

幾個Makefile通用模板分享！

嵌入式大雜燴 2022-02-15 21:30

收錄於話題

#嵌入式

169個

原文：https://blog.csdn.net/qq_20553613/article/details/90649734

大家好，我是ZhengN。

本次給大家帶來三個Makefile模板：編譯可執行程序、編譯靜態庫、編譯動態庫。

往期相關推文：[Makefile常用基礎知識梳理！](#)

1、寫在前面

對於Windows下開發，很多IDE都集成了編譯器，如Visual Studio，提供了“一鍵編譯”，編碼完成後只需一個操作即可完成編譯、鏈接、生成目標文件。

Linux開發與Windows不同，Linux下一般用的gcc/g++編譯器，如果是開發ARM下的Linux程序，還需用到arm-linux-gcc/arm-linux-g++交叉編譯器。

Linux下也可以實現“一鍵編譯”功能，此時需要一個編譯腳本“Makefile”，Makefile可以手動編寫，也可以藉助自動化構建工具（如scons、CMake）生成。手動編寫Makefile是Linux和Windows程序員的區別之一，一般地一個通用的Makefile能夠適合大部分Linux項目程序。

2、3個Makefile模板

2.1 編譯可執行文件Makefile

```
VERSION  =1.00
CC       =gcc
DEBUG    =-DUSE_DEBUG
CFLAGS   =-Wall
SOURCES  =$(wildcard ./source/*.c)

INCLUDES =-I./include
LIB_NAMES =-lfun_a -lfun_so
LIB_PATH =-L./lib
```

```
OBJ    =$(patsubst %.c, %.o, $(SOURCES))
TARGET =app

#links
$(TARGET):$(OBJ)
@mkdir -p output
$(CC) $(OBJ) $(LIB_PATH) $(LIB_NAMES) -o output/$(TARGET)$(VERSION)
@rm -rf $(OBJ)

#compile
%.o: %.c
$(CC) $(INCLUDES) $(DEBUG) -c $(CFLAGS) $< -o $@

.PHONY:clean
clean:
@echo "Remove linked and compiled files....."
rm -rf $(OBJ) $(TARGET) output
```

【要點說明】

【1】程序版本

開發調試過程可能產生多個程序版本，可以在目標文件後（前）增加版本號標識。

```
VERSION = 1.00
$(CC) $(OBJ) $(LIB_PATH) $(LIB_NAMES) -o output/$(TARGET)$(VERSION)
```

【2】編譯器選擇

Linux下為gcc/g++；arm下為arm-linux-gcc；不同CPU廠商提供的定制交叉編譯器名稱可能不同，如Hisilicon“arm-hisiv300-linux-gcc”。

```
CC = gcc
```

【3】宏定義

開發過程，特殊代碼一般增加宏條件來選擇是否編譯，如調試打印輸出代碼。-D是標識，後面接著的是“宏”。

```
DEBUG = -DUSE_DEBUG
```

【4】編譯選項

可以指定編譯條件，如顯示警告（-Wall），優化等級（-O）。

```
CFLAGS = -Wall -O
```

【5】源文件

指定源文件目的路徑，利用“wildcard”獲取路徑下所有依賴源文件。

```
SOURCES = $(wildcard ./source/*.c)
```

【6】頭文件

包含依賴的頭文件，包括源碼文件和庫文件的頭文件。

```
INCLUDES = -I./include
```

【7】庫文件名稱

指定庫文件名稱，庫文件有固定格式，靜態庫為libxxx.a;動態庫為libxxx.so，指定庫文件名稱只需寫“xxx”部分，

```
LIB_NAMES = -lfun_a -lfun_so
```

【8】庫文件路徑

指定依賴庫文件的存放路徑。注意如果引用的是動態庫，動態庫也許拷貝到“/lib”或者“/usr/lib”目錄下，執行應用程序時，系統默認在該文件下索引動態庫。

```
LIB_PATH = -L./lib
```

【9】目標文件

調用“patsubst”將源文件（.c）編譯為目標文件（.o）。

```
OBJ=$(patsubst %.c, %.o, $(SOURCES))
```

【10】執行文件

執行文件名稱

```
TARGET =app
```

【11】編譯

```
%.o: %.c  
$(CC) $(INCLUDES) $(DEBUG) $(CFLAGS) $< -o $@
```

【12】鏈接

可創建一個“output”文件夾存放目標執行文件。鏈接完輸出目標執行文件，可以刪除編譯產生的臨時文件（.o）。

```
$(TARGET):$(OBJ)  
@mkdir -p output  
$(CC) $(OBJ) $(LIB_PATH) $(LIB_NAMES) -o output/$(TARGET).$(VERSION)  
@rm -rf $(OBJ)
```

【13】清除編譯信息

執行“make clean”清除編譯產生的臨時文件。

```
.PHONY:clean  
clean:  
@echo "Remove linked and compiled files....."  
rm -rf $(OBJ) $(TARGET) output
```

2.2 編譯靜態庫Makefile

```
VERSION      =
CC            =gcc
DEBUG        =
CFLAGS       =-Wall
AR            =ar
ARFLAGS       =rv
SOURCES      =$(wildcard *.c)
INCLUDES      =-I.
LIB_NAMES     =
LIB_PATH      =
OBJ          =$(patsubst %.c, %.o, $(SOURCES))
TARGET        =libfun_a

#link
$(TARGET):$(OBJ)
    @mkdir -p output
    $(AR) $(ARFLAGS) output/$(TARGET)$(VERSION).a $(OBJ)
    @rm -rf $(OBJ)

#compile
%.o: %.c
    $(CC) $(INCLUDES) $(DEBUG) -c $(CFLAGS) $< -o $@

.PHONY:clean
clean:
    @echo "Remove linked and compiled files....."
    rm -rf $(OBJ) $(TARGET) output
```

【要點說明】

基本格式與“編譯可執行Makefile”一致，不同點包括以下。

【1】使用到“ar”命令將目標文件（.o）鏈接成靜態庫文件（.a）。靜態庫文件固定命名格式為：libxxx.a。

2.3 編譯動態庫Makefile

```
VERSION      =
CC            =gcc
DEBUG        =
```

```

CFLAGS    =-fPIC -shared
LFLAGS    =-fPIC -shared
SOURCES   =$(wildcard *.c)
INCLUDES  =-I.
LIB_NAMES =
LIB_PATH  =
OBJ        =$(patsubst %.c, %.o, $(SOURCES))
TARGET     =libfun_so

#link
$(TARGET):$(OBJ)
    @mkdir -p output
    $(CC) $(OBJ) $(LIB_PATH) $(LIB_NAMES) $(LFLAGS) -o output/$(TARGET)$(VERSION).so
    @rm -rf $(OBJ)

#compile
%.o: %.c
    $(CC) $(INCLUDES) $(DEBUG) -c $(CFLAGS) $< -o $@

.PHONY:clean
clean:
    @echo "Remove linked and compiled files....."
    rm -rf $(OBJ) $(TARGET) output

```

【要點說明】

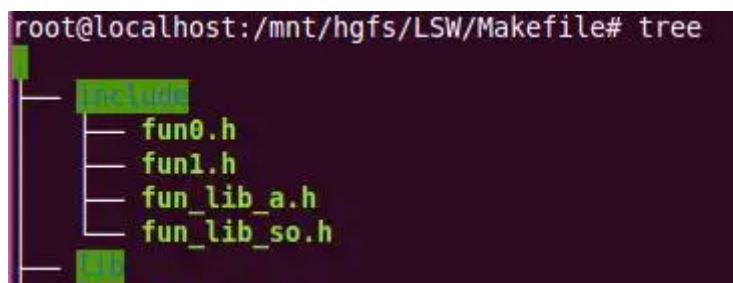
基本格式與“編譯可執行Makefile”一致，不同點包括以下。

【1】編譯選項和鏈接選項增加“-fPIC -shared”選項。動態庫文件固定命名格式為libxxx.so。

3、Demo

3.1 編譯應用程序

編寫測試例程，文件存放目錄結構如下，頭文件存放在“include”目錄，庫文件存放在“lib”目錄，源文件存放在“source”目錄，Makefile在當前目錄下。



```

root@localhost:/mnt/hgfs/LSW/Makefile# tree
.
├── include
│   ├── fun0.h
│   ├── fun1.h
│   ├── fun_lib_a.h
│   └── fun_lib_so.h
└── lib

```



源碼1:

```
/*头文件*/
#ifndef _FUN0_H_
#define _FUN0_H_
#endif

extern void fun0_printf(void);
extern void fun1_printf(void);

/*源文件*/
#include <stdio.h>
#include "fun0.h"

void fun0_printf(void)
{
    printf("Call \'fun0\'. \r\n");
}
```

源碼2:

```
/*头文件*/
#ifndef _FUN1_H_
#define _FUN1_H_
#endif

extern void fun1_printf(void);

/*源文件*/
#include <stdio.h>
#include "fun1.h"
```

```
void fun1_printf(void)
{
    printf("Call \'fun1\'.\r\n");
}
```

主函數源碼：

```
/*源文件*/
#include <stdio.h>
#include "fun0.h"
#include "fun1.h"
#include "fun_lib_a.h"
#include "fun_lib_so.h"

int main(void)
{
    #ifdef USE_DEBUG
        printf("Debug Application startup.\r\n");
    #endif

    fun0_printf();
    fun1_printf();
    fun_lib_a_printf();
    fun_lib_so_printf();
    return 0;
}
```

庫文件，“./lib”目錄下存放兩個庫文件，一個靜態庫libfun_a.a，一個動態庫libfun_so.so。

Makefile文件即為“2.1節”的Makefile模板。

測試運行：

```
root@localhost:/mnt/hgfs/LSW/Makefile# make
gcc -I./include -DUSE_DEBUG -Wall -c source/fun0.c -o source/fun0.o
gcc -I./include -DUSE_DEBUG -Wall -c source/fun1.c -o source/fun1.o
gcc -I./include -DUSE_DEBUG -Wall -c source/main.c -o source/main.o
gcc ./source/fun0.o ./source/fun1.o ./source/main.o -L./lib -lfun_a -lfun_so -o output/app1.00
root@localhost:/mnt/hgfs/LSW/Makefile# ./output/app1.00
Debug Application startup.
Call 'fun0'.
Call 'fun1'.
Call 'fun_lib_a'.
Call 'fun_lib_so'.
```

【如果執行文件提示無“libfun_so.so”，則需拷貝“libfun_so.so”到根目錄下的“/lib”或者“/usr/lib”目錄下，因為系統執行程序，默認從該路徑引腳動態庫】

3.2 生成靜態庫

編寫測試例程，生產的庫文件即為“3.1節”調用的庫文件（libfun_a.a）。文件存放目錄結構如下：

```
root@localhost:/mnt/hgfs/LSW/Mlib/source_a# tree
.
├── fun_lib_a.c
├── fun_lib_a.h
└── Makefile

0 directories, 3 files
```

源文件：

```
/*头文件*/
#ifndef _FUN_LIB_A_H_
#define _FUN_LIB_A_H_
#endif

extern void fun_lib_a_printf(void);

/*源文件*/
#include <stdio.h>
#include "fun_lib_a.h"

void fun_lib_a_printf(void)
{
    printf("Call \'fun_lib_a\'.\r\n");
}
```

Makefile文件即為“2.2節”的Makefile模板。

編譯生成靜態庫：

```
root@localhost:/mnt/hgfs/LSW/Mlib/source_a# make
gcc -I. -c -Wall fun_lib_a.c -o fun_lib_a.o
ar rv output/libfun_a.a fun_lib_a.o
ar: creating output/libfun_a.a
a - fun_lib_a.o
ranlib output/libfun_a.a
```

3.3 生成動態庫

編寫測試例程，生產的庫文件即為“3.1節”調用的庫文件（libfun_so.so）。文件存放目錄結構如下：

```
root@localhost:/mnt/hgfs/LSW/Mlib/source_so# tree
.
├── fun_lib_so.c
├── fun_lib_so.h
└── Makefile

0 directories, 3 files
```

源文件：

```
/*头文件*/
#ifndef _FUN_LIB_SO_H_
#define _FUN_LIB_SO_H_
#endif

extern void fun_lib_so_printf(void);

/*头文件*/

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

void fun_lib_so_printf(void)
{
    printf("Call \'fun_lib_so\'.\r\n");
}
```

編譯生成動態庫：

```
root@localhost:/mnt/hgfs/LSW/Mlib/source_so# make
gcc -I. -c -fPIC -shared fun_lib_so.c -o fun_lib_so.o
gcc fun_lib_so.o -fPIC -shared -o output/libfun_so.so
```

溫馨提示

由於微信公眾號近期改變了推送規則，如果您想經常看到我們的文章，可以在每次閱讀後，在頁面下方點一個「贊」或「在看」，這樣每次推送的文章才會第一時間出現在您的訂閱列

表裡。

免責聲明：本文來源網絡，免費傳達知識，版權歸原作者所有。如涉及作品版權問題，請聯繫我進行刪除。

往期推薦：

[跨平台構建工具，cmake是yyds？bjd！](#)

[C語言、嵌入式中幾個非常實用的宏技巧](#)

[分享一個自用的、極簡的log模塊！](#)

[分享一個很酷的IDE！軟工必備](#)

[C語言、嵌入式位操作精華技巧大匯總](#)

[嵌入式大雜燴週記|第1期](#)

[Hello系列|cmake簡明基礎知識](#)

[乾貨|項目乏力？nanopb助你一臂之力](#)



Linux大陸

Hello Linux

13篇原創內容

公眾號



嵌入式大雜燴

本公眾號專注於嵌入式技術，包括但不限於C/C++、嵌入式、物聯網、Linux等編程學...

267篇原創內容

公眾號

在公眾號聊天界面回復**1024**，可獲取嵌入式資源；回復**m**，可查看文章匯總。

點擊**閱讀原文**，查看更多分享。

收錄於話題#嵌入式 169

[閱讀原文](#)

喜歡此內容的人還喜歡

18000 字的SQL優化大全

浪尖聊大數據

深入SpringBoot排查@Transactional引起的空異常！

業餘草

推薦MyBatis的一個簡單配置搞定加解密！

業餘草