# MAKEFILE

## GIỚI THIỆU

### Đặt vấn đề

#### Những vấn đề khi biên dịch

* Một chương trình đơn giản Chỉ có một vài file
* Một chương trình “không đơn giản”
* Nhiều dòng lệnh
* Nhiều module
* Nhiều người tham gia viết

#### Vấn đề xảy ra

* Khó quản lý một file lớn (cả người và máy).
* Mỗi thay đổi cần thời gian biên dịch lâu.
* Nhiều người lập trình không thể thay đổi cùng một file đồng thời
* Chương trình được phân ra thành nhiều module

#### Giải pháp

Chia project ra thành nhiều file.

#### Mục tiêu

* Chia project ra thành các modul một cách đúng đắn.
* Thời gian biên dịch phải ngắn nếu có sự thay đổi
* Dễ dàng bảo trì cấu trúc project

### Makefile

#### Makefile là gì?

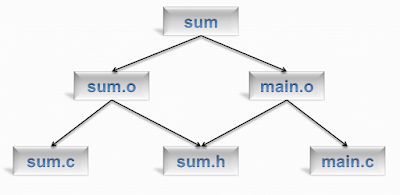
* Makefile là một file dạng **script** chứa các thông tin
* Cấu trúc project (file, sự phụ thuộc)
* Các lệnh để tạo file
* Lệnh **make** sẽ đọc nội dung Makefile, hiểu kiến trúc của project và thực thi các lệnh

#### Cấu trúc project

* Cấu trúc và sự phụ thuộc của project có thể được biểu diễn bằng một DAG (Directed Acyclic Graph)

**Ví dụ:**

* Chương trình chứa 3 file: **main.c**, **sum.c**, **sum.h**
* File sum.h được dùng bởi cả 2 file **main.c** và **sum.c**
* File thực thi là **sum**



|  |  |
| --- | --- |
| sum.h | |
| 1  2  3  4  5 | #ifndef SUM\_H\_  #define SUM\_H\_  #include <stdio.h>  int sum(int a, int b);  #endif /\* SUM\_H\_ \*/ |

|  |  |
| --- | --- |
| sum.c | |
| 1  2  3  4  5 | #include "sum.h"  int sum(int a, int b){      return (a+b);  } |

|  |  |
| --- | --- |
| main.c | |
| 1  2  3  4  5  6  7  8  9 | include <stdio.h>  #include "sum.h"  int main(int argc, char \*\*argv){      int x;      x= sum(1, 2);      printf("x = %d \n", x);      return 1;  } |

|  |  |
| --- | --- |
| Makefile | |
| 1  2  3  4  5  6 | sum: main.o sum.o      gcc -o sum main.o sum.o  main.o: main.c sum.h      gcc -c main.c  sum.o: sum.c sum.h      gcc -c sum.c |

#### Phân tích cấu trúc Makefile

##### Rule

A picture containing text

Description automatically generated

Rule: Có thể có nhiều Rule, trong ví dụ trên có 3 rule

##### Target

A picture containing table

Description automatically generated

Target:

##### Dependency

Diagram

Description automatically generated

Dependency:

##### Action

Diagram

Description automatically generated

Action: Câu lệnh compile để tạo ra Target từ Dependency,

option là **"-c"** để tạo ra file object từ file source code,

option là **"-o"** để tạo ra file chương trình nhị phân từ file object

Mỗi Rule có thể không có hoặc có nhiều Action

**Lưu ý:**

Diagram

Description automatically generated

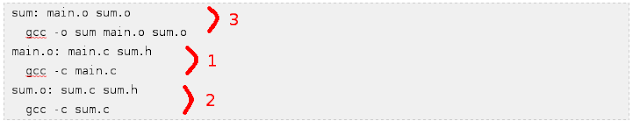
Action được thụt lùi vào một Tab (bằng 4 lần blank space) so với Target

##### Thứ tự thực hiện

Khi bạn thực hiện lệnh make, chương trình make sẽ nhảy đến target đầu tiên là sum với mục đích để tạo ra nó, để làm được điều đó make đi kiểm tra lần lượt (từ trái qua phải: main.o sum.o) xem các dependency của sum đã tồn tại chưa. Denpendency đầu tiên là main.o chưa có, cần phải tìm rule nào đó mà ở đó main.o đóng vai trò là target, make tìm ra rule thứ 2 và nó nhảy đến thực hiện rule thứ 2 để tạo ra main.o (lưu ý khi nó chạy rule 2 thì cũng giống y như khi chạy rule đầu tiên, có thể coi như là đệ quy). Sau khi tạo ra main.o, make trở về rule 1 để tiến hành kiểm tra tiếp xem dependency thứ hai là sum.o đã tồn tại chưa, sum.o chưa có vì thế make tiến hành các bước tương tự như đối với main.o. Sau khi tất cả các dependency được tạo ra, make mới có thể tạo ra file chạy cuối cùng là sum.

Vậy make thực hiện theo nguyên tắc / thứ tự như sau:

* Tạo ra các file object trước (**main.o,** **sum.o)**
* Tạo ra chương trình nhị phân cuối cùng từ các file object đã được tạo ra trước đó (**sum**)



##### Compile & Execute

"**cd**" vào thư mục chứa project và thực hiện lệnh **make**, chương trình **sum** sẽ được tạo ra cùng với 2 file object là **sum.o** và **main.o** , chạy thử chương trình:

|  |
| --- |
| [ninhld@localhost ~]$  [ninhld@localhost ~]$ cd /home/ninhld/Github/eslinuxprogramming/Makefile  [ninhld@localhost Makefile]$  [ninhld@localhost Makefile]$ **make**  **gcc -c main.c**  **gcc -c sum.c**  **gcc -o sum main.o sum.o**  [ninhld@localhost Makefile]$  [ninhld@localhost Makefile]$  [ninhld@localhost Makefile]$ **./sum**  x = 3  [ninhld@localhost Makefile]$  [ninhld@localhost Makefile]$ |

Table

Description automatically generated

#### Nguyên lý biên dịch lại của Makefile

Việc compile lại project dựa vào hai yếu tố:

* Thời gian chỉnh sửa (date modified)
* Cây phụ thuộc trong Makefile
* Có nghĩa là một khi Dependency thay đổi thì Target tương ứng cũng phải được compile lại.

Theo Makefile như trong ví dụ trên:

* Nếu chỉnh sủa **sum.h** thì **main.o** và **sum.o** phải được tạo lại, mặt khác **main.o & sum.o** lại là dependency của **sum** nên **sum** sẽ được tạo lại.

|  |
| --- |
| [ninhld@localhost Makefile]$ make  gcc -c main.c  gcc -c sum.c  gcc -o sum main.o sum.o |

+ Nếu chỉnh sửa **sum.c** thì **sum.o** được tạo lại, đương nhiên **sum** phụ thuộc **sum.o** nên **sum** sẽ được tạo lại.

|  |
| --- |
| [ninhld@localhost Makefile]$ make  gcc -c sum.c  gcc -o sum main.o sum.o |

Khi viết Makefile tránh gộp tất cả lại làm một như dưới đây, vì khi đó nếu một trong các file source code thay đổi thì cũng phải compile lại tất cả các file khác, điều đó làm mất nhiều thời gian:

Makefile

|  |
| --- |
| sum: main.c sum.c sum.h  ${CC} -o sum main.c sum.c sum.h |

### Khuôn dạng đầy đủ của Makefile

sum.h (giữ nguyên)

sum.c (giữ nguyên)

main.c (thêm #ifdef ... #endif)

|  |
| --- |
| #include <stdio.h>  #include "sum.h"    int main(int argc, char \*\*argv){    int x;  x= sum(1, 2);    **#ifdef DEBUG**  printf("x = %d \n", x);  **#endif**    return 1;  } |

Makefile

|  |
| --- |
| .**PHONY**: all, install, clean    TARGET=sum    HDRS+= sum.h  CSRCS+= main.c sum.c  CPPSRCS+=    OBJSDIR=./build  OBJS:= $(patsubst %.cpp, $(OBJSDIR)/%.o, $(CPPSRCS))  OBJS+= $(patsubst %.c, $(OBJSDIR)/%.o, $(CSRCS))    CFLAGS += -I./include -DDEBUG -Wall -g  LDFLAGS += -L./lib -lm    CC:= gcc  CXX:= g++    **all**: ${TARGET}  ${TARGET} : $(OBJS)  @echo " [LINK] $@"  @mkdir -p $(shell dirname $@)  @$(CXX) $(OBJS) -o $@ $(LDFLAGS)    $(OBJSDIR)/%.o: %.c $(HDRS)  @echo " [CC] $@"  @mkdir -p $(shell dirname $@)  @$(CC) -c $< -o $@ $(CFLAGS)    $(OBJSDIR)/%.o: %.cpp $(HDRS)  @echo " [CXX] $@"  @mkdir -p $(shell dirname $@)  @$(CXX) -c $< -o $@ $(CFLAGS)  **install**:  cp -rf ${TARGET} /usr/local/bin    **clean**:  rm -rf ${OBJSDIR}/\*.o  rm -rf ${TARGET} |

**Giải thích:**

#### all, install, clean, .PHONY

##### all

Là target mặc định, khi bạn thực hiện lệnh make thì chương trình make sẽ tiến hành compile để tạo ra chương trình cuối cùng tương ứng với target này, trong ví dụ này là ${TARGET} tương ứng với sum; hai lệnh dưới đây là tương đương

|  |
| --- |
| make  make all |

##### install, clean

Các target do người dùng tự định nghĩa, tương ứng với các câu lệnh:

|  |
| --- |
| make install #thực hiện action cp  make clean #thực hiện action rm |

##### .PHONY target

Target này có tác dụng chống xung đột cho chương trình make trong trường hợp trong thư mục source code có một file source nào đó trùng tên với các target trong Makefile, ví dụ all.c, install.c hay clean.c. Tốt nhất trong Makefile luôn dùng từ khóa này.

#### Phép gán: TARGET, CC, CXX, ...

Các từ in hoa như trên biểu diễn cho phép gán, có thể hiểu nó tương tự như #define trong C, nó giúp cho chương trình sáng sủa, dễ hiểu và rút ngắn Makefile đáng kể. Phép gán được thực hiện bằng các cách sau:

* "**=**" gán cố định
* "**:=**" tương tự như "="
* "**+=**" phép gán nối dài (append), ví dụ khi viết hai dòng

|  |
| --- |
| CSRCS+= main.c  CSRCS+= sum.c  => CSRCS = main.c sum.c |

Để lấy giá trị của phép gán thì dùng các cách sau:

|  |
| --- |
| ${TARGET}  hoặc  $(CFLAGS) |

##### HDRS / CSRCS /CPPSRCS

Header , C source, C++ source file. Trong vd trên các file đều nằm trong folder gốc, nếu file nào nằm ở các thư mục con, bạn cũng cần chỉ đúng đường dẫn đễn file, vd:

|  |
| --- |
| HDRS+= ./src/sum.h  CSRCS+= main.c \  ./src/sum.c  CPPSRCS+= |

"\" là ký tự xuống dòng.

##### OBJSDIR / OBJS

Thư mục chứa các object / các object

##### CFLAGS / LDFLASG

Các flags chứa các options để pass vào cho compiler, xem lại [tại đây](http://eslinuxprogramming.blogspot.com/2015/04/cross-compiler.html) (mục 7).

Chú ý - DDEBUG tương ứng với define DEBUG để pass vào source code, ví dụ trong main.c ở trên.

##### CC / CXX

Compiler để compile, trong vd này sử dụng gcc/g++ để chạy trên máy tính chứ không phải board nhúng, khi [cross compiler](http://eslinuxprogramming.blogspot.com/2015/04/cross-compiler.html) thì bạn chỉ cần đặt đúng compiler tương ứng với platform cho board là được, vd:

|  |
| --- |
| CC=arm-linux-gcc  CXX=arm-linux-g++ |

#### Shell

Lệnh Shell được đưa vào trong cặp khóa **$( )** như dưới đây:

|  |
| --- |
| OBJS:= $(patsubst %.cpp, $(OBJSDIR)/%.o, $(CPPSRCS))  mkdir -p $(shell dirname $@) |

#### Makefile tương đương: $<, $@, $\*, $?, ...

|  |  |
| --- | --- |
| **Macro** | **Definition** |
| $< | Source code hiện tại để compile, tương ứng với %.c và %.cpp, ví dụ: main.c, sum.c |
| $@ | Target hiện tại tương ứng với $(OBJSDIR)/%.o, ví dụ: main.o, sum.o |
| $\* | Tương tự $< nhưng không có suffix, ví dụ: main, sum |
| $? | Danh sách dependency tương ứng với %.c $(HDRS), ví dụ: tương ứng với main.o là main.c và sum.h |

Được sử dụng trong đoạn Makefile chính:

|  |
| --- |
| ${TARGET} : $(OBJS)  @echo " [LINK] $@"  @mkdir -p $(shell dirname $@)  @$(CXX) $(OBJS) -o **$@** $(LDFLAGS)    $(OBJSDIR)/%.o: %.c $(HDRS)  @echo " [CC] $@"  @mkdir -p $(shell dirname $@)  @$(CC) -c $< -o $@ $(CFLAGS)    $(OBJSDIR)/%.o: %.cpp $(HDRS)  @echo " [CXX] $@"  @mkdir -p $(shell dirname $@)  @$(CXX) -c $< -o $@ $(CFLAGS) |

Bạn đọc có thể trace các macro bằng các dòng lệnh sau:

|  |
| --- |
| @echo " [CC] $@ $< $\* $?"  @echo " [CXX] $@ $< $\* $?" |

**Nguyên lý hoạt động:**

Chỉ cần đoạn lệnh ngắn gọn trên là đủ để để compile tất cả các target được liệt kê trong OBJS thông qua ký tự đại diện "%"; compile main.o từ main.c, compile sum.o từ sum.c, tất nhiên có các dependency HDRS

3.5 Các ký tự khác

* @ ở đầu các Action trong Makefile

Loại bỏ các dòng trace mặc định của chương trình make khi đang compile

* -f

Nếu bạn đặt tên của script không phải mặc định là Makefile (ví dụ: MyMakefile) thì bạn cần thêm option -f vào

|  |
| --- |
| make -f MyMakefile  make -f MyMakefile install  make -f MyMakefile clean  ... |

### Tạo Makefile theo kiểu module cho project lớn

Nếu project lớn có nhiều subfolder thì bạn nên chia thành nhiều phần cho dễ quản lý, cũng theo vd trên nhưng bây giờ giả sử sum.h và sum.c ở trong một subfolder gọi là src



Trong trường hợp này bạn thêm một makefile phụ là src.mk đặt trong thư mục src

|  |
| --- |
| src.mk |
| CSRCS+= ./src/sum.c  CPPSRCS+=    HDRS+= $(wildcard ./src/\*.h) |

|  |
| --- |
| Makefile |
| .PHONY: all, install, clean    TARGET:=sum    **-include ./src/src.mk**  **CSRCS+= main.c**    OBJSDIR=./build  OBJS:= $(patsubst %.cpp, $(OBJSDIR)/%.o, $(CPPSRCS))  OBJS+= $(patsubst %.c, $(OBJSDIR)/%.o, $(CSRCS))    **INCDIR+= -I./src**  CFLAGS += -DDEBUG -Wall -g  LDFLAGS += -L./lib -lm        CC:= gcc  CXX:= g++    all: ${TARGET}  ${TARGET} : $(OBJS)  @echo " [LINK] $@"  @mkdir -p $(shell dirname $@)  @$(CXX) $(OBJS) -o $@ $(LDFLAGS)    $(OBJSDIR)/%.o: %.c $(HDRS)  @echo " [CC] $@"  @mkdir -p $(shell dirname $@)  @$(CC) -c $< -o $@ $(CFLAGS) ${**INCDIR**}    $(OBJSDIR)/%.o: %.cpp $(HDRS)  @echo " [CXX] $@"  @mkdir -p $(shell dirname $@)  @$(CXX) -c $< -o $@ $(CFLAGS) ${**INCDIR**}    install:  cp -rf ${TARGET} /usr/local/bin    clean:  rm -rf ${OBJSDIR}  rm -rf ${TARGET} |