : point.c
       gcc -Wall -c point.c
max.o : max.c
     gcc -Wall -c max.c

principal.o : point.h max.h
point.o : point.h
max.o : max.h
```

- Mục tiêu : phát sinh tự động phần thứ 3 với lệnh makedepend

Lệnh makedepend

- (Ubuntu : cài đặt package `xutils-dev`)
  Chỉ cần thay thế phần thứ 3 bởi
- depend :
  ```
  makedepend principal.c point.c max.c
  ```
- Sử dụng : `make depend`
  - Thực thi `makedepend principal.c point.c max.c`
  - Thêm vào cuối file `Makefile`: `"#DONOTDELETE"` theo sau là phần thứ 3 được sinh ra bởi `makedepend`

Câu hỏi ?