# 教育部顧問式嵌入式軟體聯盟
# 實驗模組整合與單一平台建置計畫

實驗模組名稱:熟悉 arm 嵌入式系統平台與發展環境

開發教師: 黃連進(micro@mail.tku.edu.tw)

**開發學生:** 呂世申

**修訂學生：**黃耀德，李杰欣

**學校系所:** 淡江大學資訊工程學系

**聯絡電話:** 02-26215656 ext 2741

**聯絡地址:** 淡水鎮英專路 151 號淡江大學
資訊工程學系

**繳交日期:** 2006 年 7 月 30 日

**修訂日期:** 2011 年 6 月

**實驗平台:** Real6410

**實驗主軸:** 嵌入式系統

**實驗內容關鍵字:** Cross Compiler, Real6410

**建檔編號:**

# 實驗目的

教導學生如何於 PC Linux 建立嵌入式系統之發展環境，包括

1. 下載 cross compiler(toolchain)，
2. 安裝 cross compiler，
3. 使用 cross compiler 產生執行檔，
4. 利用 sftp 上傳執行檔到目標板，
5. 利用 ssh 連線到目標板進行程式測試。

由於整個過程是透過網路進行，學生的學習場地可以不限於教室，也可以隨時選擇適合的時間學習。

# 實驗器材

- PC，個人電腦 1 台
  - Requirement: PentiumIII 以上，128MB，安裝 Linux RedHat 或 Fedora Core 或 Ubuntu。
  - Purpose: 用於執行 cross compiler，產生目標板之機器碼。

- 華亨: Real6410 一塊
  - Spec: ARM11(S3C6410, 800MHz), 1GB FLASH, 256MB DRAM (詳細規格參閱 *Real6410 簡介*)。
  - Price: NT$23,000
  - 代理商: 華亨科技股份有限公司

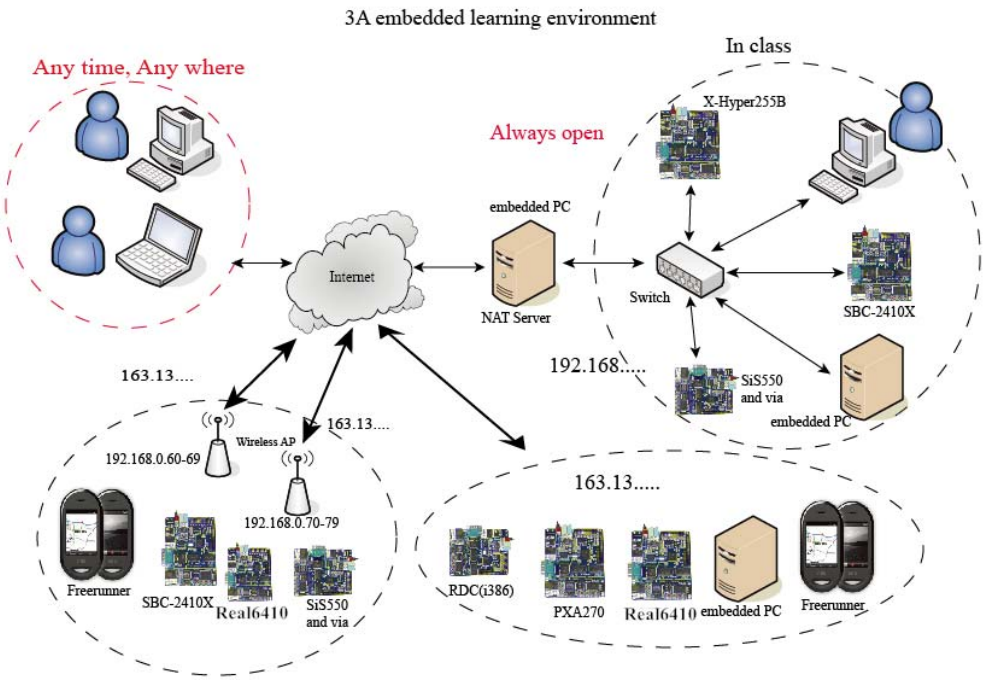
# 實驗所需軟體

- PC，個人電腦
  - 安裝 Linux RedHat，Fedora Core 或 Ubuntu。
  - arm Cross Compiler(GCC 4.1.2 + glibc 2.6)

- 華亨: Real6410 ARM11 實驗板
  - ssh, sftp

# Table of Contents





# 1. Real6410 簡介

## 1.1 硬體特性

| |
|---|
| 1. CVBS 視訊輸出 |
| 2. TV 輸出 |
| 3. USB OTG |
| 4. USB HOST |
| 5. 100M 網卡 |
| 6. RS232 輸出 |
| 7. SPI/IIC/IO/ADC |
| 8. 5V 電源輸出 |
| 9. SD Card 接口 |
| 10. SIM300 GPRS 介面 |
| 11. 電源開關 |
| 12. RESET |
| 13. Camera 介面 |
| 14. 耳機輸出 |
| 15. 1.5W 喇叭 |
| 16. 手機按鍵設計 |
| 17. U-Boot 配置設定 |
| 18. 24 bit RGB LCD 接口 |

## 1.2 軟體功能(for Linux)

| 名稱 | 功能特性 | 說明 |
|---|---|---|
| BIOS | Bootloader(u-boot) | |
| | Xmodem | |
| | Flash update | |
| | Set kernel parameter | |
| | Set partition | |
| kernel | Linux kernel 2.6.28.6 | |
| | ROM/CRAM/EXT2/FAT32/NFS/ YAFFS/UBIFS/JFFS2 | |
| Device driver | System Interrupt and Timer Driver | |
| | Serial device driver | |
| | Flash/Block memory device driver | |
| | 10Base-T Ethernet device driver | |
| | RTC device driver | |
| | USB Host/Slave device driver | |
| | LEDs device driver | |
| | Buttons device driver | |
| | LCD/TFT device driver | 480x272 |
| | Frame buffer device driver | |
| Network Services | Openssl v0.9.7m | Secure Socket Layer |
| | ssh (dropbear 0.53.1) | ssh server |
| | sftp | ftp server |
| | vsftp 2.3.2 | ftp server |
| | Web server(thttpd+php 4.4.9) | Web Server |
| Basic commands | Busybox 1.18.4 | Common Unix utilities |
| Data base | Sqlite 7.3.17 | SQL DataBase server |

# 2. 下載 toolchain (cross compiler)

利用網頁瀏覽器，例如 Firefox，下載 ARM cross compiler (openmoko-gtk2-2011-06-07.tar.xz)到您已經安裝 Linux 的 PC 上面。這個檔案大約 68MB，所以請組長指派組員下載，再分傳給其他組員。檔案下載點：

http://163.13.128.179

課程資料下載/0000 嵌入式系統/OpenMoko/cross-compiler/openmoko-gtk2-2011-06-07.tar.xz

# 3. 安裝 toolchain -以 root 身份

從網站下載回來的 cross compiler 必須將其解壓縮到正確目錄，才可以使用。主要步驟如下：

1. 下載回來檔案通常放於「下載」這個資料夾內，利用以下命令切換到「下載」資料夾

   **cd ~/**下載

> xz 的壓縮率相較於其他的像是 bz2，gz 等等壓縮格式都來得高，但其壓縮時間會花費較多。
>
> 原檔 /usr/local/openmoko 整個資料夾 407.5MB
>
> 壓縮檔 大小
>
> Openmoko.tar.gz 134.3MB
>
> Openmoko.tar.bz2 116.0MB
>
> Openmoko.tar.xz 74.3MB
>
> 原檔越大，越能看出壓縮差別。

2. 接者用 sudo 執行以下命令，

   **sudo tar Jxvf** openmoko-gtk2-2011-06-07.tar.xz **-C /**
   (若是無法解壓縮，那麼可能是沒有安裝 **xz-utils**，輸入
   **sudo apt-get install xz-utils )**

3. 用 tar 解壓縮後，會產生一個名稱為「/usr/local/openmoko/arm」的子目錄，
4. 此目錄內 bin, include 與 lib 分別為 cross compiler、c 語言引入檔與程式庫之所在。

操作過程如下：

```
[test@localhost ] cd ~/下載

[test@localhost 下載]# sudo tar Jxvf openmoko-gtk2-2011-06-07.tar.xz -C /

# 若不想要有以下過程顯示出來，可以把參數 v 拿掉即可。
usr/
usr/local/
usr/local/openmoko/

                .

# 過程過多，為不佔版面，所以省略。
[test@localhost ]$
```

## 3.1 include

include/ 主要是放置一些可引入的標頭檔；提供一些常數的定義，系統的宣告和函式庫函數宣告等等。

## 3.2 bin

這個目錄主要放置 cross compiler 的執行檔與相關檔案，這些程式必須於你的 Linux 主機(RedHat, Fedora Core 或 Ubuntu)上面執行。以下幾個比較常用程式的功能：

arm-linux-gcc -> C 編譯程式。
arm-linux-ld -> 連結程式。
arm-linux-g++ -> C++編譯程式。
arm-linux-ar -> 檔案收藏程式 archive。
arm-linux-strip -> 刪除執行檔內的符號。
arm-linux-ranlib ->收藏檔 archive 之索引產生程式。

## 3.3 lib

lib 主要是放置一些常用的「程式庫」，包括「動態或共享(share)」與「靜態」函數庫兩種。這些程式必須放於 arm 上面才能執行；因為他們皆為 arm 的機器碼。有分為四種：

1. **libLABRARY_NAME-GLIBC_VERSION.so**：
   這是主要的共享函式庫。
   範例：libcrypt-0.9.28.so
2. **libLABRARY_NAME.so.MAJOR_REVISION_VERSION**
   這是一個邏輯連結 ，連到actual shared libraries，差別在於只有主要版本號碼。
   範例：libcrypto.so.0.9.7
3. **libLIBRARY_NAME.so**
   這通常是一個邏輯連結，連到major revision version，差別在於只有 **LIBRAY_NAME**，而沒有版本編號。當主要共享函式庫更新時，可以不用更新此一連結。
   範例：libcrypto.so
4. **libLIBRARY_NAME.a**
   靜態或稱為傳統函式庫。
   範例：libcrypto.a

# 4. 使用 cross compiler 產生目標板的機器碼

用 vi 寫一段小程式練習，範例如下：

```
[TestID@localhost ~]$ vi hello.c
#以下是 hello.c 的內容
#include <stdio.h>
int main(void)
{
    printf("This is my first program!\n");
    return 0;
}

[TestID@localhost ~]$ gcc -o hello hello.c

[TestID@localhost ~]$ ls -ln
total 12
-rwxrwxr-x 1 500 500 4711 Sep 12 20:21 hello
-rw-rw-r-- 1 500 500   91 Sep 12 20:21 hello.c

[TestID@localhost ~]$ file hello
hello: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.6.9,
dynamically linked (uses shared libs), for GNU/Linux 2.6.9, not stripped

[TestID@localhost ~]$ ./hello
This is my first program!
[who@s195 work]$

#接著用” /usr/local/openmoko/arm/bin/arm-linux-gcc”來編譯，產生執行檔

/usr/local/openmoko/arm/bin/arm-linux-gcc -o arm_hello hello.c
[TestID@localhost ~]$ ls -ln
total 20
-rwxrwxr-x 1 500 500 5324 Sep 12 20:29 arm_hello
-rwxrwxr-x 1 500 500 4711 Sep 12 20:21 hello
-rw-rw-r-- 1 500 500  91 Sep 12 20:21 hello.c

# arm_hello 產生後，我們用”file”檢查看看是否為 arm 的執行檔
[TestID@localhost ~]$ file arm_hello
arm_hello: ELF 32-bit LSB executable, ARM, version 1 (ARM), dynamically linked (uses
shared libs), not stripped
```

```
#由於以後會常常用到 arm-linux-gcc 這個命令，所以可以把 cross compiler 的 bin/路徑加到 PATH 中
#將 PATH=$PATH:/usr/local/openmoko/arm/bin/ 加到 ~/.bashrc, ~/.bash_profile, 或 /etc/profile
#這三個選一個就可以了。若是加到 ~/.bashrc 與 ~/.bash_profile，則只有自己的 PATH
#會有此路徑；若加在 /etc/profile，則所有 users 皆會有此路徑。
#推薦加在 ~/.bashrc 或 ~/.bash_profile 就可以，因為並不是大家都需要用到這個路徑，
#如此，可以節省系統資源
```

# 5. 用 sftp 上傳執行檔到目標板

```
[TestID@localhost ~]$ sftp TestID@163.13.128.171

#使用 secure ftp 上傳到 163.13.128.171 上
#可直接在 IP 前加登入帳號，否則預設是用當時環境的 user ID

Connecting to 163.13.128.171...
The authenticity of host '163.13.128.171 (163.13.128.171)' can't be established.
RSA key fingerprint is b5:bd:99:7a:55:6f:f6:81:39:0f:2e:70:41:97:d9:3b.
Are you sure you want to continue connecting (yes/no)? yes

#若第一次連線，會出現要建立 RSA key fingerprint 的確認。
#請輸入 yes

Warning: Permanently added '163.13.128.171' (RSA) to the list of known hosts.
TestID@163.13.128.171's password:

#輸入密碼後，利用 put 這個命令將 hello 檔案上傳到遠端平台。其他常用的 sftp 命令可以用 man sftp 查閱喔
#最後，用 bye 命令結束這次的 sftp 連線

sftp> put hello
Uploading hello to /home/TestID/hello
hello                  100% 4711    5.0KB/s   00:01
sftp> bye
[TestID@localhost ~]$
```

# 6. 用 ssh 連線到目標板進行測試

```
[TestID@localhost ~]$ ssh TestID@163.13.128.171(這裡的 ID 與 IP 都是測試用,非正式 ID,IP)
```

#ssh 的使用方法跟 sftp 一樣，所以我們一樣先加上所要登入的 ID 在 IP 前。

```
The authenticity of host '163.13.128.181 (163.13.128.171)' can't be established.
RSA key fingerprint is b5:bd:99:7a:55:6f:f6:81:39:0f:2e:70:41:97:d9:3b.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '163.13.128.171' (RSA) to the list of known hosts.
TestID@163.13.128.171's password:
==========================================================

sysinfo: 2011-04-26

   anonymous ftp server : http://163.13.128.179

==========================================================

1. OpenMoko/Real6410 info:
   cross-compiler --
OpenMoko/cross-compiler/openmoko-gtk2-2009-05-04.tar.bz2(115MB)

2. sbc2410 info:
   cross-compiler -- arm/cross-compiler/arm-3.4.6-xorg-2010-05-04.tar.bz2(96MB)
   cross-compiler -- penMoko/cross-compiler/openmoko-gtk2-2009-05-04.tar.bz2(115MB)
   DataSheet -- arm/sbc2410/BSP/SBC2410_LINUX.pdf

3. qt2440 info:
   cross-compiler -- arm/cross-compiler/arm-3.4.6-xorg-2010-05-04.tar.bz2(96MB)
   DataSheet -- arm/qt2440/BSP/datasheet

==========================================================

BusyBox v1.11.2 (2008-08-31 11:43:46 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.
BusyBox v1.11.2 (2008-08-31 11:43:46 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.
```

#以上是登入後所顯示的告示

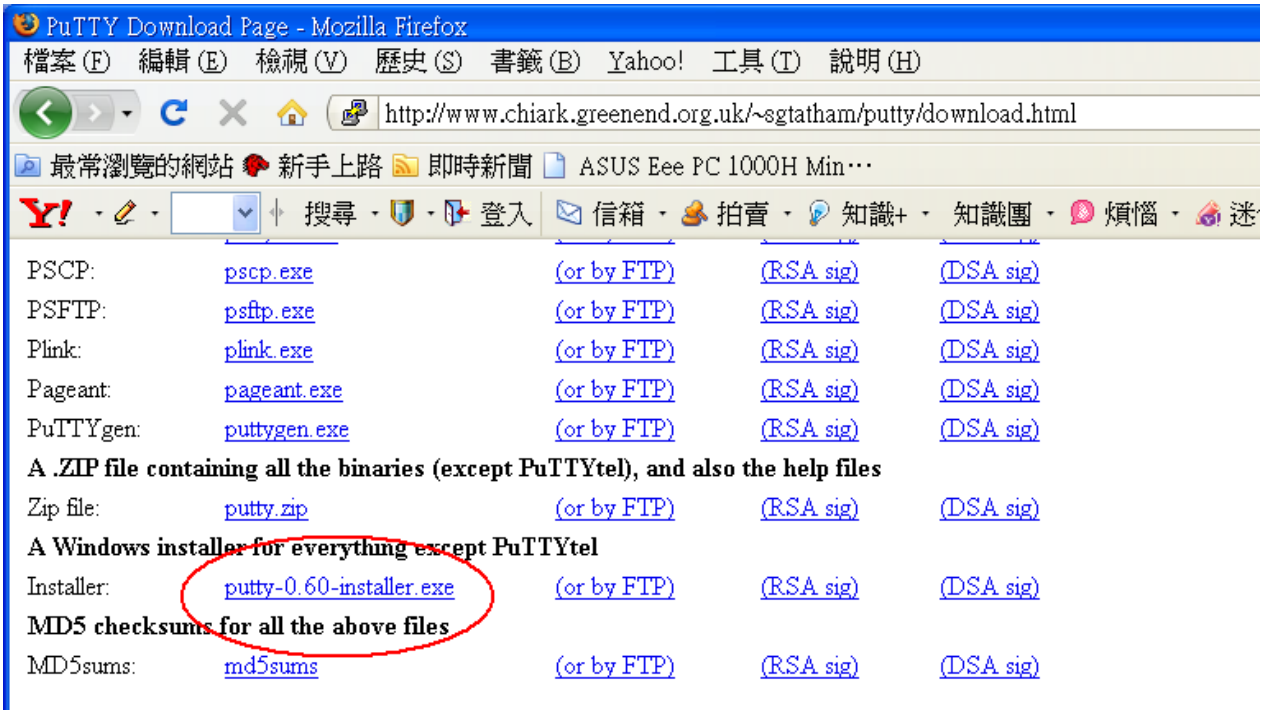
```
#接著，我們就可以利用 Busybox 所提共的命令，操作遠端的機器
#最後，利用 exit 結束本次的連線

[TestID @Real6410-171 TestID]$ ls
hello
[TestID @ Real6410-171 TestID]$ ls -l
drwxr-sr-x    2 TestID    TestID    1024 Jul 24 00:45 .
drwxr-xr-x    8 TestID    TestID    1024 Jul 21 11:05 ..
-rwx------    1 TestID    TestID    4711 Jul 24 00:45 hello
[TestID @ Real6410-171 TestID]$ chmod +x hello

#由於剛剛上傳的 hello 這個程式還沒有執行的權限，所以我們用 chmod +x 讓它成為可以執行。

[TestID @ Real6410-171 TestID]$ ./hello
This is my first program!
[TestID @ Real6410-171 TestID]$ exit
```

# 5. 用 Putty 連線到目標板進行測試

除了於 linux 底下用 ssh 連線到目標板以外，我們也可以於 Windows 作業系統底下利用 putty 進行連線。

## 7.1 下載

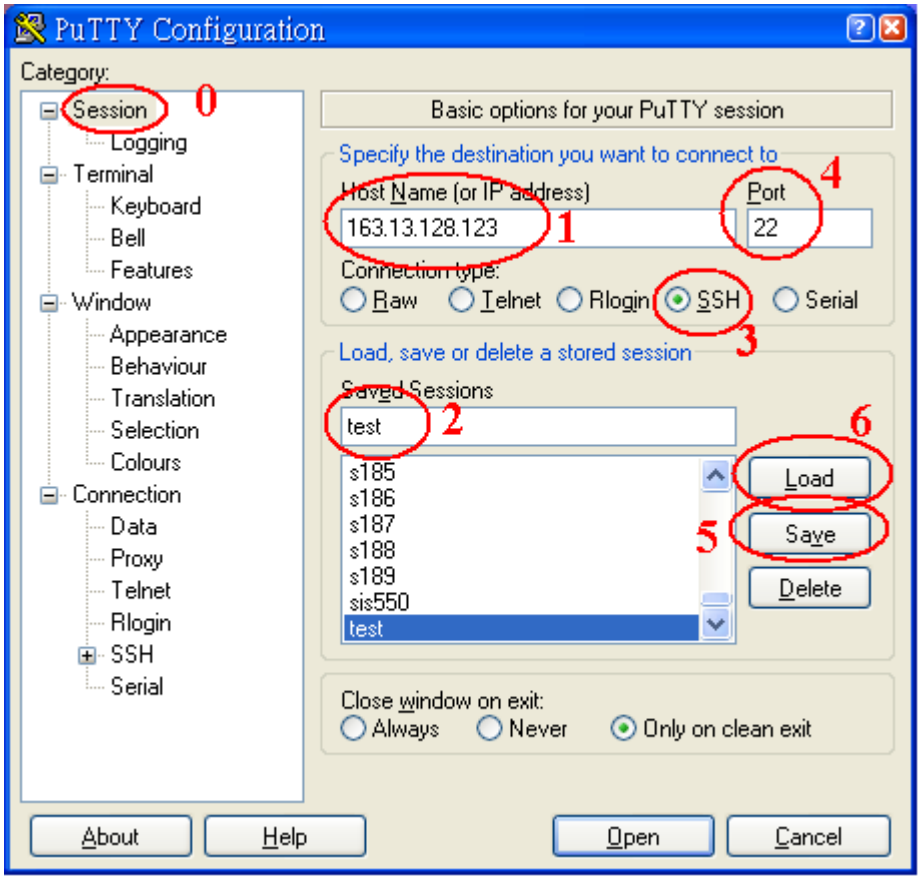
首先，請到這個網站 http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html 將 putty-0.60-install.exe 這隻程式下載回來(2008/9/23 有效)，並且進行安裝。



## 7.2 設定

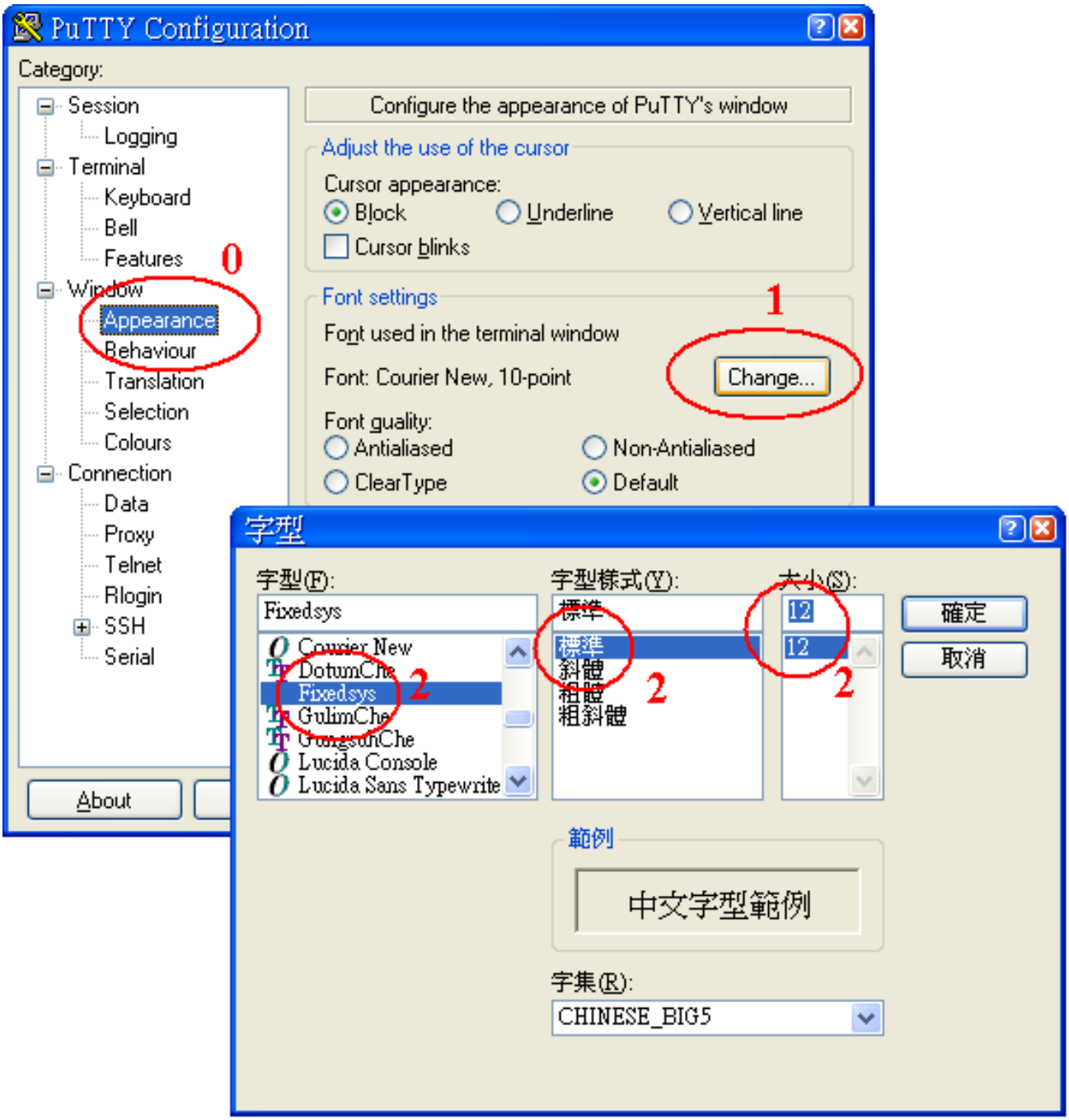
第一次執行 putty 時，最好進行網路連線與個人喜好設定。網路連線的設定如下：

0. 選取 Session 這個類別
1. 在「Host Name」這裡輸入遠端主機的 IP，例如 163.13.128.123
2. 在「Saved Sessions」這裡輸入連線的名稱，例如 test
3. 將連線方式「Connection type」設為「SSH」
4. 「SSH」連線「port」通常為 22，但可能為其他數值
5. 此時可以按「Save」這個按鈕，儲存以上的設定值
6. 或是按「Load」這個按鈕，載入其他的設定值，進行修改。



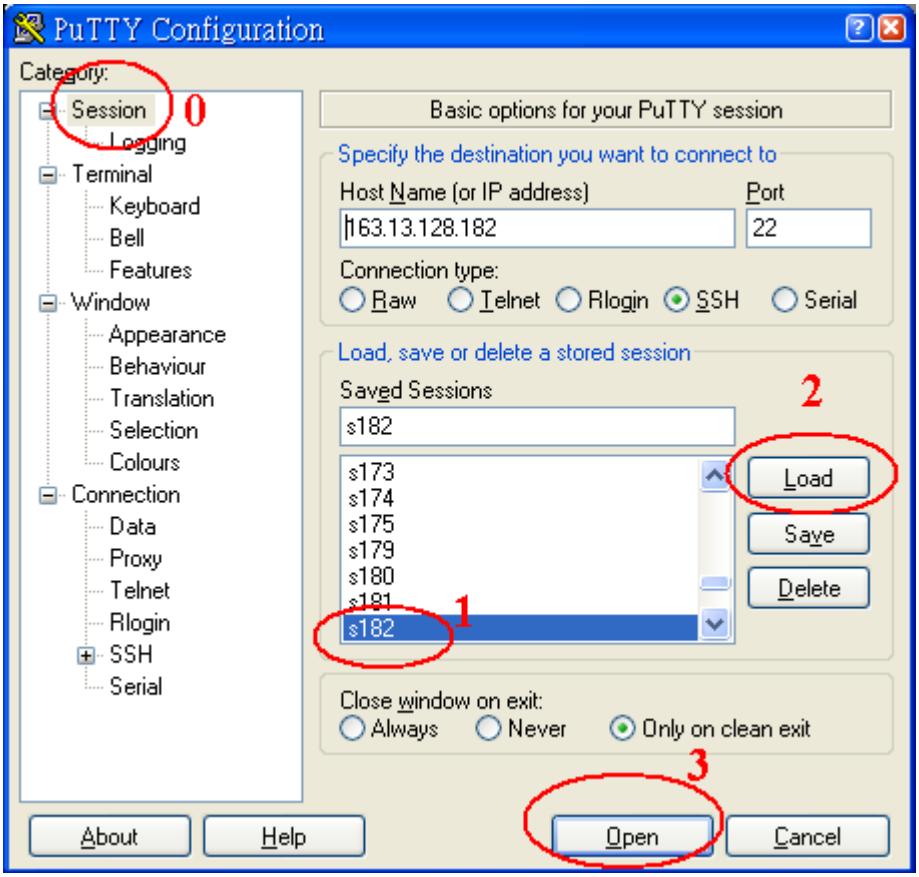
接者，通常會進行如外觀「Appearance」等設定，讓 putty 用個人喜好的字體顯示，步驟如下：

0. 選取「Appearance」這個類別
1. 按「Font Setting」內的「Change…」這個按鈕，此時會顯示另一個「字形」對話盒
2. 選取自己喜歡的字形，樣式與大小，然後按「確定」
3. 最後利用「Session」這個類別的「Save」按鈕儲存起來，



## 7.3 啟動

當我們按照以上的步驟進行安裝與設定，以後啟動 putty 時，只要選取「Session」載入設定「Load」，就可以開啟「Open」連線到遠端目標。



# 6. 用 Makefile 產生執行檔

我們用 makefile 來產生執行檔，make 與 makefile 為一體的東西，在此以範例說明如何「1 支程式由 2 支獨立的原始碼所組成」且用 make 與 makefile 去管理。

```
#流程：
#首先我們先寫出兩隻陽春的程式 main.c && hello_fun.c
#再寫 makefile，不一定檔名要 makefile 其它檔名也可以，只不過 make 預設是
#指定 makefile，之後就是在 cammad line 打”make”即可。
#
[TestID@localhost ~]$ vi main.c
//main.c 內容
#include <stdio.h>
int main(void)
{
    puts("This is my first program!\n");
    hello_fun();
    return 0;
}
[TestID@localhost ~]$ vi hello_fun.c
//hello_fun.c 內容
#include <stdio.h>
void hello_fun()
{
   puts("Hello world !!\n");
}
[TestID@localhost ~]$ vi makefile
#makefile 內容開始
#程式中的 CROSS, CC, STRIP, CFLAG, OBJS 皆是我們的 macro variable
#便於修改參數
CROSS=/usr/local/openmoko/arm/bin/arm-linux-
CC=$(CROSS)gcc
STRIP=$(CROSS)strip
CFLAGS=-c –Wall

# all 這 target 後面接的相依項目，就是要產生的目標檔
all: hello

OBJS = main.o hello_fun.o
hello: $(OBJS)
    $(CC) -o $@ $(OBJS)     ### 非常重要：這裡要 Tab 按鍵跳位，不可以用 Space
```

```
    $(STRIP) $@          ### 非常重要：這裡要 Tab 按鍵跳位，不可以用 Space

main.o: main.c
    $(CC) $(CFLAGS) $<    ### 非常重要：這裡要 Tab 按鍵跳位，不可以用 Space

hello_fun.o: hello_fun.c
    $(CC) $(CFLAGS) $<     ### 非常重要：這裡要 Tab 按鍵跳位，不可以用 Space
clean:
    rm -f hello *.o        ### 非常重要：這裡要 Tab 按鍵跳位，不可以用 Space
# 補充：
# $? ：代表需要重建的相依性項目。
# $@：目前的目標項目名稱。
# $< ：代表目前的相依性項目。
# $* ：代表目前的相依性項目, 不過不含副檔名。
```

```
[TestID@localhost ~]$ make
#亦可以加參數"-f"指定 makefile
/usr/local/openmoko/arm/bin/arm-linux-gcc -c -Wall main.c
main.c: In function `main':
main.c:6: warning: implicit declaration of function `hello_fun'
/usr/local/openmoko/arm/bin/arm-linux-gcc -c -Wall hello_fun.c
/usr/local/openmoko/arm/bin/arm-linux-gcc -o hello main.o hello_fun.o
/usr/local/openmoko/arm/bin/arm-linux-strip hello
[TestID@localhost ~]$ ls
hello  hello_fun.c  hello_fun.o  main.c  main.o  makefile

#hello 產生後，我們一樣用"file"檢查看看
[TestID@localhost ~]$ file hello
hello: ELF 32-bit LSB executable, ARM, version 1 (ARM), dynamically linked (uses shared
libs), stripped

#此時就可以參考前面所提及的步驟，將 hello 這隻程式上傳到所分配的 arm 實驗板，
#進行測試。測試完成後，記得將 hello 這個檔案用 rm 命令清除，節省 arm 實驗板的空間。
#
[TestID@localhost ~]$ make clean
rm -f hello *.o
# 最後，我們用 clean 把不要的東西給清除
[TestID@localhost t]$ ls
hello_fun.c  main.c  makefile
[TestID@localhost ~]$
```

# 7. 認識 linux 函數庫

## 9.1 前言

函數庫(libraries)是一組事先編譯好的，並且可再利用的函數。linux 作業系統上有許多的函數庫，如 glibc，libpthread 等。標準的系統函數庫通常放在/lib 或/usr/lib 這兩個目錄內。函數庫的檔案名稱都是以 lib 開頭，中間接代表該函數庫的名稱，後接副檔名。副檔名為.a 代表靜態函數庫，.so 代表共享函數庫(share library)或動態函數庫(dynamic library)。以下將介紹靜態函數庫(static library)跟動態函數庫(dynamic library)的製作與使用。

最簡單的靜態函數庫就是收集目的碼檔案。當一個程式需要使用函數庫中的函數時，要先引用標頭檔。在這些標頭檔中，會宣告該函數的原型。編譯器跟連結器會把程式碼跟函數庫整合成一個可執行檔。

靜態函數庫的一個缺點是，當我們同一時間執行許多程式，而這些程式使用到函數庫中的相同函數。這樣會造成函數庫的程式碼重複佔用記憶體空間。另外，在每個執行檔中，也複製著相同的函數庫程式碼。無形中造成了記憶體與磁碟空間的浪費，所以就出現了共享函數庫來解決這個問題。

當一個程式引用動態函數庫的函數時，在連結過的程式碼裡，並沒有所呼叫的函數的程式碼，而是在程式載入記憶體執行時，才去解決共享函數庫的函數與呼叫問題。所以，系統只準備一份函數庫的程式碼，一次提供給許多程式，並且只佔用一份空間。

## 9.2 靜態函數庫之製作與使用

```
#流程：
#首先我們先寫出三支陽春的程式 fred.c、bill.c 和 program.c
#先指定編譯 program.c，產生目的檔
#接下來要利用 ar 來產生靜態函數庫
#最後編譯時，要使用-L 跟-l 告知編譯器函數庫的位置
[TestID@localhost ~]$ vi fred.c
#include<stdio.h>
void fred(int arg)
{
printf("fred: we passed %d\n",arg);
}
[TestID@localhost ~]$ vi bill.c
#include <stdio.h>
void bill(char *arg)
{
```

```
  printf("bill:we passed %s\n",arg);
}

[TestID@localhost ~]$ vi program.c
#include "lib.h"
int main()
{
  bill("Hello world");
  fred(3);
  return 0;
}
[TestID@localhost ~]$ vi lib.h
void bill(char *);
void fred(int);

[TestID@localhost ~]$ /usr/local/openmoko/arm/bin/arm-linux-gcc \
      -c fred.c bill.c
#把 fred.c 和 bill.c 編譯成目的檔
[TestID@localhost ~]$ /usr/local/ openmoko/arm/bin/arm-linux-ar \
        crv libfoo.a bill.o fred.o
#利用 ar 來產生靜態函數庫
a - bill.o
a - fred.o
[TestID@localhost ~]$ /usr/local/ openmoko/arm/bin/arm-linux-gcc \
       -o program program.c -L. –lfoo

#  -L 是告訴編譯器函數庫的位置(.代表目前目錄）
#  -l 是告訴編譯器使用 libfoo.a 函數庫
[TestID@localhost ~]$ sftp TestID@163.13.128.171
Connecting to 163.13.128.171...

TestID@163.13.128.171's password:
sftp>put program
sftp>bye
[TestID@localhost ~]$ ssh TestID@163.13.128.171
TestID@163.13.128.171's password:
TestID@Real6410-171 TestID]$ ./a.out
bill:we passed Hello world
fred: we passed 3
```

## 9.3 動態函數庫之製作與使用

```
#這次使用的程式碼同上
#在安裝程式庫時，你需把以 real name 命名的程式庫放在適當的目錄
#(/usr/lib 或/usr/local/lib)，然後建立兩條分別以
# soname 及 linker name 命名的 symbolic link 指向 real name 程式庫檔案

    arm-linux-gcc -fPIC -Wall -c fred.c bill.c

#以-fPIC 選項建立一 object file，PIC(position-independent code)，是
#shared library 必需使用的選項，這令 shared library 擁有自己的動態記憶
#體區塊(i.e. 它使用全域記憶體偏移量表(global offset table，GOT)，不受
#主程式的記憶體限制)

arm-linux-gcc -shared -Wl,-soname,libfoo.so.1 \
    -o libfoo.so.1.0.1 fred.o bill.o  -lc

# 這指令會產生 fred.o 跟 bill.o，然後再需要以「-W1,-soname」連結選項將
# 目的檔編譯成「.so」。會建立一個 realname(真正擁有已編譯的程式碼)的檔案。
# 其檔名包括 lib,函數庫名稱,”.so”,主版本號碼跟發佈版本號碼。
# 如 libxxx.so.N1.N2.N3

    ln -s libfoo.so.1.0.1 libfoo.so.1

#建立 soname 的 symbolic link。soname 以 lib 開頭，加上函數庫名稱，在加
#上”.so”跟”.”及版本號碼。如”libxxx.so.N”。

ln -s libfoo.so.1.0.1 libfoo.so

#建立 linker 的 symbolic link。它是編譯器所搜尋的函數庫名稱，
# 就是 real name 刪去所有版本號碼的名稱。

arm-linux-gcc  program.c –L. -lfoo

#告訴編譯器，連結目前資料夾底下的 libfoo 函數庫
#並且執行檔的名稱為預設的 a.out

tar jcvf lib.tar.bz2 libfoo.so.1.0.1 libfoo.so.1 libfoo.so
#利用 tar 程式把 libfoo.so.1.0.1，libfoo.so.1，libfoo.so 打包起來，
#要把動態函數庫移到嵌入式系統平台上
```

```
[TestID@localhost ~]$ sftp TestID@163.13.128.171
Connecting to 163.13.128.171...
TestID@163.13.128.171's password:
#把 lib.tar.bz2 搬到/lib 底下，編譯的程式放到家目錄下
sftp>put a.out       #把 a.out 執行檔傳到使用者的家目錄
sftp>
sftp>put lib.tar.bz2
sftp>bye
[TestID@localhost ~]$ ssh TestID@163.13.128.171
TestID@163.13.128.171's password:
[TestID@Real6410-171 TestID]$

==============================================
注意：將程式庫安裝於 lib 這個目錄，需要有 root 權限，
      如有此需求，請向平台的管理員申請。如此才可以進行
      以下操作。
==============================================

[root@Real6410-171 TestID]# tar -jxvf lib.tar.bz2 –C /lib
#解開 lib.tar.bz2
[TestID@Real6410-171 ~]# cd ~     #切回使用者的家目錄
[TestID@Real6410-171 ~]# ./a.out   #執行 a.out
bill:we passed Hello world
fred: we passed 3
[TestID@Real6410-171 ~]# ls -l a.out program

#以下為相同的程式碼，連結靜態函數庫與利用動態函數庫時，
#檔案大小的差異。
#a.out 是連結動態函數庫，program 連結靜態函數庫，a.out 的大小為
#6272 位元組，program 大小為 6493 位元組。因此動態函數庫可縮小可執行檔
#的體積。
-rwx------    1 0       1006        6272 Jul 16 19:45 a.out
-rwx------    1 0       1006        6493 Jul 16 19:45 program
```

# Reference


1. vi http://people.cis.ksu.edu/~bhoward/vi/index.html
   http://thomer.com/vi/vi.html
   http://www.digilife.be/quickreferences/indexe.html
2. Gcc http://gcc.gnu.org
3. ssh/sftp http://www.openssh.com/
4. LinuxBasics.org http://linuxbasics.org/LDP/LDP/intro-linux/intro-linux.pdf
5. Linux Documentation Project http://www.tldp.org/
6. FreeBSD Handbook http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/index.html
7. Putty 下載 http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
8. Makefile 撰寫 http://www.study-area.org/cyril/opentools/opentools/makefile.html
   Tutorial - Makefile http://www.opussoftware.com/tutorial/TutMakefile.htm
   如何寫一個簡單的 Makefile http://www.linux.org.tw/CLDP/OLD/doc/makefile-ch1.html
9. Debian 參考手冊
   http://www.debian.org/doc/manuals/reference/index.zh-tw.html#contents
10. GCC 使用手冊 http://gcc.gnu.org/onlinedocs/
11. GNU C library - Glibc 函數庫參考手冊 http://www.gnu.org/software/libc/manual/
12. Busybox 使用手冊 http://www.busybox.net/downloads/BusyBox.html