Glibc 2 HOWTO 中文版

原作 Eric Green, ejg3@cornell.edu
譯者 Allex Juang, allex@ms2.accmail.com

1.5 版, 1998年 2月 8日

這份 glibc 2 HOWTO 內容包括在 Linux 系統上安裝與使用第二版 GNU C Library。

目錄

1 簡介

1.1 關於 glibc 2

Glibc 2 是最新版的 GNU C 函式庫。 它目前不需修改就可以在 GNU Hurd，Linux i386， m68k， 以及 alpha 系統上執行， 而 2.1 版中將加入對 Linux PowerPC， MIPS， Sparc， Sparc 64 與 Arm 等系統的支援。未來將加入對其它架構及作業系統的支援。

在 Linux 上，glibc 2 是主要版本爲 6 的 libc，也就是 Linux libc 5 的後繼者。 Linux libc 的發展人員想以此逐漸取代 libc 5。 就 2.0.6 來看，glibc 已具有成熟產品的品質。 Glibc 2.1 版 (在不久的將來) 將加入更多的功能， 並移植到更多系統上，足夠當作主流使用。

Glibc 2 有三個選擇性的附加套件：

Crypt

UFC-crypt 加密套件。 由於出口限制所以要分離出來。

LinuxThreads

Posix 1003.1c "pthread" 介面的實作。

Locale data

包含製造使用 glibc 的 internationalization 功能所需 locale 資料檔案的資料。

強烈建議安裝 Cryt 與 LinuxThreads 這兩個附加套件， 如果不安裝的話，有可能會與其它系統上的函式庫造成不相容的情形。 (假如你不想使用這兩個附加套件的話， 那你必須在執行 configure 時加上 –disable-sanity-checks 的選項)

1.2 關於這份文件

這份文件涵蓋如何在 Linux 系統上安裝 glib 2， 適用對象是那些在 intel-based 機器上使用 libc 5 的使用者。 但是對於其它系統以及其它不同函式庫 (如 glibc 1) 的使用者， 只要將這份文件中出現的檔名以及系統架構名稱作適當的代換， 應該也可得到些幫助。

這份 HOWTO 的最新版本的拷貝可在 *Linux Documentation Project* `<http://sunsite.unc.edu/LDP>` 或是 `<http://www.imaxx.net/~thrytis/glibc/Glibc2-HOWTO.html>` 找到。

1.3 這份文件的最新改變

1.5 與 1.4 版的差別：

- 由 Ed Bailey 加入索引
- 我的新電子郵件信箱位址

1.4 與 1.3 版的差別：

- 將目前狀態由實驗性改成成熟產品。
- 更新發展中的移植列表。
- 更新最新版到 2.0.6。

2 選擇你的安裝方式

有許多種方式安裝 glibc，你可以安裝成一個測試用的函式庫， 將目前使用的函式庫當作是內定的，然後在編譯程式的時候用不同的選項來試用新的函式庫。 用這種安裝方式以後可以輕易的移除 glibc 。 (雖然用 glibc 連結的程式在移除 glibc 後都將不能使用) 另外，用這種方式安裝，你必須由原始碼重新編譯， 並沒有 binary distribution 可以安裝成測試的函式庫。 這樣的安裝請參照4 (安裝成測試用的函式庫)。

這份文件中所說的另一種方式則是將 glibc 安裝成主要函式庫。 所有新編譯的程式都將會使用 glibc，但是你也可以在編譯時用不同的選項而將程式連結到舊的函式庫。 你可以直接安裝已經編譯好的版本

，也可以自己編譯函式庫。 假如想更改最佳化或是組態選項， 或是想加入已編譯好的版本中未支援的 add-on 的話， 那你必需取得原始碼並且自己編譯，這個安裝的過程請參照 5 (安裝成主要的 C 函式庫)

Frodo Looijaard 描述了另一種 glibc 的安裝方式。 他的方法是將 glibc 安裝成次要的函式庫， 然後設定交叉編譯器 (cross compiler) 來使用 glibc。 這個方式比起這份文件中的測試安裝要複雜的多，但是比較容易連結到 glibc。 這個方法在他的 *Installing glibc-2 on Linux* `<http://huizen.dds.nl/~frodol/glibc/>` 文件中說明。

假如你現在使用的是 Debian 1.3， 但是不想爲了 glibc 而升級到不穩定的版本， 那麼 *Debian libc5 to libc6 Mini-HOWTO* `<http://www.gate.net/~storm/FAQ/libc5-libc6-Mini-HOWTO.html>` 會告訴你如何使用 Debian 的 package 來升級。

假如你要把 glibc 2 安裝到一重要的系統上， 那也許你應使用測試的安裝，或是在備用系統上試試看。即使沒有臭蟲，某些程式在編譯之前仍然需要某些修改， 原因在於某些函式原型與型態上的改變。

3 取得函式庫

Glibc 2 包含 glibc package 以及數個附加套件： LinuxThreads、 Locale 與 Crypt， 原始碼在

- `<ftp://prep.ai.mit.edu/pub/gnu/glibc-2.0.6.tar.gz>`
- `<ftp://prep.ai.mit.edu/pub/gnu/glibc-linuxthreads-2.0.6.tar.gz>`
- `<ftp://prep.ai.mit.edu/pub/gnu/glibc-localedata-2.0.6.tar.gz>`
- `<ftp://prep.ai.mit.edu/pub/gnu/glibc-crypt-2.0.6.tar.gz>`

你需要大約 150MB 的磁碟空間來作完全的編譯與安裝， 核心函式庫的基本安裝大約要 50MB。

2.0.6 並沒有 binary 的 package， 2.0.4 的 binary package 只有 i386 及 m68k 的版本， 而 2.0.1 for alpha 則可以在以下地方找到：

- Intel x86:
    - `<ftp://prep.ai.mit.edu/pub/gnu/glibc-2.0.4.bin.i386.tar.gz>`
    - `<ftp://prep.ai.mit.edu/pub/gnu/glibc-crypt-2.0.4.bin.i386.tar.gz>`
- Alpha:
    - `<ftp://prep.ai.mit.edu/pub/gnu/glibc-2.0.1.bin.alpha-linux.tar.gz>`
    - `<ftp://prep.ai.mit.edu/pub/gnu/glibc-crypt-2.0.1.bin.alpha-linux.tar.gz>`
- m68k:
    - `<ftp://prep.ai.mit.edu/pub/gnu/glibc-2.0.4-m68k-linux.bin.tar.gz>`
    - `<ftp://prep.ai.mit.edu/pub/gnu/glibc-crypt-2.0.4-m68k-linux.bin.tar.gz>`

由於 cypt 這個附加套件有出口限制，非美國使用者請到 `<ftp://ftp.ifi.uio.no/pub/gnu>` 取得。

如果你用的是 RedHat distribution， 那你可以從 `<ftp://ftp.redhat.com/pub/redhat/>` 取得 glibc2 的 rpm。 在 RedHat distribution 5.0 中， glibc 2 是主要的 C 函式庫。

如果你用的是 Debian distribution，那你可以從 `<ftp://ftp.debian.org/debian/dists/unstable/main/>` 取得 glibc2 的 package。檔名是 libc6。 Glibc 2 目前是 Debian 測試版的基本 package， 等到 Debian 2.0 出來之後將會是主要的 C 函式庫。

4 安裝成測試用的函式庫

這一節的內容是關於如何將 glibc 2 安裝成測試用的函式庫。 所有編譯的程式都連結到你目前的函式庫， 除非你給了某些額外的參數才會連結到新的函式庫。 由於路徑被編譯到某些檔案中，你需要由原始碼來安裝這函式庫。

4.1 編譯與安裝

4.1.1 需求

- 約 150 MB 的磁碟空間
- GNU make 3.75
- gcc >= 2.7.2 (最好是 2.7.2.1)
- binutils 2.8.1 (假如是 alpha 你還需要 snapshot)
- bash 2.0
- autoconf 2.12 (假如你更動了 configure.in)
- texinfo 3.11

在一台安裝有 64MB 記憶體 i586@133 的機器上， 編譯完整的函式庫 (包括附加套件) 大約須要 3 小時。 而在 i686@200 則須約一個半小時。

4.1.2 解開原始碼

你必需由 archive 中取出原始碼。如此才能編譯它，最好的方式如下：

```
tar xzf glibc-2.0.6.tar.gz
cd glibc-2.0.6
tar xzf ../glibc-linuxthreads-2.0.6.tar.gz
tar xzf ../glibc-crypt-2.0.6.tar.gz
tar xzf ../glibc-localedata-2.0.6.tar.gz
```

這會將 linuxthreads、 crypt 與 localedata 等目錄放至 glibc-2.0.6 的目錄底下， 如此 configure 才可以找到這些 add-ons。

4.1.3 設定組態

在 glibc-2.0.6 的目錄底下，建立一個新的目錄 compile，並且 cd 到 compile 底下， 所有的工作都會在這個目錄中完成，這會簡化後續的清除工作。 (發展者似乎並未讓 'make clean' 作得很好)

```
mkdir compile
cd compile
```

執行 `../configure`。 要使用父加套件，你必需用 –enable-add-ons 指定， 例如說 –enable-add-ons=linuxthreads,crypt,localedata。

你也必需指定要安裝的目錄，/usr/i486-linuxglibc2 是個不錯的選擇， 這樣的 configure 命令如下：

```
../configure --enable-add-ons=linuxthreads,crypt,localedata --prefix=/usr/i486-linuxglibc2
```

4.1.4 編譯與安裝

要編譯與驗證，執行

```
make
make check
```

假如 'make check' 成功了，安裝函式庫：

```
make install
```

4.2 更新 dynamic loader

1. 建立連結， 從 `ld.so` 到 `/lib/ld-linux.so.2`：

   ```
   ln -s /usr/i486-linuxglibc2/lib/ld-linux.so.2 /lib/ld-linux.so.2
   ```

   當一個檔案連結時，這個函式庫是唯一一個位置固定的。 當穩定的版本出現的時候， 在 /lib 使用連結也會使得將 glibc 升級爲主要函式庫時更爲輕鬆。

2. 修改 `/etc/ld.so.conf`。 你需要增加新的函式庫所在的路徑到檔案的最後， 這路徑應該是 `<prefix>/lib`。 假如用以上的選擇就應該是 `/usr/i486-linuxglibc2/lib`。 在你修改了 `/etc/ld.so.conf` 之後，執行：

   ```
   ldconfig -v
   ```

4.3 設定 gcc

安裝的最後一步是更新 `/usr/lib/gcc-lib` 以使 gcc 知道如何使用新的函式庫。 首先，你需要複製目前的組態，要知道目前的組態，使用 gcc 的 -v 選項：

```
% gcc -v
Reading specs from /usr/lib/gcc-lib/i486-unknown-linux/2.7.2.2/specs
gcc version 2.7.2.2
```

這裡，i486-unknown-linux 是目前所用的系統，2.7.2.2 則是目前版本。 你需要將 `/usr/lib/gcc-lib/`<系統> 複製到新的測試系統目錄：

```
cd /usr/lib/gcc-lib/
cp -r i486-unknown-linux i486-linuxglibc2
```

到你的新系統目錄以及版本目錄下：

```
cd /usr/lib/gcc-lib/i486-linuxglibc2/2.7.2.2
```

修改目錄下的 `specs`。在這檔案中，將 `/lib/ld-linux.so.1` 換成 `/lib/ld-linux.so.2`，你也必需要移去檔案中所有的 `%{...:-lgmon}` 表示式，因爲 glibc 不使用 gmon 函式庫來做 profile。你可以在9 (Specs 檔範例)找到一個範例。

4.4 更新標頭檔連結

你需要在你的 include 目錄下建立連結到其它的 include 目錄：

```
cd /usr/i486-linuxglibc2/include
ln -s /usr/src/linux/include/linux
ln -s /usr/src/linux/include/asm
ln -s /usr/X11R6/include/X11
```

也許你有其它的函式庫，例如說 ncurse，而這些函式庫會需要它們自己的標頭檔，你應該從 `/usr/include` 複製或連結這些檔案。(有些函式庫需要重新用 glibc2 來 compile，在這種情形下，只要編譯並且安裝到 `/usr/i486-linuxglibc2`)

4.5 測試你的安裝

要測試安裝是否成功，建立一檔案 glibc.c 如下：

```
#include <stdio.h>

main()
{
    printf("hello world!\n");
}
```

用 "-b <安裝目錄> -nostdinc -I<安裝目錄>/include -I/usr/lib/gcc-lib/<新系統目錄>/<gcc 版本>/include" 的選項編譯：

```
% gcc -b i486-linuxglibc2 -nostdinc -I/usr/i486-linuxglibc2/include -I/usr/lib/gcc-lib/i486-linuxglibc2/2.7.2.2/include glibc.c -o glibc
```

使用 ldd 確定程式是用 glibc2 連結，而非舊有的 libc：

```
% ldd glibc
libc.so.6 => /usr/i486-linuxglibc2/lib/libc-2.0.5.so (0x4000d000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

如果通過編譯，連結也確定， 並且執行時也產生 "hello world!" 的輸出，那麼安裝就成功了。

5 安裝成主要函式庫

這一節的內容是關於將 glibc2 安裝成你的主要 C 函式庫。 任何你編譯的新程式都將會連結到這一個函式庫， 除非你用了特殊的編譯選項來連結到其它的版本。

假如你是用 RedHat 或 Debian 並且已下載相關的 rpm 或 deb 檔， 請參閱 RedHat 或 Debian 的安裝說明，你可以直接跳過這一節。

5.1 由原始碼編譯函式庫

這一節解釋如何由原始碼編譯 glibc 2 與附加套件。 如果你想要改變最佳化與組態選項或使用額外的套件，那你就必須編譯函式庫。

5.1.1 需求

- 約 150 MB 的磁碟空間
- GNU make 3.75
- gcc >= 2.7.2 (最好是 2.7.2.1)
- binutils 2.8.1 (假如是 alpha 你需要 snapshot)
- bash 2.0
- autoconf 2.12 (假如你更動了 configure.in)
- texinfo 3.11

在一安裝有 64MB 記憶體的 i586@133 機器上， 編譯完整的函式庫與附加套件大約需要三個小時。 而在 i686@200 的機器上則約需要一個半小時。

5.1.2 解開原始碼

你需要從 archive 中解出原始碼才能編譯它， 最好的方式是：

```
tar xzf glibc-2.0.6.tar.gz
cd glibc-2.0.6
tar xzf ../glibc-linuxthreads-2.0.6.tar.gz
tar xzf ../glibc-crypt-2.0.6.tar.gz
tar xzf ../glibc-localedata-2.0.6.tar.gz
```

這將會把linuxthreads、crypt與localedata等目錄放到glibc-2.0.6目錄下，如此configure才能找到這些附加套件。

5.1.3 設定組態

在`glibc-2.0.6`的目錄底下，建立一個新的目錄compile，並且cd到compile底下，所也的工作都會在這個目錄中完成，這會簡化後續的清除工作。(發展者似乎並未讓 'make clean' 作得很好)

```
mkdir compile
cd compile
```

執行`../configure`。要使用附加套件你必需用–enable-add-ons來指定，例如–enable-add-ons=linuxthreads,crypt,localedata。也許你也會想要指定安裝的目錄，要符合linux標準架構，指定爲–prefix=/usr。(當在linux系統上prefix被設爲`/usr`，configure知道要調整其它的路徑，好將libc.so以及其它重要的函式庫放到`/lib`) 完整的configure命令如下：

```
../configure --enable-add-ons=linuxthreads,crypt,localedata --prefix=/usr
```

5.1.4 編譯

要編譯與驗證，執行：

```
make
make check
```

5.2 準備安裝

無論是由原始碼編譯或是預先編譯好的版本，你現在必需移動某些檔案來迎接新的函式庫。任何新編譯的程式會被連結到glibc，但是舊有靜態連結的程式仍然倚靠著libc 5，所以你不能只是覆蓋掉舊有的版本。

1. 建立一個新的目錄來放置舊檔案：

   ```
   mkdir -p /usr/i486-linuxlibc5/lib
   ```

2. 舊的標頭檔必需從`/usr/include`移開：

   ```
   mv /usr/include /usr/i486-linuxlibc5/include
   ```

3. 建立一個新的include目錄，並且設定到其它include目錄的連結：

   ```
   mkdir /usr/include
   ln -s /usr/src/linux/include/linux /usr/include/linux
   ln -s /usr/src/linux/include/asm /usr/include/asm
   ln -s /usr/X11R6/include/X11 /usr/include/X11
   ln -s /usr/lib/g++-include /usr/include/g++
   ```

這些連結須要視你的系統而稍作調整。 最少在 Slackware 中，g++ 標頭檔案在 `/usr/local/g++-include`，而 Debian 卻放在 `/usr/include/g++`，並且將 `/usr/lib/g++-include` 連結到 `/usr/include/g++`。在後者的情形，你也許會想要將原始的 g++ include 目錄移回到 `/usr/include`。

4. 回存所有額外的標頭檔與連結。 某些非標準的函式庫會將檔案放至 `/usr/include`， 或是在 `/usr/include` 放置一連結到它們的 include 目錄下。 這些檔案與連結必須回存，如此才可以正確的使用這些額外的函式庫。
5. 將你的新函式庫路徑 (例如 `/usr/i486-linuxlibc5/lib`) 加入到 `/etc/ld.so.conf` 的頂端。 你最好要有 ld.so 1.8.8 或是更新的版本， 以避免在安裝完 glibc 之後出現一些奇怪的訊息。
6. 搬移或複製所有的舊函式庫到新的目錄。

```
mv /usr/lib/libbsd.a /usr/i486-linuxlibc5/lib
mv /usr/lib/libc.a /usr/i486-linuxlibc5/lib
mv /usr/lib/libgmon.a /usr/i486-linuxlibc5/lib
mv /usr/lib/libm.a /usr/i486-linuxlibc5/lib
mv /usr/lib/libmcheck.a /usr/i486-linuxlibc5/lib
mv /usr/lib/libc.so /usr/i486-linuxlibc5/lib
mv /usr/lib/libm.so /usr/i486-linuxlibc5/lib
cp /lib/libm.so.5.* /usr/i486-linuxlibc5/lib
cp /lib/libc.so.5.* /usr/i486-linuxlibc5/lib
```

假如你的 `/usr` 與 `/` 是在不同的 partition 上， 那 `libm.so.5` 與 `libc.so.5` 應該用複製的非搬移， 因爲一些用來啓動 linux 的程式會需要這兩個函式庫， 因此必須要位在 root partition。

7. 將 `/usr/lib/*.o` 移到新的目錄下。

```
mv /usr/lib/crt1.o /usr/i486-linuxlibc5/lib
mv /usr/lib/crti.o /usr/i486-linuxlibc5/lib
mv /usr/lib/crtn.o /usr/i486-linuxlibc5/lib
mv /usr/lib/gcrt1.o /usr/i486-linuxlibc5/lib
```

8. 在你的函式庫移動後， 更新 library cache。

```
ldconfig -v
```

5.3 由預先編譯好的套件安裝

如果你是由一已預先編譯好的 glibc 安裝， 你必須：

```
cd /
gzip -dc glibc-2.0.bin.i386.tar.gz | tar tvvf -
gzip -dc glibc-crypt-2.0.bin.i386.tar.gz | tar tvvf -
ldconfig -v
```

如果你用的是不同的系統架構或是版本，請置換適當的檔名。

5.4 由原始碼安裝

要由原始碼安裝，執行：

```
make install
ldconfig -v
```

5.5 更新 gcc specs

安裝的最後一步 (不管你是由預先編譯好的套件或是由原始碼安裝) 是更新 gcc 的 `spec` 檔，讓你可以正確的連結你的程式。要知道 gcc 所用的是那一個 spec 檔，如下輸入：

```
% gcc -v
reading specs from /usr/lib/gcc-lib/i486-unknown-linux/2.7.2.2/specs
gcc version 2.7.2.2
```

在這裡，系統是 i486-unknown-linux，而版本則是 2.7.2.2。你必須將 `/usr/lib/gcc-lib/<系統>` 複製到舊的系統目錄下：

```
cd /usr/lib/gcc-lib/
cp -r i486-unknown-linux i486-linuxlibc5
```

到原先的目錄與版本目錄下：

```
cd /usr/lib/gcc-lib/i486-unknown-linux/2.7.2.2
```

並修改在同一目錄下的 `specs`。將這檔案中的 `/lib/ld-linux.so.1` 改成 `/lib/ld-linux.so.2`，並且要將所有的 `%{...:-lgmon}` 表示式刪除，因爲 glibc 不使用 gmon 函式庫來做 profiling。你可以在 9 (Specs 檔範例) 找到一份 specs 檔的範例。

5.6 測試你的安裝

要測試安裝是否正確，請建立 glibc.c 如下：

```
#include <stdio.h>

main()
{
    printf("hello world!\n");
}
```

並且 compile 這個程式：

```
% gcc glibc.c -o glibc
```

使用 ldd 來查證這個程式是連結到 glibc2 而非你的舊 libc：

```
% ldd glibc
libc.so.6 => /lib/libc.so.6 (0x4000e000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

假如這個程式可以通過 compile，並且在執行後產生 "hello world!" 的輸出，那麼表示已經安裝成功。

6 使用非主要的 C 函式庫編譯程式

有些時候你會想要使用不同的函式庫來編譯你的程式，這一節會解釋如何達成這個目的。 我會使用在前兩節中所使用的目錄與安裝名稱，記得要改變這些名稱以符合你的設定。

6.1 一個關於使用非主要 C 函式庫的警告

在你開始編譯任何會在開機時用到的程式之前， 假如程式是動態連結，並且是在 non-root partition 被 mount 之前使用的話， 那所有連結用的函式庫都必須在 root partition 上。 在前面的安裝程序裡，我們將 glibc 安裝成主要的函式庫， 而舊的函式庫仍然保留在 `/lib`， `/lib` 會在 root partition 上， 也就是說，所有在開機時會用到的程式都能正常工作。 然而，若你的 `/usr` 是在不同的 partition 上， 且把 glibc 當成測試用的函式庫，安裝在 `/usr/i486-linuxglibc2`， 那除非 `/usr` 已經被 mount，否則所有你用 glibc 編譯的程式都將不能使用。

6.2 用測試安裝的 glibc 編譯程式

要用測試安裝的 glibc 編譯程式， 你必須要將 include 路徑重設成 glibc 的 include 路徑。 指定 "-nostdinc" 會取消正常 include 路徑， 而 "-I/usr/i486-linuxglibc2/include" 則會指到 glibc 的 include 目錄， 你也必須要指定 gcc 標準 include 目錄， 通常是 `/usr/lib/gcc-lib/i486-linuxglibc2/2.7.2.2/include`。 (這裡假設你安裝測試的函式庫在 i486-linuxglibc2， 而 gcc 的版本是 2.7.2.2)

要用測試安裝的 glibc 來連結，你必須指明 gcc 設定， 這是由 "-b i486-linuxglibc2" 這選項來達成。

對大部份的程式， 你可以簡單的將其加入 makefile 的 `$CFLAGS` 與 `$LDFLAGS` 選項中：

```
CFLAGS = -nostdinc -I/usr/i486-linuxglibc2/include -I/usr/lib/gcc-lib/i486-linuxglibc2/2.7.2.2/include -b i486-linuxglibc2
LDFLAGS = -b i486-linuxglibc2
```

假如你使用 configure 這個 script，在執行 configure 之前， 定義 shell 變數 `$CFLAGS` 與 `$LDFLAGS` (在 csh/tcsh 下用 env/setenv 在 sh/bash/其它之下用 set/export)。 這樣產生的 makefile 應該會包含正確的 `$CFLAGS` 與 `$LDFLAGS`。 不過並非所有的 configure scrpt 都會去取得這些變數， 所以在執行完 configure 之後，你還是要檢查一下 makefile， 假如需要的話，手動修改它。

假如你所要編譯的程式只呼叫 gcc，(而不直接呼叫 cpp 或 binutils) 你可以使用以下的 script，省去每次指定所有選項的麻煩：

```
#!/bin/bash
/usr/bin/gcc -b i486-linuxglibc2 -nostdinc \
             -I/usr/i486-linuxglibc2/include \
             -I/usr/lib/gcc-lib/i486-linuxglibc2/2.7.2.2/include "$@"
```

編譯時你可以使用這個 script 而不是 gcc。

6.3 當 glibc 是主要函式庫時用 libc 5 編譯程式

當你已經將 glibc 安裝成主要的函式庫之後，又想用舊的函式庫來編譯程式的時候，你必須將 include 路徑重設成舊的路徑。指定 "-nostdinc" 將會使預設的路徑無效，而 "-I/usr/i486-linuxlibc5/include" 會將 include 的路徑指到舊函式庫的 include 路徑，你也必須要指定 "-I/usr/lib/gcc-lib/i486-linuxlibc5/2.7.2.2/include" 以含入 gcc 所需的 include 檔。注意要視你新的目錄與 gcc 版本來調整這些路徑。

要用舊的 libc 來做連結，你必須指定 gcc 所用的架構。這是由 "-b i486-linuxlibc5" 這個選項所指定的。

對大部份的程式，你可以簡單的將這些選項加入到 makefile 裡的 `$CFLAGS` 與 `$LDFLAGS` 選項裡：

```
CFLAGS = -nostdinc -I/usr/i486-linuxlibc5/include -I/usr/lib/gcc-lib/i486-linuxlibc5/2.7.2.2/include -b i486-linuxlibc5
LDFLAGS = -b i486-linuxlibc5
```

假如你使用 configure 這個 script，在執行 configure 之前，定義 shell 變數 `$CFLAGS` 與 `$LDFLAGS` (在 csh/tcsh 下用 env/setenv 在 sh/bash/或其它之下用 set/export)。這樣產生的 makefile 應該會包含正確的 `$CFLAGS` 與 `$LDFLAGS`。不過並非所有的 configure scrpt 都會去取得這些變數，所以在執行完 configure 之後，你還是要簡檢查一下 makefile，假如需要的話，手動修改它。

假如你所要編譯的程式只呼叫 gcc，(而不直接呼叫 cpp 或 binutils) 你可以使用以下的 script，省去每次指定所有選項的麻煩：

```
#!/bin/bash
/usr/bin/gcc -b i486-linuxlibc5 -nostdinc \
             -I/usr/i486-linuxlibc5/include \
             -I/usr/lib/gcc-lib/i486-linuxlibc5/2.7.2.2/include "$@"
```

編譯時你可以使用這個 script 而不是 gcc。

7 編譯 C++ 程式

因爲 libg++ 使用了部份的數學函式庫，也就是說會連結到 libm。由於你目前的 libg++ 是用舊版的函式庫所建立的，所以你必須重新用 glibc 來 compile libg++ 或是取得一份預先編譯好的版本。最新版的 libg++ source 與用 glibc 所連結的預先編譯版可以在 `<ftp://ftp.yggdrasil.com/private/hjl/>` 找到。

7.1 用測試的 C 函式庫安裝 libg++

假如你已經將 glibc 安裝成測試用的函式庫，那你必須安裝到 glibc 所安裝的目錄下 (例如由前一節所示的 `/usr/i486-linuxglibc2`)。 假如你要安裝預先編譯好的版本 (這也是我建議的安裝方式，因爲我一直都不能用這種方式成功地編譯 libg++)， 你必須將檔案解到暫存的目錄下， 將 `usr/lib/` 下的所有檔案搬到 `<install directory>/lib/`， 並將 `usr/include/` 下的所有檔案搬到 `<install directory>/include/` 下， (記得先刪除 `include/g++` 這個連結！) 再將 `usr/bin/` 下的檔案搬到 `<install directory>/bin/` 。

7.2 在主要 glibc 下安裝 libg++

若你已經將 glibc 安裝成主要的函式庫，假如還想要用舊的 libc 來編譯 g++ 的程式， 那你必須先將舊的 libg++ 檔案移至舊的 libc 的目錄下。 也許最簡單的方式就是如前一節用 libc 5 來安裝一份新版的拷貝， 然後再正常安裝 glibc 版。

7.3 使用非主要的 C 函式庫來編譯 C++ 程式

假如你想用非主要的 C 函式庫來編譯 C++ 程式， 那你必須要加入 g++ 的 include 目錄。 在上面的例子中， 對測試用的 glibc 是 `/usr/i486-linuxglibc2/include/g++`， 而對主要 glibc 則是 `/usr/i486-linuxlibc5/include/g++`， 這通常都可以由添加至 `$CXXFLAGS` 這個變數來完成：

```
CXXFLAGS = -nostdinc -I/usr/i486-linuxglibc2/include -I/usr/lib/gcc-lib/i486-linuxglibc2/2.7.2.2/include -I/usr/i486-linuxlibc5/include/g++ -b i486-linuxglibc2
```

8 報告臭蟲

假如你以爲這函式庫有錯，請先閱讀 FAQ，也許已經有人遇到過這個問題， 並且也已經有了簡單的解決方法。 另外，你也應該查看 `INSTALL` 檔中的 "Recommended Tools to Install the GNU C Library" 一節， 因爲有些錯誤不在於 glibc 而是在於這些工具。

假設你找到了臭蟲，請先確定那的確是臭蟲。 一個判斷的好方法是看看 glibc 的行爲是否與其它的 C 函式庫相同。 假如相同的話，那也許你是錯的，而函式庫是正確的。 (但也不一定如此) 假如不相同的話，那肯定兩者之中有一個是錯的。

接著，請到 `<http://www-gnats.gnu.org:8080/cgi-bin/wwwgnats.pl>`， 並且查看臭蟲資料庫，確定這個問題尚未有人提出。 你也應該看看 `BUGS` (內付於 libc 中) 所列出的目前已被發現的臭蟲。

一旦你確定找到了一隻新的臭蟲，請試著將它縮小到最小的 test case。 就 C 函式庫而言，你只需要將它縮小到單獨一個函式呼叫(假如可以的話)。 這應該不是太困難的事。

最後一個步驟是就你的 test case 提出臭蟲報告。 要送出臭蟲報告，請包含你的 test case、你所得到的結果、你所期待的結果、 你認爲可能的原因在那 (只要你曾想過的任何原因)、你的系統類別、你所使用的 GNU C 函式庫、 GNU CC 編譯器、GNU binutils 的版本。另外， 請附上執行 `configure` 時產生的 `config.status` 與 `config.make`， 這兩個檔案會在你執行 `configure` 時候的目錄下。

所有的臭蟲報告都應該使用 `glibcbug` 這個 GNU libc 所附的 shell script 送至 *bugs@gnu.org* `<mailto:bugs@gnu.org>` (舊的位址 *bugs@gnu.ai.mit.edu* `<mailto:bugs@gnu.ai.mit.edu>` 也仍然可以使用)。 或是到 GNATS 網站上利用 `<http://www-gnats.gnu.org:8080/cgi-bin/wwwgnats.pl>` 這個介面來做。

你的建議與問題應該送到 *bugs-glibc@prep.ai.mit.edu* `<mailto:bugs-glibc@prep.ai.mit.edu>` 這個郵遞列表。 假如你沒有閱讀新聞群組 gnu.bug.glibc， 那麼你可以要求 *bug-glibc-request@prep.ai.mit.edu* `<mailto:bug-glibc-request@prep.ai.mit.edu>` 加入這個列表。

請不要將 GNU C 函式庫的臭蟲報告送到 <bug-gcc@prep.ai.mit.edu>， 這是爲了 GNU CC 而設立的，而 GNU CC 與 GNU C 函式庫是由不同人們的所維護的。

9 Specs 檔範例

這裡是一份 specs 的範例。這將使 gcc 用 glibc2 來編譯與連結。 它應該位於 `/usr/lib/gcc-lib/<`新系統目錄`>/<`gcc 版本`>`. 假如你用的是 x86 的系統，那你也許可以直接將這一段複製到檔案裡去。

```
*asm:
%{V} %{v:%{!V:-V}} %{Qy:} %{!Qn:-Qy} %{n} %{T} %{Ym,*} %{Yd,*} %{Wa,*:%*}

*asm_final:
%{pipe:-}

*cpp:
%{fPIC:-D__PIC__ -D__pic__} %{fpic:-D__PIC__ -D__pic__} %{!m386:-D__i486__} %{posix:-D_POSIX_SOURCE} %{pthread:-D_REENTRANT}

*cc1:
%{profile:-p}

*cc1plus:

*endfile:
%{!shared:crtend.o%s} %{shared:crtendS.o%s} crtn.o%s

*link:
-m elf_i386 %{shared:-shared}   %{!shared:     %{!ibcs:       %{!static:       %{rdynamic:-export-dynamic}     %{!dynamic-linker:-dynamic-linker /lib/ld-linux.so.2}}

*lib:
%{!shared: %{pthread:-lpthread}        %{profile:-lc_p} %{!profile: -lc}}

*libgcc:
-lgcc

*startfile:
%{!shared:      %{pg:gcrt1.o%s} %{!pg:%{p:gcrt1.o%s}                  %{!p:%{profile:gcrt1.o%s}                         %{!profile:crt1.o%s}}}}    crti.o%s %{!shared:

*switches_need_spaces:

*signed_char:
%{funsigned-char:-D__CHAR_UNSIGNED__}

*predefines:
-D__ELF__ -Dunix -Di386 -Dlinux -Asystem(unix) -Asystem(posix) -Acpu(i386) -Amachine(i386)

*cross_compile:
```

```
0

*multilib:
. ;
```

10 雜項

10.1 進一步的資料

10.1.1 相關網頁

- *FSF's GNU C Library Home Page* `<http://www.gnu.org/software/libc/libc.html>`
- *Using GNU Libc 2 with Linux* `<http://www.imaxx.net/~thrytis/glibc/>`
- *Installing glibc-2 on Linux* `<http://huizen.dds.nl/~frodol/glibc/>`.
- *Debian libc5 to libc6 Mini-HOWTO* `<http://www.gate.net/~storm/FAQ/libc5-libc6-Mini-HOWTO.html>`.

10.1.2 新聞群組

- *comp.os.linux.development.system*
- *comp.os.linux.development.apps*
- *linux.dev.kernel*
- *gnu.bugs.glibc*

10.1.3 郵遞列表

Glibc 2 Linux 討論列表

這個郵遞列表是爲了讓安裝有 glibc，也就是新的 GNU C 函式庫，的使用者討論用的。 議題包括相容性的問題，以及在 linux/glibc 環境下程式的編譯問題。 要訂閱這個郵遞列表， 請寄一封信至 *Majordomo@ricardo.ecn.wfu.edu* `<mailto:Majordomo@ricardo.ecn.wfu.edu>`， 信的內容爲 "subscribe glibc-linux <your email address>"。

10.2 貢獻者列表

大部份的資訊都是由 *GNU Libc web page* `<http://www.gnu.org/software/libc/libc.html>` 以及 Ulrich Drepper 的 <drepper@gnu.ai.mit.edu> glibc 2 announcement 所偷來的， Andreas Jaeger <aj@arthur.rhein-neckar.de> 則提供了部份的臭蟲報告。

以下各位都曾對本文件提供相關資訊以及意見：

- Allex Juang<allex@ms2.accmail.com.tw>

- Mark Brown <M.A.Brown-4@sms.ed.ac.uk>
- Ulrich Drepper <drepper@gnu.ai.mit.edu>
- Scott K. Ellis <ellis@valueweb.net>
- Aron Griffis <agriffis@coat.com>
- Andreas Jaeger <aj@arthur.rhein-neckar.de>
- Frodo Looijaard <frodol@dds.nl>
- Ryan McGuire <rmcguire@freenet.columbus.oh.us>
- Shaya Potter <spotter@capaccess.org>
- Les Schaffer <godzilla@futuris.net>
- Andy Sewell <puck@pookhill.demon.co.uk>
- Gary Shea <shea@gtsdesign.com>
- Stephane <sr@adb.fr>
- Jan Vandenbos <jan@imaxx.net>

這份文件的翻譯是由以下數人所完成的：

- 中文： Allex Juang<allex@ms2.accmail.com.tw>
- 法文： Olivier Tharan <tharan@int-evry.fr>
- 日文： Kazuyuki Okamoto <ikko-@pacific.rim.or.jp>

10.3 回饋

除了撰寫這份 HOWTO、維護 *glibc 2 for Linux* `<http://www.imaxx.net/~thrytis/glibc>` 這網頁、並且在我的機器上使用之外，我與 glibc 計劃完全無關， 我也不了解這方面的議題，但會儘可能的對寄給我的問題提供幫助。 我也很歡迎你提出任何的回饋、修正、或是建議，請將它們寄到 *ejg3@cornell.edu* `<mailto:ejg3@cornell.edu>`。

10.4 Copyright.

此份文件的版權屬於 Eric Green，這份文件可以在 LDP 的執照下散佈。