chapter 9 建立與使用函式庫

9-1 函式庫簡介

函式庫(library)是一些事先編譯過的函數集合，善用函數庫可以簡化程式的開發。在Linux中標準的系統函數庫通常放在目錄/lib和/usr/lib，函式庫的名稱都是以lib開頭，再加上自定的名稱，附屬檔名為.a或.so。名稱加上.a的函式庫代表這是一個靜態的函式庫，名稱加上.so的函式庫代表這是一個動態的函式庫。

9-1.1 靜態函式庫與共享函式庫

靜態的函式庫可以使用ar工具來產生。

當編譯應用程式時，使用靜態的函式庫，則函式庫的元件會連結到執行檔，產生的執行檔較大，但以後執行程式時不需要函式庫。

當編譯應用程式時，使用動態的函式庫，則在產生應用程式的執行檔時，連結器(linker)會檢查函式庫中使用到的元件其函數名稱和變數。在應用程式執行時，動態載入器(loader)會載入共享函數庫中所需元件到記憶體，再動態連結到應用程式。其優點是產生的執行檔較小，缺點是執行時需要有函式庫的配合。

動態連結程式庫(Shared library)是在程式開始執行時才載入的，其優點在於

(1)減少執行檔的大小，

(2)更新程式庫而無需重新編譯其他程式，以及

(3)甚至可在程式執行時更改程式庫。

在使用shared library前，你需要知道與shared library有關的名詞：

soname：
每一個shared library都有一個以「lib」開頭的程式庫名稱，然後加上程式庫的名稱，在名稱末端再加上「.so」，以及period(i.e.「.」號)與版本號碼。一個全稱程式庫名稱(fully-qualified soname)應該是「libxxxx.so.N」，「xxxx」是程式庫名稱，「N」是版本號碼。
real name： 真正載有已編譯程式碼的檔案名稱，傳統上棋檔名要包含「lib」、程式庫名稱、「.so」、主次版本號碼及發佈版本號碼，例如「libxxxx.so.N1.N2.N3」。

linker name： 編譯器所搜尋的程式庫名稱，傳統上就是real name刪去所有版本號碼後的名稱，例如「libxxxx.so」

9-1.2 使用函式庫

在預設的情況下，Linux的C語言編譯器只會使用C函數庫libc，若在編譯時還需要其他的函數庫，例如數學函數庫libm，則編譯時必須指示編譯器額外使用的函數庫。例如在編譯hello.c時，同時使用數學函數庫libm..a。

#gcc -o hello hello.c /usr/lib/libm.a

也可以在gcc指令加上-l選項，指示編譯器使用的函式庫。例如：

#gcc -o hello hello.c -lm

其中-lm表示使用libm函式庫，注意l和m中間不可以有空白，m表示libm.a函數庫。

10-2 製作靜態函式庫

函數庫的功用就是收集一些已經編譯好的函數，當撰寫一個應用程式時，若需用到函數庫中的一個函數時，必須先引用一些標頭檔。在這些標頭檔中會宣告該函數的原型。

靜態函數庫的檔案結尾一般為.a，製作一個靜態函數庫可以用ar指令，其步驟如下：

步驟1：

將準備加入函數庫的程式檔案編譯成目的檔案。例如下列指令將initapi.c和randapi.c編譯成為initapi.o和randapi.o。

#gcc -c -Wall initapi.c

#gcc -c Wall randapi.c

步驟2：

執行ar指令，將目的檔案加入靜態函數庫中。例如下列指令將initapi.o和randapi.o加入函數庫libmyrand.a。

#ar -cru libmyrand.a initapi.o randapi.o

其中參數-cru 用來建立或加入目的檔案，c建立靜態函數庫，r告知ar取代已經存在的目的檔案，u表示取代的目的檔案必須比現存的目的檔案還新。

9-3 製作共享函式庫

使用靜態函數的缺點是在同一時間執行的許多應用程式都使用到相同的函數，則每一個應用程式都會有相同的函數占用記憶體，造成相同函數重複佔用記憶體的問題。

當一個應用程式使用共享函數庫，編譯連結後的程式碼不會包含函數庫的程式碼，只有在執行時，程式被載入記憶體才會去解決會呼叫函數的問題。系統必須準備一份共享函數庫的程式碼，當應用程式執行時，函數的程式碼只會佔據一份記憶體，沒有重複占用記憶體的問題。當函數庫更新時，不用每一個程式都重新更新。其製作過程如下：

步驟1：

告知gcc編譯器欲產生的目的檔案屬於position-independence，即以後建立的函數庫不會與應用程式綁在一起。使用的指令如下：

#gcc -fPIC -c initapi.c

#gcc -fPIC -c randapi.c

步驟2：

使用-share告知gcc建立共享函數庫，下列指令會建立一個共享函數庫libmyrand2.so。

#gcc -shared initapi.o randapi.o -o libmyrand2.so

9-4 實作函式庫與共享函式庫

目的：

練習靜態函數庫和共享函數庫的製作。

要求：

1. 編輯程式檔initapu.c和randapi.c。

2. 編輯測試主程式，和相關標頭檔。

3. 以多檔案編輯的方式驗證initapi.c和randapi.c的正確性。

4. 將initapi.c和randapi.c加入靜態函數庫。

5. 練習編譯主程式時使用靜態函數庫。

6. 將initapi.c和randapi.c加入動態函數庫。

7. 練習編譯主程式時使用動態函數庫。

步驟1：

編輯程式碼initapu.c，其中的init_rand()函數在呼叫後會重新設定亂數種子。

```
#include <stdlib.h>
#include <time.h>

void init_rand()
{
    time_t seed;
    seed=time(NULL);
    srand(seed);
}
```

[說明]

C語言的標準中內建了亂數函數srand()以及rand()。使用亂數前必須呼叫 srand()函式將亂數函數中的種子值(seed)初始化，不先呼叫本函數、或種子值固定，都會造成新手常見的「每次執行產生的亂數結果都相同」問題。為了方便起見可以使用 time(NULL)這個系統內的時鐘作為seed，使每次執行產生的亂數序列不同。一般而言，只需在程式開始使用亂數前呼叫一次srand()即可，不必重複呼叫 (也不要不小心把srand包在迴圈裏面)。rand()是一個亂數函數，每呼叫一次rand()會傳出一個新的亂數。

步驟2：

編輯randapi.c，此程式包含2個函數，get_rand()函數在呼叫後會產生0-1之間的實數亂數，get_maxrand(int max)函數在呼叫後會產生0-(max-1)間的整數亂數。

```
#include <stdlib.h>
float get_rand()
{
    float rand_value;
    rand_value=((float)rand()/(float)RAND_MAX);
    return rand_value;
```

```
}

int get_maxrand(int max)
{
    int rand_value;
    rand_value=(int)((float)max*rand()/(RAND_MAX+1.0));
    return rand_value;
}
```

[說明]

怎樣獲得在一定範圍內的隨機數？

直接的方法是
rand() % N /* 不好 */
試圖返回從 0 到 N - 1 的數字。但這個方法不好, 因為許多隨機數 發生器的低位比特並不隨機。 一個較好的方法是:
(int)((double)rand() / ((double)RAND_MAX + 1) * N)
如果你不希望使用浮點, 另一個方法是:
rand() / (RAND_MAX / N + 1)
兩種方法都需要知道 RAND_MAX, 而且假設 N 要遠遠小於 RAND_MAX。 RAND_MAX 在 ANSI 裡 #define 在 <stdlib.h>。
RAND_MAX 是個常數, 它告訴你 C 庫函數 rand() 的固定範圍。你不可以設 RAND_MAX 為其它的值, 也沒有辦法要求 rand() 返回其它範圍的值。

如果你用的隨機數發生器返回的是 0 到 1 的浮點值, 要取得範圍在 0 到 N - 1 內的整數, 只要將隨機數乘以 N 就可以了。

RAND_MAX這個常數是rand()所能產生的最大亂數，定義在stdlib.h裡面。rand()產生的亂數範圍為： 0 <= rand() <= RAND_MAX。RAND_MAX的值隨著compiler的不同可能會有所變化。一些compiler如Dev-C++裡面RAND_MAX的值為215-1=32767，另一個常見的RAND_MAX的值為231-1=2147483647。您可以print出RAND_MAX來確認。一般而言RAND_MAX的值愈大愈好。

步驟3：

編輯相關標頭檔randapi.h，此檔案包含init_rand()，get_rand()和get_maxrand()函數的原型須告。

```
#ifndef _RAND_API_H
#define _RAND_API_H

extern void init_rand(void);
extern float get_rand(void);
extern int get_maxrand(int max);
#endif
```

步驟4：

編輯測試主程式test.c，此程式型後會印出6個介於0-1之間的實數亂數，6個介於0-5之間的整數亂數。

```
#include "randapi.h"
#include <stdio.h>
int main()
{
    int i;

    init_rand();
    printf("get_rand():\n");
    for (i=0; i<6; i++) {
      printf("%f ",get_rand());
    }
    printf("\n");
    printf("get_maxrand(6)\n");
    for (i=0; i<6; i++) {
      printf("%d ",get_maxrand(6));
    }
    printf("\n");
    return 0;
}
```

步驟5：

以多檔案編譯方式測試test.c是否可以正常編譯和執行。執行後會印出6個介於0-1之間的實數亂數，6個介於0-5之間的整數亂數。

```
#gcc initapi.c randapi.c test.c -o test
./test
get_rand():
0.349672 0.289722 0.718103 0.381626 0.928865 0.243624
get_maxrand(6):
4 1 0 4 0 3
```

步驟6：

將initapi.c和randapi.c加入靜態函數庫libmyrand.a

```
#gcc -c Wall initapi.c
#gcc -c Wall randapi.c
#ar -cru libmyrand.a initapi.o randapi.o
```

步驟7：

連結靜態函數庫，重新編譯test.c，產生testa執行檔。

```
#gcc test.c -L. -lmyrand -o testa
#./testa
get_rand(): 0.349672 0.289722 0.718103 0.381626 0.928865 0.243624 get_maxrand(6): 4 1 0 4 0 3
```

步驟8：

將initapi.c和randapi.c加入共享函數庫libmyrand2.so

```
#gcc -fPIC -c initapi.c
#gcc -fPIC -c randapi.c
#gcc -shared initapi.o randapi.o libmyrand2.so
```

步驟9：

連結共享函數庫，重新編譯test.c，產生testo執行檔。

```
#gcc test.c -L. -lmyrand2 -o testo
#./testo
./testo: error while loading shared libraries:
```

執行testo時需要libmyrand2.so，但是Linux動態載入器找不到這個檔案。

步驟10：

在linux中有一個很好用的指令ldd，可以幫忙查看testo執行時需要的相觀共享函數庫。

```
#ldd testo
    linux-gate.so.1 => (0x00b92000)
    libmyrand2.so => not found
    lib.c.so.6 => /lib/libc.so.6 (0x00869000)
    /lib/ld-linux.so.2 (0x00847000)
```

步驟11：

原來Linux只會到/lib和/usr/lib目錄下找尋共享函數庫，但是在步驟8產生的libmyrand2.so函數庫位於應用程式所在目錄，才在造成執行上的錯誤。

解決的方法是設定LD_LIBRARY_PATH變數，告知Linux共享函數庫位於目前的應用程式所在目錄。設定好後，再執行ldd指令，可看到已經找到libmyran2.so。

[說明]

當我們用到數學函式cos()，cos這個symbol，gcc並不曉它到底是什麼東西， 是變數，是函式，要預留多少空間給他等等，完全沒有任何訊息，你必須標頭 檔要#include ，gcc才知道。而且因為specs這個檔裡面只有要 link -lc也就是只有libc.so這個檔內的symbol會被蒐尋， 像printf scanf等都在這裡面，可是像cos()等就沒有了， 所以函式庫的選項要多加 -lm ，這時ld才會來找libm這個函式庫，

編譯的時候，gcc會去找-L，再找gcc的環境變數LIBRARY_PATH，再找內定目錄 /lib /usr/lib /usr/local/lib 這是當初compile gcc時寫在程式內的， gcc環境變數與pass給ld的機制在~gcc/gcc/collect2.c下找得到。 這上面只是搜尋路徑而已，如果要不加-lm 也能正確的主動搜尋某個特定的lib，例如libm， 就要去在specs這個檔案改一下，把math這個函式庫加進自動聯結函式庫 之一。就不用寫-lm了。

RUN TIME的時候， 如果編譯時沒有指定-static這個選項，其實可執行檔並不是真的可執行， 它必須在執行(run time)時需要ld.so來做最後的連結動作，建造一個可執行的 image丟到記憶體。如果是靜態連結，編譯時ld會去找libm.a的檔 。如果是動態連結去找libm.so。 所以每次有新改版程式， 或新加動態函式庫如果不在原本的/etc/ld.so.conf搜尋路徑中，都要把路徑 加進來，然後用

ldconfig -v

會重建cache並且顯示它所參照的函式庫。

```
#export LD_LIBRARY_PATH=./
#ldd testo
linux-gate.so.1 => (0x00b92000) libmyrand2.so => ./libmyrans2.so (0x00c70000) lib.c.so.6 => /lib/libc.so.6 (0x00869000) /lib/ld-linux.so.2 (0x00847000)
#./testo
get_rand(): 0.349672 0.289722 0.718103 0.381626 0.928865 0.243624 get_maxrand(6): 4 1 0 4 0 3
```

步驟12：

步驟11的缺點是每次執行testo都需要設定一次LD_LIBRARY_PATH變數，比較簡便的方法是將libmyrans2.so複製到/usr/lib目錄。

```
#cp libmyrans2.so /usr/lib
#ldd
linux-gate.so.1 => (0x00b92000) libmyrand2.so => ./libmyrans2.so (0x00c70000) lib.c.so.6 => /lib/libc.so.6 (0x00869000) /lib/ld-linux.so.2 (0x00847000) #./
```

9-5 AR指令

在Linux可以用ar指令建立靜態函數庫，其維護也可以用ar指令。

顯示靜態函式庫中包含的目的檔案

在ar指令中加入-t選項可顯示靜態函式庫中包含的目的檔案。

```
#ar -t libmyrand.a
initapi.o
randapi.o
```

刪除靜態函式庫中的目的檔案

在ar指令中加入-d選項可刪除靜態函式庫中的目的檔案。

```
#ar -d libmyrand.a initapi.o
#ar -t libmyrand.a
randapi.o
```

顯示詳細資訊

在ar指令中加入-d選項可顯示詳細資訊。

```
#ar -dv libmyrand.a initapi.o
No member named 'initapi.o'
```

插入目的檔案於指定的靜態函式庫

在ar指令中加入-ru選項可插入目的檔案於指定的靜態函式庫。

```
#ar -ru libmyrand.a initapi.o
#ar -t libmyrand.a
randapi.o
initapi.o
```

取出靜態函式庫中的目的檔案

在ar指令中加入-x選項可取出靜態函式庫中的目的檔案

```
#ls *.o
initapi.o randapi.o
#rm initapi.o
rm:是否移除普通檔案'initapi.o'? y
#ls *.o
randapi.o
#ar -xv libmyrand.a initapi.o
x - initapi.o
#ls *.o
initapi.o randapi.o
```

總結：

- 函式庫：目標檔庫
  1. 靜態函式庫：
     1. 附檔名：通常為 libxxx.a 的類型；
     2. 編譯行為：整個函式庫的資料被整合到執行檔中，所以利用編譯成的檔案會比較大。
     3. 獨立執行的狀態：編譯成功的可執行檔可以獨立執行，而不需要再向外部要求讀取函式庫的內容。
     4. 升級難易度：函式庫升級後，連執行檔也需要重新編譯過一次，將新的函式庫整合到執行檔當中。
  2. 動態函式庫：
     1. 附檔名：通常為 libxxx.so 的類型；
     2. 編譯行為：編譯時執行檔中僅具有指向動態函式庫所在的指標而已，並不包含函式庫的內容，所以檔案會比較小。
     3. 獨立執行的狀態：不能被獨立執行，程式讀取函式庫時，函式庫『必須要存在』，且函式庫的『所在目錄也不能改變』。
     4. 升級難易度：函式庫升級後，執行檔不需要進行重新編譯，故目前的 Linux distribution 比較傾向使用動態函式庫。
- Linux 放置函式庫之目錄：

```
/usr/lib
/lib
# kernel 的函式庫放在 /lib/modules
```

- 如何判斷某個可執行的 binary 檔案含有什麼動態函式庫？
  1. ldd 指令

```
[root@linux ~]# ldd [-vdr] [filename]
選項：
--version ：列印 ldd 的版本序號
-v ：列出所有內容資訊；
--help ：指令用法資訊
```

  2. 找出檔案 /usr/bin/passwd 的函式庫資料。

```
[root@linux ~]# ldd /usr/bin/passwd
        linux-gate.so.1 =>  (0x00d19000)
        .......中間省略......
        libpam_misc.so.0 => /lib/libpam_misc.so.0 (0x00bd6000)
        .......中間省略......
```

  3. 找出函式 /lib/libc.so.6 的相關其他函式庫。

```
[root@linux ~]# ldd /lib/libc.so.6
        /lib/ld-linux.so.2 (0x00bf1000)
        linux-gate.so.1 =>  (0x00632000)

[root@linux ~]# ldd -v /lib/libc.so.6
        /lib/ld-linux.so.2 (0x00bf1000)
        linux-gate.so.1 =>  (0x00111000)

        Version information:
        /lib/libc.so.6:
                ld-linux.so.2 (GLIBC_2.1) => /lib/ld-linux.so.2
                ld-linux.so.2 (GLIBC_2.3) => /lib/ld-linux.so.2
                ld-linux.so.2 (GLIBC_PRIVATE) => /lib/ld-linux.so.2
```

- 如何將動態函式庫載入快取記憶體(cache)，以增進動態函式庫的讀取速度？
  1. 在 /etc/ld.so.conf 寫入想要讀入快取記憶體的動態函式庫所在的目錄；
  2. 利用執行檔 ldconfig 將 /etc/ld.so.conf 的資料讀入快取中；
  3. 同時也將資料記錄一份在檔案 /etc/ld.so.cache 中。

```
[root@test root]# ldconfig [-f conf] [-C cache] [-p]
參數說明：
-f conf ：conf 預設為 /etc/ld.so.conf
-C cache：cache 預設為 /etc/ld.so.cache
-p      ：列出目前在 cache 內的資料
```

  4. 例題：將 xorg 相關函式庫加到快取記憶體中：

```
[root@dywHome2 ~]# cat /etc/ld.so.conf
include ld.so.conf.d/*.conf
/usr/X11R6/lib
/usr/lib/qt3/lib
[root@dywHome2 ~]# ldconfig -p | grep "libafb"
[root@dywHome2 ~]# vi /etc/ld.so.conf
[root@dywHome2 ~]# cat /etc/ld.so.conf
include ld.so.conf.d/*.conf
/usr/X11R6/lib
/usr/lib/qt3/lib
/usr/lib/xorg/modules # 加入此行
[root@dywHome2 ~]# ldconfig
[root@dywHome2 ~]# ldconfig -p | grep "libafb"
        libafb.so (libc6) => /usr/lib/xorg/modules/libafb.so
```

- 建立函式庫：將多個目標檔案組合成一個函式庫檔案
  1. 函式庫建立指令 ar。例如：

```
$ cc -c f1.c               # 產生 f1.o
$ cc -c f2.c               # 產生 f2.o
$ ar rv mlib.a f1.o f2.o   # 將目標檔 f1.o 及 f2.o 插入到函式庫 mlib.a

# 選項 r ：在函式庫中插入目標檔。當插入的目標檔名已在庫中存在，則替換。
# 選項 v ：顯示執行操作選項的附加信息。
```

  2. nm：查看目標檔庫檔案 mlib.a 的內容。

```
$ nm mlib.a
```

  3. make 內建函式庫管理法則，例如：將 .c 檔案變成 .a 函式庫

```
.c.a:
   $(CC) -c $(CFLAGS) $<
   $(AR) $(ARFLAGS) $@ $*.o

# 變數 $(AR) 和 $(ARFLAGS)，一般分別為命令 ar 和選項 rv。
```

  4. 若有一函式庫 fud，包含 bas.o 檔案，則上例法則中的變數為
     1. $< 代表目前的相依性項目，就是 bas.c。
     2. $@ 代表目前的目標項目，就是 fud.a 函式庫。
     3. $* 代表目前的相依性項目，不過不含副檔名，就是 bas。

- 實作－管理一個函式庫
  1. 將 2.o 和 3.o 插入 mylib.a 函式庫中。新的 Makefile5 如下：

```
all: myapp
# Which compiler
CC = gcc
# Where to install
INSTDIR = /usr/local/bin
# Where are include files kept
INCLUDE = .
# Options for development
CFLAGS = -g -Wall -ansi
# Options for release
# CFLAGS = -O -Wall -ansi
# Local Libraries
MYLIB = mylib.a
myapp: main.o $(MYLIB)
   $(CC) -o myapp main.o $(MYLIB)
$(MYLIB): $(MYLIB)(2.o) $(MYLIB)(3.o)
main.o: main.c a.h
2.o: 2.c a.h b.h
3.o: 3.c b.h c.h
clean:
   -rm main.o 2.o 3.o $(MYLIB)
install: myapp
   @if [ -d $(INSTDIR) ]; \
    then \
      cp myapp $(INSTDIR);\
      chmod a+x $(INSTDIR)/myapp;\
      chmod og-w $(INSTDIR)/myapp;\
      echo "Installed in $(INSTDIR)";\
   else \
      echo "Sorry, $(INSTDIR) does not exist";\
   fi
```

  2. 執行

```
$ rm -f myapp *.o mylib.a
$ make -f Makefile5
gcc -g -Wall -ansi   -c -o main.o main.c
gcc -g -Wall -ansi   -c -o 2.o 2.c
ar rv mylib.a 2.o
a - 2.o
gcc -g -Wall -ansi   -c -o 3.o 3.c
ar rv mylib.a 3.o
a - 3.o
```

```
gcc -o myapp main.o mylib.a
```

3. 試驗 3.o 的相依性項目

```
$ touch c.h
$ make -f Makefile5
gcc -g -Wall -ansi   -c -o 3.o 3.c
ar rv mylib.a 3.o
r - 3.o
gcc -o myapp main.o mylib.a
$
```

- 以子目錄來管理函式庫
  1. 方法一：由主要的 makefile 呼叫子目錄的 makefile。

```
mylib.a:
    (cd mylibdirectory;$(MAKE))
```

     1. 要先製作 mylib.a。
     2. 當 make 依據這個法則建立函式庫時，它會進入子目錄 mylibdirectory；
     3. 隨後再喚起 make 命令來管理函式庫。這會喚起一個新 shell。
     4. 括弧是為了確保，所有的處理動作都是在一個 shell 中。
  2. 方法二：在 makefile 中使用變數，目錄加上 D，檔案加上 F。

```
.c.o:
      $(CC) $(CFLAGS) -c $(@D)/$(<F) -o $(@D)/$(@F)
# 在子目錄中編譯檔案，並將編譯完的目的檔留在子目錄中。
```

     1. 若目標項目所在目錄為 mydir，程式 2.c，目標項目 2.o 檔案，則上例法則中的變數為
        1. `$(@D)` 代表目前的目標項目所在目錄，就是 mydir。
        2. `$(<F)` 代表目前的相依性項目的檔案名稱，就是 2.c。
        3. `$(@F)` 代表目前的目標項目的檔案名稱，就是 2.o。
     2. 隨後利用相依性項目，更新現在目錄的函式庫，法則如下：

```
mylib.a: mydir/2.o mydir/3.o
     ar rv mylib.a $?
# $? 代表需要重建的相依性項目，就是 mydir/2.o 及(或) mydir/3.o
```

參考資料：

http://www.csie.cyut.edu.tw/~dywang/linuxProgram/node40.html