移植 Linux 至 ARM 嵌入式處理器

陳銘宏 林盈達
國立交通大學資訊科學系
300 新竹市大學路 1001 號
Tel: 03-5712121 ext. 56667
E-MAIL：{mhchen,ydlin}@cis.nctu.edu.tw

摘要

Linux 是一個 Open Source 的 UNIX-like 作業系統，除了有著廣大的支援社群以外，穩定、模組化、擁有廣大的應用免費應用軟體支援是它主要的優點。ARM 嵌入式處理器則是目前應用在嵌入式系統領域中，佔有率最高的處理器，同時也是 Linux 核心目前所能夠支援的處理器之一，然而要使得 Linux 在 ARM 嵌入式處理器上運作，勢必要經過移植 (porting) 的過程，也就是要將平台相依 (platform dependent) 的部分做適當的修改。而微小化的作業系統又是目前嵌入式作業系統的趨勢，因此近年來微型化的語言函式庫(library)也是大行其道，但使用微型化的函式庫又會逼迫使用者需要重新利用新的函式庫編譯(compile)所有需要的應用程式，讓系統能夠轉移到新的函式庫上。在本文裡，我們實際將 Linux 與 uClibc 安裝到一個以 Intel® StrongARM™ SA-110 為處理器核心的實驗版上，並且讓系統在能夠運作的狀況下作最小化。在保留基本的 Shell、少數核心工具的功能和動態連結函示庫的設定下，利用變更函示庫和傳統工具集的方式，可以將系統映像檔由 6.2MB 縮減縮減到 1.7MB 或更小的大小。此外，透過觀察 Kernel 啟動的程序和原始碼的佈局,則可清楚的指出,開機流程和硬體裝置驅動程式是和平台相依性最為有關的部分。我們會在本文中闡明其運作機制。瞭解這些部分，將是移植 Linux 到不同平台上的主要關鍵。

1. 簡介

移植的廣泛定義，是讓一套軟體可以在一套選定硬體平台上正常運作。在這篇文章中,我們談的是如何移植 Linux 這套作業系統到 ARM 嵌入式處理器的平台上。由於 Linux 核心團隊一直堅持著免費與開放式程式碼的精神，因此許多的自由軟體設計師都願意在這樣的一個平台上持續的開發各式強大的軟體,其中的大

多數軟體，也都以 Open Source 的方式提供給 Internet 上的所有人來使用。基於開放原始碼與具有完整度高的免費組件的優勢，近年來，Embedded Linux 也越來越受到市場的注目。而且選用 Linux 取代其他商業化的嵌入式作業系統，如表 1 所示，相較於其他現有作業系統，除了可以減低系統授權所需要的費用。廣大的社群也提供了許多免費的支援和不同平台的解決方案，如此一來，就能夠大幅降低開發硬體規格多變的嵌入式系統時，移植作業系統部分所需增加的負擔。

	Non- Commercial Embedded Linux	**Commercial Embedded Linux**	**VxWorks**	**QNX**	**pSOS**	**Windows CE**
Price	Free	Depend on distributions	$130M (20M royalty)	$20M	$105M ($15M royalty)	$3M royalty $2M Tools
Tools Prices	Free	Free	$7,500~ $10,000	$2,955	$4,955	$995
Royalty	Free	Free	$1~$35	$3~$80	<$1~$30	<$5~$30
Focus	Networking Datacom	Networking Datacom Handhold device	Telecom, Datacom, Digital Imaging, CE	Industrial automation Medical system, Telecom, Datacom, CE	Automobile industry, Some CE	Higher-end apps
Strengths	1.Open Source 2.Free 3. Complete POSIX compliant	1.Partial Open Source, 2.Cover more hosts than original Linux 3.Tech Support and good IDE	1.Most complete integrated environment 2.Largest coverage of CPU and hosts	1.micro-kernel architecture 2.Complete POSIX compliant 3.Most scalable	1.Very good IDE 2.Optimized for Motorola processors and DSPs	1.Win32 2.NT Connectivity 3.Robust GUI

表 1. 現有嵌入式系統比較表[1]

在移植時要特別注意的是，Linux 核心本身具有良好虛擬記憶體系統，因此他僅能在具有記憶體管理單元（Memory Management Unit, MMU）的處理器上運作，如果想要在沒有 MMU 的處理器上運行 Linux，則需要使用專為此種處理器而開發的 uClinux 套件。

在另一方面，ARM 處理器核心是一個 32 位元的精簡指令集架構（RISC,

Reduced Instruction Set Computer)，因此，其執行核心設計較為簡單，因此也具有著耗電量低的優勢，被廣泛的運用到各種嵌入式系統裡。通常一顆嵌入式處理器會包含一個處理器核心還有許多的運算周邊，以我們在本文中所使用的實驗版為例，它使用了 Intel® StrongARM™ 110 的處理器，裡面包含了 Intel®的 StrongARM™處理器核心（相容於 ARM 的指令集架構）以及中斷控制器、計時器等等的周邊。從表 2 我們發現有些處理器核心並沒有內建記憶體管理單元，而在實際上，在 StrongARM™與 ARM9 系列之前的所有 ARM 處理器核心，都是沒有記憶體管理單元的，也就是說原始的 Linux 核心並沒有辦法在上面運行，也就是必須使用 uClinux 核心才可以讓 Linux 在上面運行。

Type	CPU Core	Cache Size (Inst/Data)	Tightly Coupled Memory	Memory Manage-ment	AHB Bus Interface	Thumb	DSP	Jazelle	Clock MHz **
Embedded Cores	ARM7TDMI	No	No	No	Yes*	Yes	No	No	133
	ARM7TDMI-S	No	No	No	Yes*	Yes	No	No	100-133
Intel ARM-based Processors	StrongARM	16K/8K	No	MMU	N/A	No	No	No	206~233
	Intel XScale	32K/32K	No	MMU	N/A	Yes	Yes	No	400+

表 2. ARM 處理器核心比較表

近年來，日益增加的嵌入式系統需求，也使得過去隨著功能擴充，日益肥大的作業系統和函式庫有必要進行瘦身，由於 Linux 系統核心採用可模組化的設計，因此可以輕易的將不需要的模組由核心中移除，輕易的達到縮小系統核心的目標，而函式庫則有 uClibc 和 dietlib 等用於取代傳統 glibc 的微型函式庫出現，但是如表 3 所示，在比較過數種不同的微型 library 後，uClibc 是目前最適合使用的 library，因此在接下來的內容中，我們將提到如何利用 uClibc 來取代傳統的 glibc，並且利用 uClibc 所提供的工具來建立一個新的系統。

	GNU C Library	uClibc	Diet Libc	newlib	Sglibc
Size	Largest	Small	Smallest	Small	Large
Compatibility	Yes	Good	Bad	Unknown	Good

Speed	Faster	Fast	Fast	Fast	Fastest
Portability	Yes	Yes	Yes	Yes	Yes
MMU-less support	No	Yes	Yes	Unknown	No
Licensing	LGPL	LGPL	GPL	BSD, GPL...	LGPL
Still active	Yes	Yes	Yes	Yes	No
Note	The standard C library for Linux	Rewrite for embedded Linux. Support standard libc API	Reduce size by breaking libc API compatibility	Managed by Red Hat . A collection of several library parts	A patch for GNU C library to discard sections of glibc source

表 3. 微型化 library 比較表

而除了函示庫和核心以外，最能夠節省空間的就是縮小應用程式所佔用的空間，除了移除所有不必要的軟體以節省空間以外，我們也可以尋找較小的替代軟體來取代現有的工具。BusyBox 就是在這種需求下所產生的工具，他利用單一個程式提供大部分傳統工具集所提供的服務，並且可以依照不同的需求進行特製化，讓 BusyBox 執行檔的大小能夠有所調整，以達到進一步縮減整個作業系統大小的目標。

在這樣的介紹以後，我們可以發現，要建構一個較小的嵌入式 Linux 環境，我們將會需要去重新的編譯核心，建構 uClibc 的編譯與執行環境，並且將舊有的工具集換為較小的替代工具。在下一節，我們將簡單的介紹在進行移植前我們所需要考慮的事情。而在第三節，我們將描述如何一步一步的將 Linux 安裝到我們的實驗版上。在第四節，我們將會討論如何讓系統所佔用的空間更加的縮小，並且討論在安裝過程中可能遇到的問題。最後將是我們的結論。

2. 移植

所謂的移植(porting)，從軟體的角度來看，就是使一套軟體在不同平台上正常運行的過程，在此我們的軟體是指作業系統，而平台是指硬體平台。我們都知道作業系統是介於應用程式和硬體間溝通的橋樑，因此當底層的硬體有所改變時，作業系統和平台相依（platform dependent）的部分就必須跟著做變動，所以在移植前，瞭解目標平台（target platform）的硬體規格是相當重要的前置工作。除此之外，我們對於要移植的作業系統，其中與平台相依的各個部分要先熟悉，如此才能將這些部分迅速的轉移到其他的平台上。

因此，在進行移植前，有四個問題是要先釐清的，它們關係到移植規模的大小和困難度。

■ 目標平台

目標平台裡包含了所使用的處理器和處理器外的周邊裝置，而處理器中可能也整合了一些運算周邊，舉如中斷控制器、計時器等等。因此在進行移植前，我們可以從目前作業系統的支援程度來決定哪些部分要改寫以及哪些部分可以套用原本的程式碼。以由 Intel® x86 Processor 移植到 Intel® StrongARM™ 110 為例，這兩種處理器的核心可以說是完全不相同的，而且 x86 系列的處理器幾乎都整合了 FPU，但在 ARM 上則幾乎都沒有 FPU 的支援，而在內部核心上，使用的執行碼形式也不同，x86 是使用 CISC 為基礎，而 ARM 則是使用 RISC 為基礎，因此我們需要做處理的部分，除了產生目標碼的方式以外，還有包含暫存器等等系統底層的部分需要去做變更。但如果是由 StrongARM™移植到 Xscale™上，因為兩者的核心大致上可以相容，因此在新的平台上，我們可以僅需要做較小部分的修改，就可以在另一種不同的核心上運行。如此一來，就可以針對剩下的周邊撰寫新的裝置驅動程式，而不需要在核心的修改上花太多的時間。

■ 記憶體管理單元(Memory management unit)

記憶體管理單元負責作業系統中虛擬記憶體保護的工作，Linux 虛擬記憶體

的機制就是靠它來完成的，當我們要將 Linux 移植到一個沒有記憶體管理單元的平台上，所需要做的修改就非常的多了，雖然目前已經有 uClinux 的出現，但是他所支援的處理器還是有限的，因此在選擇目標平台的處理器時要相當的謹慎。

■ 記憶體對映（Memory mapping）

嵌入式系統，通常為一個無磁碟的裝置，並且配置了有限容量的 SDRAM 和 Flash Rom 或 MTD(Memory Technology Device)，其中記憶體控制器（Memory controller）負責兩者在處理器位址空間（Address space）的對映，不同於 PC 的是，每種平台的記憶體對映位置不一定相同，這是由於硬體預設值設定的不同所導致的，因此會有平台硬體規格相似但記憶體對映位置不同的問題產生，所以在進行移植時，可能只需要藉由重新規劃記憶體控制器就可以使某平台的核心在新平台上面運作，這種狀況特別常在相似系列的核心出現。

■ 系統儲存裝置

由於嵌入式系統通常不會使用 Hard Disk 作為儲存系統的空間，因此儲存媒體本身是否被作業系統支援也是需要被考慮的。除此之外，依照不同的需求，也可能需要選擇不同的檔案系統格式，如 JFFS2 等適用於嵌入式系統的檔案系統，都是可能需要被使用的。

3. 移植 Linux 到 ARM 平台

在前一節我們瞭解，要將 Linux 移植到 ARM 嵌入式處理器上，必須先釐清楚 Linux 哪些部分為平台相依的，這也正是本文要探討的問題之一。但在探討相依問題前，我們先試著將 Linux 安裝在以 ARM 為處理器核心的實驗版上。因為，我們認為，要瞭解一個系統，將它實際安裝並且執行是瞭解他們相依性的第一步驟。

■ 硬體規格

我們所選用的目標平台(target platform) 是由 Simtec Electronics 所製造的 StrongARM™ 110 Evaluation Board(以下簡稱 EB110ATX)，它是 StrongARM™ 嵌入式系統的一個典型開發平台，核心使用了一顆 Intel 的 StrongARM™ 110 233Mhz 嵌入式處理器，此外還使用了 digital 21285 與 Ali M1543C 來提供周邊裝置的支援，在這次的實驗中，我們使用了標準的 IDE Hard Disk 做為核心和應用程式的儲存裝置，但是只要在編譯核心與封裝時做一些修改，也可以輕易的改為使用 Compact Flash 卡或者其他媒體開機。我們將使用這一塊實驗版，並配備 4G 的硬碟、128MB 的 SDRAM、Realtek 8193D 網路卡與 S3 Trio64V+顯示卡作為實驗的環境。

3.1. 安裝步驟

當我們拿到 EB110ATX 這塊實驗版時，上面並沒有任何的作業系統，也沒有任何的編譯環境，因此我們需要在另一台 Linux 上建立一個 cross-compile 環境來編譯稍後要在 EB110ATX 上使用的核心與應用程式。因此我們首先就會開始建立 uClibc 的編譯環境設定，接著就是編譯核心，並把編譯出來的檔案安裝到硬碟上，最後利用我們編譯好的檔案將 EB110ATX 啟動。

3.2. 建立 cross-compile 環境

所謂建立 cross-compile 環境，指的是建立一個包含完整 toolchain，以便用來編譯原始碼為可執行碼的環境。而 toolchain 就是指一連串的編譯工具，因為在編譯原始碼為可執行碼的過程中，會依序的被執行到，因此被稱為 toolchain。toolchain 的 target platform 和目前的系統往往並不一定相同，跨執行器或者函示庫都是有可能的。由於我們的目標平台上計畫使用 uClibc 來取代 glibc，在察看了 uClibc 的網站後，我們發現他們提供了一個稱為 buildroot 的工具，可以用於方便的建立一個完整的 uClibc 執行/開發環境，另外也提供了完整的 toolchain 套件，可以建立一個完整的開發環境。

3.3. 取得 buildroot

目前因為在 uClibc 網頁上所提供的下載並不是最新的版本，因此我們建議使用 CVS 的方式來取得最新的原始碼

```
cvs -d:pserver:anonymous@uclibc.org:/var/cvs login
cvs -z3 -d:pserver:anonymous@uclibc.org:/var/cvs co -c
cvs -z3 -d:pserver:anonymous@uclibc.org:/var/cvs co -P buildroot
```

3.4. 設定 buildroot

buildroot 工具其實是一個結合了 toolchain 和一些預先寫好的 makefile 檔案所組成的套件，當成功的利用 CVS 下載完以後，我們還需要對 Makefile 做一些修改

首先，因為我們的目標平台是以 ARM 為核心的，因此將 ARCH:=i386 換為 ARCH:=arm，如圖 1 所示。

```
# What sortof target system shall we compile this for?
#ARCH:=i386
ARCH:=arm
#ARCH:=mips
#ARCH:=mipsel
#ARCH:=powerpc
#ARCH:=sh4
# Busybox link failing due to needing libgcc functions that are statics.
#ARCH:=cris
```

圖 1. 設定目標平台

接著就是設定需要哪些套件，在第三段的區域可以看到套件編譯設定的部分，如圖 2 所示，在這裡，我們將把需要的套件打開，如果只是需要一個執行環境，那請使用預設即可，至於 iptables 和 dhcp 等是否需要則看個人需求而定。

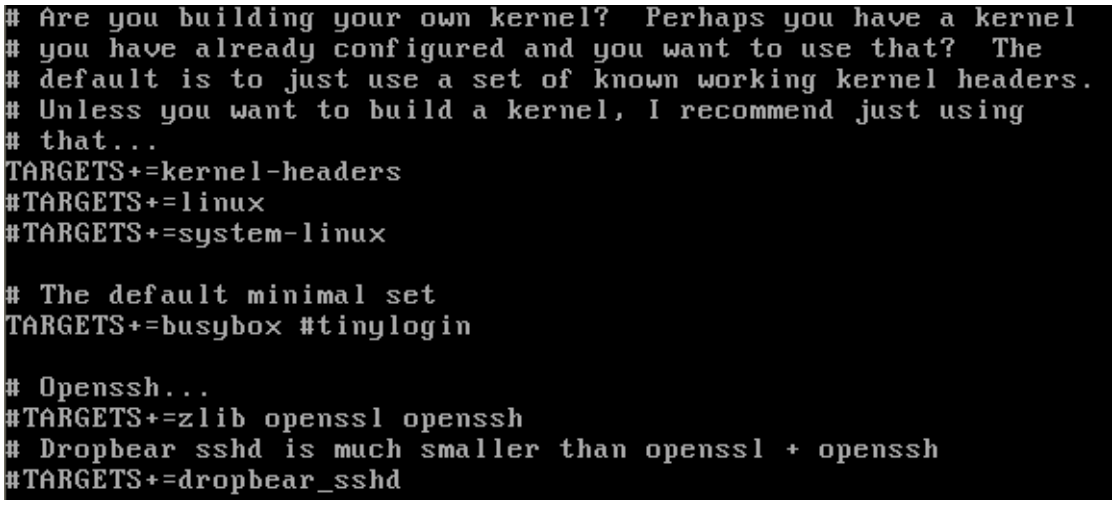

圖 2. 設定欲編譯安裝的套件

如果需要一個可以開發程式的環境，那我們有兩種選擇，第一種就是使用在編譯套件過程中，buildroot 會自動生成的 toolchain，在完成 buildroot 的 toolchain 編譯以後，我們可以在 buildroot/build_arm/staging_dir 下面找到可以編譯 ARM 執行碼的 toolchain。第二種選擇就是直接建立一個可以開發程式的 ARM 環境，這時就需要把 gcc_3_3_target、ccache_target 等其他的 TARGET 選項打開，如此一來才可以編譯出具備開發工具的執行環境。

3.5. 開始編譯

我們只要鍵入簡單的 make，buildroot 就會自動幫我們下載所需要的原始檔，並且進行開始編譯。如果一切順利沒有錯誤的話，不久後會看到一串訊息如圖 3 所示，恭喜，這時我們已經得到一個建立完成的系統映象檔了。

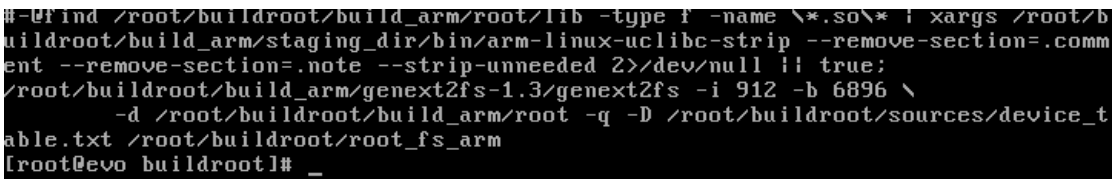

圖 3. 映象檔建立完成

接著我們可以用以下的指令將建立完成的 Image 內容放置到要在 ARM 平台上使用的硬碟分割區中。

```
dd if=root_fs_arm of=/dev/hdg1
e2fsck -f /dev/hdg1
resize2fs -p /dev/hdg1
```

當然我們也可以用更簡單的方式，就是直接複製 buildroot/build_arm/root 下所有的檔案到目標分割區下，如使用以下指令來進行複製。

```
cp –Raf buildroot/build_arm/root/* /mnt/
```

3.6. 編譯與安裝核心

這時我們已經有了一個 uClibc 的開發或執行環境，但是我們還缺少重要的 Linux 核心，我們重新回到 buildroot 的 Makefile 設定中，將 linux 的選項打開，如圖 4 所示，以便重新編譯並建立核心。

```
# Are you building your own kernel?  Perhaps you have a kernel
# you have already configured and you want to use that?  The
# default is to just use a set of known working kernel headers.
# Unless you want to build a kernel, I recommend just using
# that...
TARGETS+=kernel-headers
TARGETS+=linux
#TARGETS+=system-linux

# The default minimal set
TARGETS+=busybox #tinylogin

# Openssh...
#TARGETS+=zlib openssl openssh
# Dropbear sshd is much smaller than openssl + openssh
TARGETS+=dropbear_sshd
```

圖 4. 開啟編譯核心的選項

完成 Makefile 的修改以後，還有一些其他需要重新設定的部分，因為 buildroot 的原始開發環境是在 i386 上面，因此有一些部分將會需要做一些調整與變動。特別是需要的 Patch 和在 x86 平台上將會完全的不相同。首先我們到 buildroot/sources 目錄下，將原來的 kernel-patches 目錄作備份，並且移除裡面原來的所有檔案。接著到 ARM Linux 網站，我們可以在 developer 頁面中找

到不同版本的 ARM Linux patch，下載符合目前 kernel 版本的 patch 到 buildroot/sources/kernel-patches 中，並且在檔名的最前端加上 001，如將檔名更名為 001patch-2.4.26-vrs1.bz2 。接著我們編輯 buildroot/make/linux.mk，確定自己要下載編譯的 kernel 版本編號，並且將 LINUX_VERSION 換為加入 ARM 的 Patch 後的版本名稱，如 2.4.26-vrs1。

回到 buildroot 目錄下，重新鍵入 make，接著會下載 kernel 的原始碼，並且去執行剛剛下載的 patch 檔案，開始編譯以後，不久會看到 error 並停止編譯，此時。到 buildroot/build_arm/linux-2.4.26-vrs1/arch/arm/def-configs 下，尋找適合的預設設定檔，在我們的環境下，選用 footbridge 檔案，接著複製到 buildroot/build_arm/linux-2.4.26-vrs1 下並覆蓋過原來的.config 檔案。接著回到 buildroot/build_arm/linux-2.4.26-vrs1，執行 make oldconfig 或 make menuconfig，依照環境需求重新選擇需要的模組，並確認打開 Foorbridge Implementations 中的 CONFIG_ARCH_CATS，以便讓核心支援目前的環境。最後回到 buildroot 目錄下，重新執行 make，順利的話就可以在 buildroot/build_arm 下看到 buildroot-kernel 檔案，這就是剛剛我們所編譯出來的核心檔案，而 System.map 則可以在 buildroot/build_arm/linux-2.4.26-vrs1 下找到。接著我們可以將編譯出來的檔案用下列的方式作一些移動，然後重新執行 make 建立 file system image，將核心也包入檔案中，方便未來使用。

```
mkdir build_arm/root/boot
cp build_arm/buildroot-kernel build_arm/root/boot/vmlinuz
cp build_arm/linux-2.4.26-vrs1/System.map build_arm/root/boot/System.map
cd build_arm/root
ln –s boot/vmlinuz vmlinuz
cd ../../
make
```

接著，可以依照之前的方式，將帶有核心的系統映象檔置入目標硬碟中，然後就可以準備開始將系統轉移到 ARM 平台上了。

3.7. 啟動系統

截至目前為止，我們已經完成了部分應用工具和核心的編譯，也將檔案放入了 ARM 平台上的硬碟中，接著就是要正式的啟動系統。這部分會因為平台所使用的 Bootloader 不同而在指令上會有不小的差異，但大致上幾乎都差不多。在我們的環境中，系統啟動後，可以在終端機上看到如圖 5 所示的啟動畫面。

```
0,10,0: vendor 0x5333 product 0x8811 (VGA display, rev 0x54)
0,16,0: vendor 0x10b9 product 0x5229 (IDE mass storage, interface 0xfa, rev 0xc1
)
wd0: <ST34310A, 7AX0HL0X, 3.07>
wd0: 4111MB, 8354 cyl, 16 head, 63 sec, 512 bytes/sec
cd0: <ASUS CD-S500/A, , V1.2D>
cd0: 25MB
0,17,0: vendor 0x10b9 product 0x7101 (miscellaneous bridge)
0,20,0: vendor 0x10b9 product 0x5237 (USB serial bus, interface 0x10, rev 0x03)

>> Cyclone 1.30 (09:41 30 Jul 2002) firmware for Chalice CATS (X86EMU)
>> Copyright 1998,1999 BCCS Limited
>> Copyright 2001,2002 Simtec Electronics
>> Machine id 710000008d3f1be2

128 MB memory available

bootable devices are:
        wd0     cd0
Autobooting in 5 seconds
boot>
```

圖 5. EB110ATX 啟動畫面

接著只要呼叫 Bootloader 去載入 Linux 核心即可，也就是執行指令如下。

```
boot wd0a:vmlinuz root=/dev/hda1
```

當我們看到如圖 6 的畫面時，Linux 系統就已經啟動了，接著就可以換到 ARM 系統的顯示卡和鍵盤上，開始使用我們的 ARM Linux 了。

```
boot> boot wd0a:vmlinuz root=/dev/hda1
MBR Partition a: sector       63 len  7807527 active
symlink: new destination is: boot/vmlinuz-2.4.26-vrs1
Loading wd0a:vmlinuz to 10000000
linux (CATS): 787272 total=0xc0348 (768 KB)
exec linux...
linux: moving kernel (10000000 to 10008000) 787272
Uncompressing Linux.....|..... done, b
ooting the kernel.
```

圖 6. 載入並啟動 Linux 核心

3.8. 系統比較

依照我們目前的設定，我們可以建構出一個含有 Kernel Image、uClibc Library 和 Busybox 的 Image 檔案，而與使用一般 Library 和一般工具集的相比較，在粗略的歸納後我們得到如表 4 的結果， 如此可以發現，系統其實在變更了工具集和函示庫後，竟然可以省下 73.27%的空間，這樣縮減後的大小，可以說是相當適合在嵌入式平台上使用了。

	傳統工具&函示庫	uClibc&Busybox	縮減比例
核心(Kernel)	1122186	828635	26.16%
基本工具(/sbin)	1362013	40960	96.99%
一般工具(/bin)	1946104	381952	80.37%
函示庫(/lib)	2118396	498688	76.46%
合計	6548699≒6.24MB	1750235≒1.67MB	73.27%

表 4. 原系統與新系統所佔空間比較表

4. 深入 Linux 系統啟動與平台相依性

現在我們已經能夠在 ARM 平台上運行 Linux 了，接著我們繼續深入討論系統的開機流程(bootstrap)和與平台相依性有關的部分，主要可以分為開機程式與系統核心兩大部分，開機的程序如圖 7 所示。

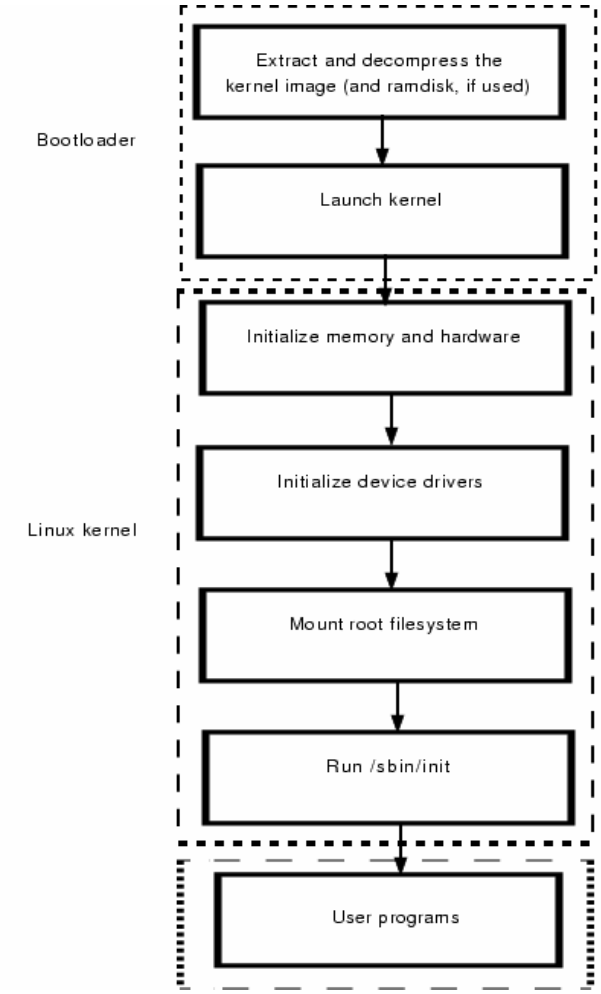

圖 7. Linux 2.4.x 開機程序

■ 開機程式與系統初始化

想要在一台機器上啟動一個作業系統，首先就是要能夠將處理器初始化，接著將核心載入到某個固定的起始位址上，這個位置通常指的是 SDRAM 的起始點，會因處理器的不同而有所區分，也就是說，上述這些行為是具有平台相依性問題的，而這些動作是由開機程式(bootstrap loader)來進行。接著就是將核心的本文(text)位置放到正確的位址，如 Linux 2.4.x 都是將核心載入到執行位址的起始位置加上 0x1000 bytes 的地方，如圖 8 所示，此為一段核心原始碼，裡面就註明了我們所使用的平台 FootBridge，在系統開始時，需要把核心解壓縮到 0x7c000000 位址。這些設定在進行核心的編譯前就需要完成，如此才能夠正確的啟動核心。

```
arch/arm/boot/compressed/head.S
-------------------------------------------------------
#elif defined(CONFIG_FOOTBRIDGE)
        .macro  loadsp,  rb
        mov \rb, #0x7c000000
        .endm
        .macro  writeb,  rb
```

圖 8. head.S 部分程式碼

■ Linux 核心

接著就是在核心中與平台相關的部分，以 Linux 2.4.26 為例，原始碼的結構大概如圖 8 所示，其中和平台相依性有關的主要為 ARCH、INCLUDE 和 DRIVERS 中的一部份檔案，這些部分會在編譯時依照不同的平台選項而選擇性的進行編譯。綜觀這幾個部分，其實就是涵蓋了系統的起始和基本架構的驅動(ARCH)、硬體基本資料的定義(INCLUDE)和硬體驅動程式(DRIVERS)三大項，而這也就是在做 Linux 移植時最為關鍵的三個部分。

綜觀以上兩點，我們可以發現要將 Linux 移植到一個新的平台上，最重要的事情就是將硬體規格與架構瞭解清楚，接著就是修改開機的程序使之能正確的將核心載入到記憶體中，而核心也需要依照平台的不同而做出相對應的修改，特別是對於不同處理器的規格和暫存器的使用方面，都需要重新做出規劃。最後就是依照其他附加硬體的不同而撰寫特定的驅動程式，雖然在大多數的狀況下驅動程式僅需要做出小部分的修改就可以在不同的平台上運行，但是依然有一些需要對系統底層做處理的驅動程式，如 IDE 驅動程式等，就需要依照不同平台而撰寫特殊的程式碼。

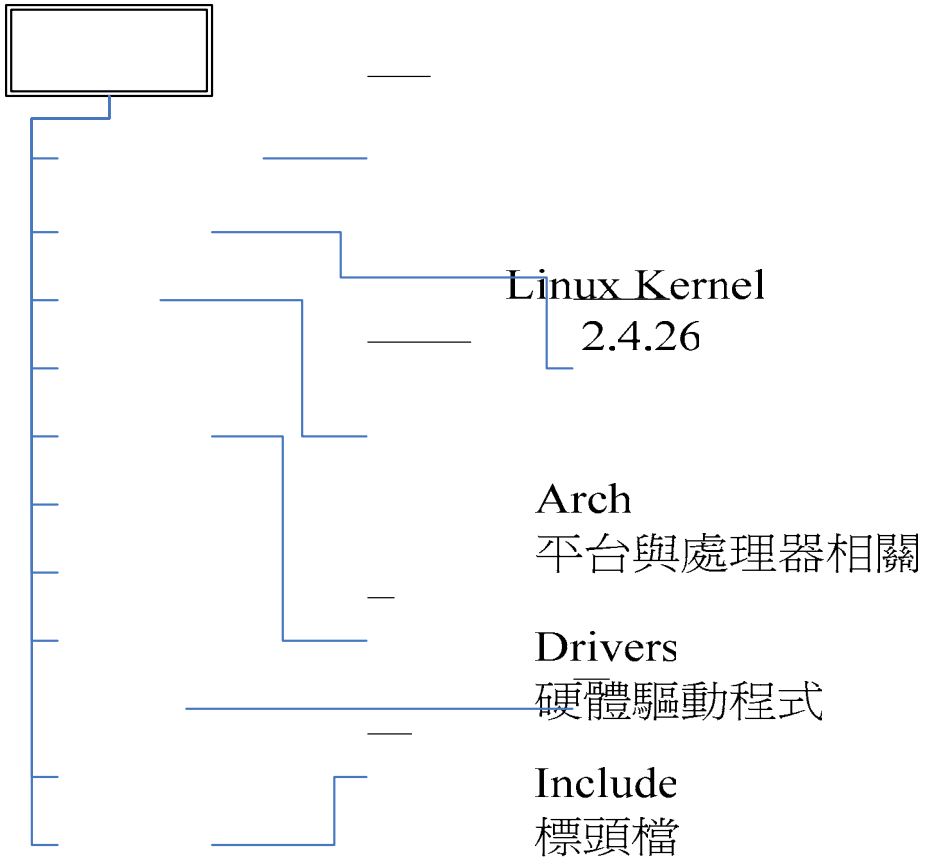

圖 8. Linux Kernel 2.4.26 目錄結構

5. 結論

在這次的報告中，我們主要嘗試安裝 Linux 到 ARM 平台，以方便我們進一步瞭解平台移植時所會遇到的問題。根據我們實驗及觀察編譯過程的結果，發現要移植平台，主要的三個動作為：**編譯環境的建置**、**開機程式的處理**以及**系統環境的設定**。編譯環境的建置僅在於由一台空置的機器開始建構系統，則需要利用 cross-compile 的方式在其他的系統上建置相關的程式。開機程式，則是負責把核心載入到記憶體的正確位置，在不同的平台和處理器上，可能都會有著不同的設定。系統環境的設定，則是依照不同平台的特殊規格，選擇適當的驅動程式模組和核心類型。

在完成修改，並移除部分不必要的工具和文件之後，我們所做出的包含核心和 Shell、Busybox、Dynamic Library 的 Image 大小約為 [illegible] 左右，而在能

達到相同功能的設定下，若使用原先的 Library 和傳統的工具集，則大約需要 6.2MB 左右或更多的空間，可以說是節省了相當多的空間。

同時，我們也可以藉由觀察 Kernel 原始碼的分層結構來瞭解，Linux 是一個具備良好架構的系統，除了部分基礎驅動程式和核心、標頭檔以外，其餘的部分都採用了相同的介面，因此就算使用不同廠商的匯流排控制晶片，如 PCI，也能使得子節點裝置驅動程式，如乙太網路卡的驅動程式，具有和平台無關的特性。整個移植過程裡，最困難的地方在於是否熟悉硬體架構，因為許多和平台相依的部分都是針對硬體本身的特性和運作方式來設計，因此若想要踏入移植這個領域，勢必要在瞭解硬體的特性和運作方式上下功夫。

6. 參考資料

[1] Linux 嵌入式系統減肥實作, 梁元彪、林盈達 2002

http://speed.cis.nctu.edu.tw/~ydlin/miscpub/embedlinux_tanet.pdf

[2] 嵌入式作業系統, 曹樂淇、曹世強、林盈達 2003

http://speed.cis.nctu.edu.tw/~ydlin/miscpub/survey_embOS.pdf

[3] Embedded Linux Survey, Blue Mug, Inc. 2002

http://www.bluemug.com/research/els/els.pdf

[4] The Linux 2.4 Kernel's Startup Procedure, Bill Gatliff 2003

http://billgatliff.com/articles/emb-linux/startup.html/index.html

[5] uClibc.org

http://www.uclibc.org

[6] ARM Linux

http://www.arm.linux.org.uk