An Application-Oriented Linux Kernel Customization for Embedded Systems

李啟泰　　洪振偉　　謝享金　　林志敏
逢甲大學資訊工程學系
台灣省臺中市西屯區
jimmy@fcu.edu.tw

摘要

近來，對於如何修改一個現有的一般性作業系統(General Purpose OS，GPOS)，而成為一個嵌入式作業系統已是一個令人注意的議題。而在現有的作業系統之中，Linux 則是其中一個普遍的一般性作業系統。雖然 Linux 系統提供了使用者可以自行藉由新增/移除功能選項來選擇自己所需要的功能，然而根據特定用途的嵌入式系統而言，Linux 核心原始碼仍然存在許多不必要的程式碼，而目前仍然缺少一個有效的方法來精簡 Linux 核心，因為經過如此修改後的 Linux 系統仍然是一個一般性作業系統，而不是我們希望得到的嵌入式系統。因此，在本文中，我們提出了一個以結合核心與應用程式一起考量為基礎，並運用”呼叫圖(call graph)”的方法，利用呼叫圖將目標系統的結構以圖形化做表現，並且找出一個可以移除多餘程式碼並且移入嵌入式系統的一個規則，為了要驗証我們所提出的方法，在文章的最後我們也製作出了一個實驗系統來証明我們提出的方法是可行而且有效的。

關鍵詞：嵌入式作業系統、Linux 核心、呼叫圖

一、導論

在嵌入式系統的研究中，如何取得一個嵌入式作業系統是一個研究的議題。嵌入式系統的廠商可能利用向嵌入式作業系統廠商購買的方式，如 WinCE 或是 Palm，此兩種作業系統目前已經廣為應用於 PDA 的產品中，然而這種方式最大的缺點在於許多資源或是技術支援必須與提供廠商合作，若是遇到所購買的作業系統無法完全符合嵌入式系統的規格時，必須額外付出費用來要求作業系統供應商來修改作業系統。另一種方式則是重新開發專用的嵌入式作業系統，然而這樣的方式在軟體開發上已經被認為是不經濟且不具有彈性的作法。

因此，目前有第三種作法為修改現有的一般性作業系統(General Purpose OS，GPOS)來取得一個嵌入式作業系統。然而這樣的 GPOS 必須具有方便取得原始碼以及符合開放式發展環境的特性。Linux 是一個能夠符合這些特性的 GPOS，並且逐漸被應用為嵌入式作業系統，最著名的例子為 IBM 的 Linux watch [1]。Linux watch 中重用了 Linux 中的功能並且移除了其餘的功能後，將之移入嵌入式系統上。一般而，Linux 對於嵌入式系統的開發中提供下列的優點：

- Linux 的原始碼可以免費取得。
- Linux 本身是 GPOS，因此提供許多的功能，這些功能涵蓋支援多種微處理機架構(如 ARM)、圖形處理、網路通訊、以及各式硬體裝置等。
- Linux 已經成為一個普遍的標準，因此能夠廣泛地取得各種技術支援以及相關說明文件。
- Linux 經過長期的演進以及改進，因此擁有很高的可靠度。

然而，利用 Linux 作為嵌入式作業系統的相關研究以及專案的經驗上，也歸納有以下的缺點，使得在作法上有其困難度：

- Linux 是一個很巨大而且複雜的 GPOS。以人工方式來手動地分析並且移除原始碼是一個既困難又花費甚巨的工程。
- Linux 擁有許多的版本。每個版本之間的差異性不易界定清楚，使得用上增加複雜度，如早期的版本可能不支援無線通訊的功能，但若是取用最新版本則可能增加太多額外的功能。
- 縮減後的 Linux 核心很難驗證以及測試正確性以及完整性。
- 目前缺乏方便且有效率的方式來縮減 Linux。Linux 雖提供了使用者三個指令—”make config”、”make menuconfig”以及”make xconfig”，讓使用者可以去選擇

自己所需要的核心功能。然而，經由這三個指令縮減後的 Linux 核心仍然是 GPOS，而非嵌入式作業系統。

- Linux 雖提供使用者藉由"新增/移除"等功能選項來選擇使用者想要保留的模組，不過這個方法仍然佔用了一些儲存的空間。而這也不適用於一個嵌入式 作業系統。

目前大多數開發方式仍舊以人工的方式去分析 Linux 原始碼，並經由移除不必要的程式碼來取得一個嵌入式 Linux，這種方式已被認為最安全及有較高的完整性，然而卻也有很低的彈性，意即這一個嵌入式 Linux 無法重複滿足不同的嵌入式系統的需求，所以必須有一個良好的方法來表現出 Linux 原始碼的結構並且配合工具，以自動或半自動的方式來縮減 Linux。因為客制化之後的嵌入式系統核心與應用系統的種類以及程式設計密切相關，因此在本文中，我們提出了一個以結合核心與應用程式一併考量為基礎，並運用"呼叫圖(call graph)"的方法[2,3,4,5] 的技術，來表現出 Linux 核心的結構，並提出一個步驟式的方法來減縮 Linux，來成為符合某特定應用的作業系統。Call graph 經常用在軟體重用工程或是軟體維護上，利用 call graph 可以將程式中的程序(procedure)彼此之間的相互關係以一個樹狀圖的方式表現。而根據 call graph，我們就可以從一個軟體系統內尋找出可重用的元件[6,7]。本研究中，利用 call graph 來描述 Linux 系統三個部分的結構，分別為"應用軟體"、系統函式庫(system library)"、和"Linux kernel"。藉由 call graph 所表現的樹狀圖包括了軟體及硬體的結構，也能夠更精準的發現不會使用到的程序，我們也就可以從 Linux 將這個程序移除。最後，就可以獲得一個規模較小且針對應用程式等特殊用途的 Linux。

為了驗證本方法，我們製作了一個實驗系統，在此實驗中假設要製作一台 CD 播發機，為此而重用 Linux 中所提供的 CD 播放軟體—ACDC [8]。由於 ACDC 本身具有圖化使用者操作介面，並且原先也支援滑鼠以及鍵盤等輸入輸出裝置，因此為了能夠移除這些不必要的部分，本實驗中應用了本研究的方法，由 ACDC 這個應用程式，開始表現出整個樹狀結構，並經由分析來移除不必要的部分，以求取得一個精簡的 Linux 核心來作為嵌入式作業系統，根據此實驗將列出相關的實驗數據，在實驗數據中將比較加上 Linux 原先所提供的"make config"時，kernel 精簡的百分比，由數據中可以看出利用"make config"指令後再加上本研究的方法能夠取得一個精簡的 Linux 核心。

本文之後的架構如下：第二節中將詳細介紹本研究主要的重點—一個步驟式的 Linux 核心縮減方法，此方法總共包含有七個步驟，在此章節中將詳細描述每個步驟。第三節中，則介紹一個實驗系統，名稱為 ACDC 播放器，此實驗系統的完成將完全依照七個步驟而逐步產生，並且將整理相關數據來證明本研究確實能夠有效地縮減 Linux 核心的大小。最後，在第四節中將為本研究作一個簡單的結論。

二、使用呼叫圖技術來縮減 Linux 核心

Linux 核心是一個龐大的系統，它包含了許多的程序，而這些程序也藉彼此之間的呼叫來合作執行使用者想要進行的工作。而由於 Linux 核心並沒有一個固定的結構，因此對於一個程式設計者來說，預先設計出一個 Linux 核心是非常複雜的一件事。因此縮減一個 Linux 核心所會遭遇到的第一個問題就是必須了解 Linux 核心的結構，以降低縮減核心過程所遭遇到的阻礙。而呼叫圖可以解決這個問題。呼叫圖可以將程式中的程序彼此之間的相互關係以一個樹狀圖來表示，意即將一個複雜的 Linux 核心表現在一個樹狀圖上，程式設計者可以很明確的由此樹狀圖來了解當一個工作發生時，Linux 核心內的程的呼叫順序，藉由分析呼叫關係去歸納出一個移除不必要的程式碼的法則。

呼叫圖的技術常應用於軟體維護(software maintenance)與軟體再用(software reuse)的研究中，在軟體維護的研究中，一個現有的程式可以利用呼叫圖來表現其程式結構，並且依循法則來界定出不必要的程式碼或是可能產生錯誤的程式碼，並加以移除來減少記憶體空間的浪費、加快程式執行或是讓該程式提供安全以及可靠度。在此樹狀圖中，可能是以一行的程式碼或是程序作為樹狀圖的節點，一般而言必須以程式語言的特性作為決定。以 C 程式語言為例，main()是主要的程式進入點，因此用以作為樹狀圖的根節點(root)，而在 main()所程式片段中所呼叫的程序就是其餘的節點，藉由此樹狀圖可以表現出一個 C 語言程式的程序之間的呼叫結構。

然而，應用呼叫圖來表現出 Linux 核心的結構仍存在以下的問題：

- 近代的作業系統多以階層(layer)結構來

設計的(見圖 1)，因此 Linux 核心在 Linux 系統中不是唯一元件，而是處在 Linux 系統中的中間層部分。而在其它階層部分，像是應用軟體，函式庫以及裝置驅動程式等，都是和 Linux 核心相同可以被重用來移入嵌入式系統之上。

- Linux 核心是一個巨大而且具有模組化的一個元件，因此利用呼叫圖去對核心的每一行程式碼作描述是沒有意義且沒有效率的作法。
- Linux 核心中沒有一個作為主要進入點的程序，單只表現出核心的結構是無意義的，必須再搭配其他層的結構來加以分析，目前可以找到具有根節點的樹狀結構就是在 Linux 最上層的應用軟體，大多數的 Linux 應用程式是以 C 以及組合語言做為程式語言，C 語言能夠產生一個樹狀圖，只要分析應用程式以哪些節點呼叫核心，就可以產生一個完整的樹狀圖。

在本研究中，將提出一個步驟式的方法，在此方法中把縮減 Linux 核心的過程分成七個步驟：

- 從應用程式的原始碼建立出應用程式的呼叫圖
- 從函式庫的原始碼建立出一個函式庫的呼叫圖
- 由 Linux 原始碼建立一個核心的呼叫圖
- 確認有那些硬體裝置是嵌入式系統原本所有以及所須要的
- 結合應用程式呼叫圖、函式庫呼叫圖和核心呼叫圖，並找出應用程式所需要的系統呼叫
- 確定核心所需要的例外處理程序
- 移除不需要的程序並且測試精簡後的核心

從 2.1 節開始，將詳細依序說明這七個步驟

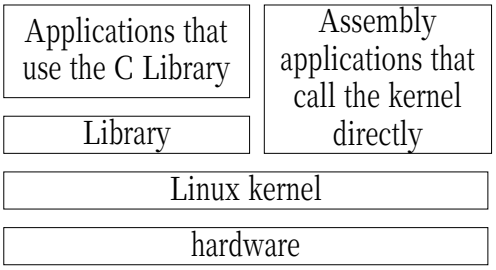

圖 1： Linux 系統的階層式結構

2.1 從應用程式的原始碼建立出應用程式的呼叫圖

Linux 作業系統的原始碼可以由網路上去免費取得，而且 Linux 含有多種應用程式，對於發展嵌入式系統的廠商或是專案來說，依據嵌入式系統的需求取用 Linux 所具有的特定應用軟體，製造加工成為他們所需要的嵌入式系統。

在縮減 Linux 核心過程中的第一步為將可以取得原始碼的應用程式建立出一個應用程式的呼叫圖。核心本身並沒有主要的進入點來作為呼叫圖的根節點，反之位於 Linux 最上層的應用軟體，因為多為 C 語言程式，因此可以找出由 main()當作根節點的樹狀圖，所以在第一步驟中即先建立出應用軟體的呼叫圖。

本文中，一個呼叫圖的定義如下：

定義：一個呼叫圖是具有方向性的，而一個程式中的呼叫圖通常會被定義成圖 $CG=(P, E, s)$，在圖 $CG=(P, E, s)$中，$P \cup s$ 表示圖形中節點(nodes)的集合(每個節點代表一個程式中的程序)，而圖形中 E 的部分則是用來表示程序之間彼此呼叫的關係，例如在所有 E 中的一個有向邊(p_1, p_2)表示唯一存在一個程序 1 去呼叫程序 2，有可能是只呼叫一次或是呼叫多次。在呼叫圖中，可以把程序 1 看成是程序 2 的上一個要執行的程序，而程序 2 則是接替程序 1 成為下一個要執行的程序。路徑(p_x, p_y)則是表示存在一個連接 p_x 到 p_y 的一個路徑。S 在呼叫圖 CG 中則是代表最初的一個節點，而這個節點也是呼叫圖的一個進入點，而且假使 p_i 屬於 P 即表示至少存在一個路徑(s, p_i)，也就是在 P 中的每一個程序都會被呼叫到。所有指向節點 n 的 E 的數目為進入節點 n 的度數(degree)，表示成 $in(n)$，而 $in(n)$所代表的意義為程序 n 的上一個執行的程式的數目，同理，所以在呼叫圖中被節點 n 所指向的節點代表接替程序 n 成為下一個執行的程序，表示成 $out(n)$。

根據定義，一個呼叫圖可以用來表示一個程式的靜態結構，如果以 C 語言來作為例子解釋，在一個 main()函數裡所有被呼到的程序的順序可以藉由呼叫圖中的路徑來表示，而 main()函數則是代表整個函數執行的起始節點(見圖 2(b))。SUCC(main)應該包含了在執行一項工作時所有會被呼叫到的程序，也就是如果有一個程序 P 是不包含在 SUCC(main)之內就表示程序 P 在程式中是不會被使用到的。因此，為了達到精簡的目的，程序 P 就必須被移除掉。而之所以會出現不被使用到的程序也許有可能是因為程式設計者所產生的錯誤或是因為考慮容錯時加入一些不常被執行的程式碼，這種情形在一個巨大的系統像是 Linux 系

統，更為常見。不過，即使移除了不須要的程序例如程序 P，並不會造成 Linux 系統在執行上的錯誤，但也不能保證完全的安全。在圖 2(b)中可以發現 e()和 f()這兩個程序並不包含在 main()函數之下，因此對於圖 2 這個例子來說，e()和 f()這兩個程序是可以被移除掉的，因為它們並不會被使用到。

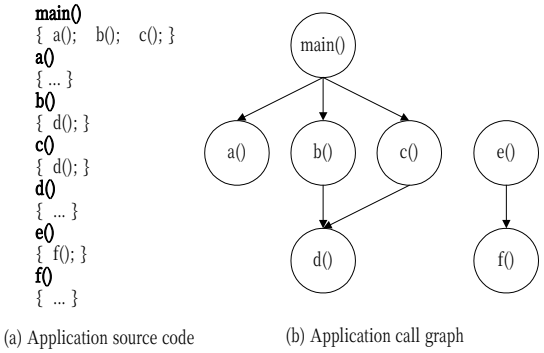

圖 2： 由應用程式原始碼產生一個呼叫圖

2.2 從函式庫的原始碼建立出一個函式庫的呼叫圖

應用程式位於 Linux 的最上層，基本上，在 Linux 系統裡執行的應用程式可以是用 C 語言所寫的，透過函式庫裡的函式呼叫對 Linux 核心提出要求作業系統服務的命令(見圖 3(a))。在 Linux 系統裡的函式庫的函式可以是輸出/輸入函式,，數學運算函式，以及字串函式等。然而對於一個嵌入式系統而言，函式庫裡存有許多不必要的函式，且這些函式勢必會佔用許多的儲存空間。

大多數的作法為重新製作一個新的函式庫來符合嵌入式系統。不過，從軟體重用的觀點，若能有效的重用這些函式呼叫，必定可以減少重新製作一個函式庫的時間。因此，本研究的第二個步驟就是建立出一個函式庫的呼叫圖，利用這個呼叫圖來表示函式庫中的呼叫結構。如同核心，函式庫的呼叫圖中並沒有一個根節點，也就是函式庫提供多個進入點讓上層的應用程式進入(圖 3(b))。

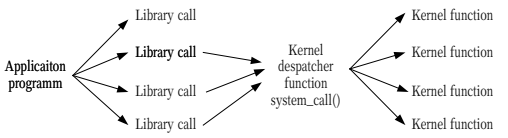

圖 3(a)： 經由函式庫呼叫去呼叫一系統服務

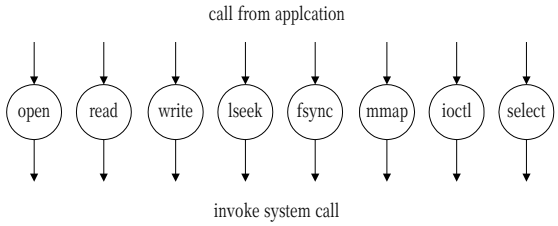

圖 3(b)： 簡易函式庫呼叫圖

2.3 由 Linux 原始碼建立一個核心的呼叫圖

在 Linux 階層式架構中，函式庫的下一層就是核心層，而這個核心層也是整個 Linux 系統最重要的部分，核心負責去控制以及協調在 Linux 系統內所產生的運算。也由於 Linux 核心所扮演的角色是如此的重要，因此，大部分發展嵌入式系統的廠商為了要確保核心能夠正確的執行，都傾向於保留原本的 Linux 核心部分而不去修改它。也因為這樣，許多不是必要的程式碼還是會存在於核心之內。

步驟 3 主要是為了分析 Linux 核心的呼叫結構。和函式庫的呼叫圖相同，核心層不只提供一個進入點，而且核心的呼叫結構對於核心層的下一個層級也支援了許多的進入點。圖 4 裡則是在描述應用程式，函式庫以及核心彼此之間的關係。

大部分近代的作業系統都是屬於中斷驅動(interrupt-driven)系統，在這類系統之下，一個事件需要透過觸發一個中斷或是產生一個例外(exception)來要求 CPU 的服務。而不論 CPU 在處理任何工作的時候，當系統一旦發生中斷或是例外處理時，CPU 都會交出控制權，然後核心就會針對這些中斷或是陷井作出相對應的處理。

為了要對 Linux 核心有更進一步的認識，我們必須要先去了解到核心在什麼時候會被執行，而下面就列出核心會被執行時的情況：

- 當 Linux 系統在開機的時候
- 當一個應用程式去呼叫核心的系統呼叫(system call)
- 如果有例外發生的時候，核心也會執行。在這個情況發生時，CPU 會暫停工作並將控制權交出來，直到核心把這個例外處理完畢後再交回控制權給 CPU。
- 當有硬體裝置對 CPU 傳送一個中斷訊號時，CPU 也會交出控制權，由核心去處理這個中斷訊號。

在第 2 個和第 3 個情況下，也就是當 Linux 系

統在開機時以及當有應用程式在呼叫一個系統呼叫，都屬於應用程式和 Linux 系統間的互動。而第 4 個情況則是屬於硬體裝置和 Linux 系統之間的互動。因此，為了要找出核心內可以被移除的程序，我們根據上述的 4 個情況來觀察核心呼叫圖內的程序其彼此之間的呼叫關係。

接下來可以藉由結合應用程式的呼叫圖，函式庫的呼叫圖以及核心的呼叫圖成為一個呼叫圖，藉由此樹狀圖能夠觀察由應用程式到核心的互動，並由呼叫路徑找出所有會被使用到的程序。如此一來，不在這個路徑的程序可能就是可以被移除的程序。

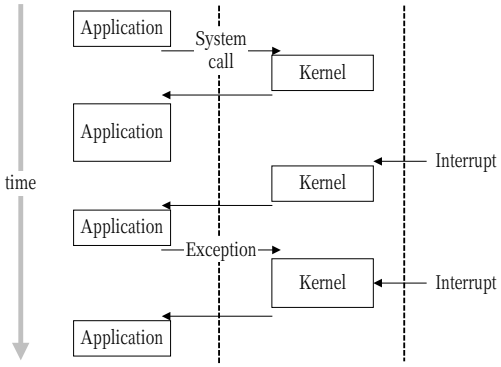

圖 4： Linux 核心中應用程式與核心之間的控制關係

2.4 確認有那些硬體裝置是嵌入式系統原本所有以及所須要的

雖然 Linux 系統支援許多的硬體裝置，例如硬碟、鍵盤、滑鼠等等，然而對於嵌入式系統來說，有些周邊裝置或許是不必要的。例如用於 PDA 上的嵌入式作業系統，鍵盤以及滑鼠並不是標準配備，因此這兩種硬體裝置的驅動程式就是不必要而可以移除的。因此，利用 Linux 作為嵌入式作業系統時，可能存在一些不會使用到的裝置驅動程式或是相關的程式碼，尤其是大多數的專案或是嵌入式系統廠商趨向於完全保留 Linux 核心，此種作法勢必存在更多不必要的程式碼。

Linux 系統提供透過選項的設定來選取所需要的驅動式而不用安裝完整的系統，再加上裝置驅動程式可以不必經由使用者安裝即可使用，然而還是會有許多相關的資料和程式碼存在於核心之內。而這些存在於核心內的相關資料可能佔用嵌入式系統的資源，第 4 步驟的目的希望移除這些不會使用到的部分。
首先必須分析以及觀察 Linux 核心和硬體之間的互動情形：

- 當 Linux 系統在開機時，系統會初始化並且去偵察周邊裝置。但是對於一個嵌入式系統來說，這個動作通常是不需要的。例如一個網路攝影機並沒有螢幕、滑鼠、鍵盤等硬體裝置，可是 Linux 系統仍然把這些裝置相關的程式碼保留下來並且載入這些不會使用到的硬體裝置的驅動程式，如此除了會佔用到資源外，也會佔用多餘的儲存空間。
- 另一個情形則是在移除不會使用到的驅動程式。移除的原則則是普遍被企業及學術界研究所公認的。大部分的企業都寧可完全移除掉所有原本 Linux 系統內的驅動程式並且發展自行設計且製造的驅動程式。

雖然 Linux 系統提供了使用者一個指令—make config，可以讓使用者自行新增/移除硬體程式碼，也許這個步驟對於有經驗的工程人員來說是一個有用的工具，可是對於經驗不多的使用者來說，這個步驟的複雜度就會大大的提升。

Linux 核心提供了位於階層結構中最頂層的應用程式不同的介面，使應用程式可以去使用硬體裝置。這些介面包括了像是 *Iseek*、*read*、*write*、*readdir*、*select*、*ioctl*、*mmap*、*release*、*fsync*、*fasync*、*check_media_change* 以及 *revalidate*，我們則希望可以找出不需要的介面並且從核心中移除以達到精簡 Linux 核心的目的。

2.5 結合應用程式呼叫圖、函式庫呼叫圖和核心呼叫圖，並找出應用程式所需要的系統呼叫

從步驟 5 到步驟 7，我們希望找出原本存在於核心內並且可以重用成為應用程式所需要的所有程序。步驟 5 將函式庫的呼叫圖加上應用程式的呼叫圖，如此一來，應用程式呼叫圖裡的根節點(main()函數)就可以成為結合後的呼叫圖裡的根節點。再從此結合的呼叫圖去比對出不會使用到的函式庫的呼叫並加以移除，這樣就可以得到一個精簡過的函式庫。然而，一個函式庫呼叫是作為連結應用程式和 Linux 核心的一個通道，因此，同上述步驟，我們將核心的呼叫圖和函式庫的呼叫圖結合起來並且找出有那些系統呼叫會和函式庫的呼叫產生互動，進而找到不需要的系統呼叫。

2.6 確定核心所需要的例外處理程序

步驟 6 的目的為移除掉不必要的例外處

理程序的程式碼，因為並不是所有 Linux 核心所提供的例外處理序對於嵌入式系統都是必要的。而在版本 1.2.3 的 Linux 系統中大約有 18 個例外處理程序，包括了 divide_error、debug、nmi、int3、overflow、bounds、invalid_op、device_not_available、double_fault、coprocessor_segment_overrun、invalid_TSS、segment_not_present、stack_segment、general_protection、page_fault、coprocessor_error 以及 alignment_chect。

2.7 移除不需要的程序並且測試精簡後的核心

在找出所有在應用程式裡不需要的程序之後，步驟 7 則是將它們從 Linux 核心內移除來實現精簡 Linux 核心。但是在移除掉不需要的程序之後，還是必須去測試精簡後的 Linux 核心是否能正確無誤的執行，如果這個新的核心會產生錯誤的話，我們就需要去核對這個核心的呼叫圖找出錯誤所在，再重新建立出一個正確的呼叫圖。

三、實驗系統：ACDC 播放器

為了要驗証前面我們所提出縮減 Linux 的方法，本研究製作了一個實驗系統。我們利用 Linux 中所提供的 CD 播放軟體—ACDC，並將其修改成為嵌入式應用程式。我們首先假設有一間公司決定要重用 ACDC 這個軟體將其應用為嵌入式作業系統再移至一個硬體裝置來使用。最後產生出來的產品，我們稱之為 ACDC 播放器。

3.1 ACDC 的縮減

首先，我們先把對於這個實驗系統的處理過程概括的區分為三個方面：

- 一開始的步驟將建立出包括應用程式，函式庫，核心這三個部分的呼叫圖。
- 接下來則是要尋找出不會使用到的程式碼、程序以及驅動程式。
- 最後則是把不需要的部分例如程序等給移除掉，並且去測試經過縮減過後的核心。

因此，為了要去重用以及縮減 ACDC 這個軟體，我們首先要去建立出相關的呼叫圖包括應用程式，函式庫以及核心的呼叫圖，然後根據前面所提的步驟 5，我們將應用程式，函式庫以及核心的三個呼叫圖結合起來並且去找出所有在 ACDC 播放器裡所需要的程序的集合。

在這個實驗系統裡我們所使用的實驗環境如下：

- Linux：Slackware 3.0
- 核心的版本：1.2.3
- 函式庫：libc 4.6.27
- 目標應用程式：文字型式的 ACDC

版本 1.2.3 的 Linux 核心之所以被我們採用的原因在於它的簡易性，因為這個版本的核心並沒有支援太過複雜的功能。對於一個嵌入式系統的領域來說，這些太過複雜的功能通常也是不適用的。由於這個原因，1.2.3 版本的 Linux 核心結構也比較容易去分析。在這個版所提供的 ACDC 軟體為一個文字型式的應用程式。

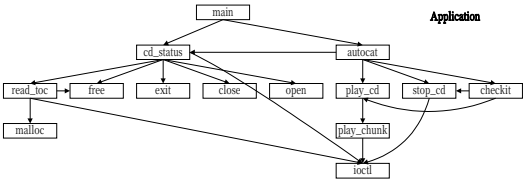

圖 5： ACDC 的呼叫圖

對於一個 CD 播放器來說，這種型式的軟體並不需要圖形式使用者介面，所以利用文字型式的 ACDC 軟體可以去簡化我們的論証過程。

為了要去找出 ACDC 播放器所需要的程序，首先我們必須先建立出 ACDC 的呼叫圖。圖 5 展示了較簡化的 ACDC 呼叫圖。因為 ACDC 播放器並不需要從鍵盤去輸入資料以及螢幕，所以我們將使用者介面移除。由於我們已事先移除許多不需要的程序，因此圖 5 的呼叫圖是已經經過縮減了。至於須要被移除的程序就要根據目標系統的需要而定了。

可以發現，一部分在圖 5 中的程序都是由 *libc* 所提供，*libc* 在 Linux 系統是一個常見的函式庫，而不管在什麼時候，當 Linux 應用程式向此函式庫要求服務時，這些程序也被視為進入點。因此，接下來我們就是要建立出 *libc* 這個函式庫的呼叫圖，圖 6 中灰色欄位部分就是我們所建立出的 *libc* 函式庫呼叫圖，再根據這個呼叫圖，我們就可以將不會使用到的函式庫的呼叫移除掉。

在 *libc* 呼叫圖中，我們可以很明顯的發現，*malloc*，*free*，*exit*，*close*，*open* 及 *ioctl* 這 6 個 *libc* 的呼叫對於 ACDC 和 *libc* 函式庫之間的互動是必要存在的，因此它們不能夠被移除，因此，其它 *libc* 的呼叫就可以被移除進而達到精簡函式庫的目的了。

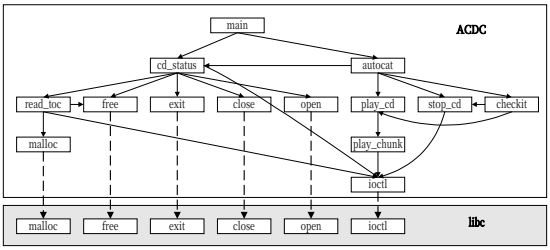

圖 6： ACDC 呼叫圖及 libc 呼叫圖之結合

接下來則是把圖 6 中的呼叫圖和核心的呼叫圖再作一次結合，結合後的呼叫圖如圖 7 所示。可以發現，核心提供了 5 個系統呼叫來作為核心部分的進入點，而這五個系統呼叫分別為 *mmap*、*exit*、*open*、*close* 以及 *ioctl*。這些系統呼叫也都有它們自己本身需要去呼叫的程序，最後 Linux 核心就可以經由這五個系統呼叫彼此之間的合作來服務硬體部分的要求。而不包含在這 5 個系統呼叫及其需要的程序內的其它部分就可以移除了。

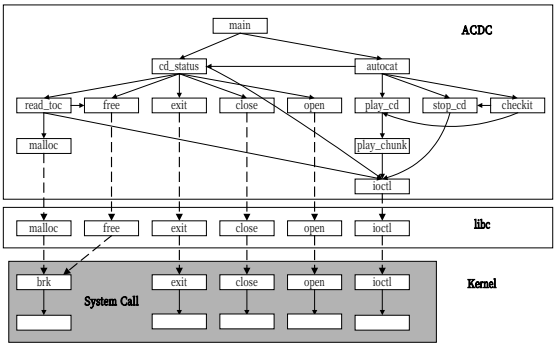

圖 7： 將圖 6 和核心呼叫圖結合後的呼叫圖

3.2 實驗結果

由於 Linux 的核心可以被分解成不同的部分並且儲存於個別的資料目錄中，再加上 Linux 提供了使用者"make config"這個指令來新增/移除裝置的驅動程式，因此，在下面的討論中我們只針對檔案*.o 的部分來研究。而檔案*.o 也是主要處理像是程序管理、記憶體管理及系統呼叫的叫用等計算工作。

為了要突顯出利用呼叫圖方式來縮減 Linux 核心的效率，於是我們作了兩組實驗數據用以比較：

實驗組 1：單獨利用呼叫圖方式來縮減的 Linux 其核心簡化的數據。

實驗組 2：混合著使用 Linux 系統所提供的指令─config，以及呼叫圖方式去修改 Linux 核心，並藉此觀察是否可以得到規模比實驗組 1 還要小的 Linux 核心。

從表 1 可以發現經由實驗組 1 所縮減後的 Linux 核心各個部分大約從原本核心裡移除了 13.9%的程序，被移除掉所佔比例最大的部分在於核心.o，而這個部分同時也是 Linux 核心內的中心角色。從表 1 的實驗數據裡可以發現我們總計移除了大約 30.5%的程序，而這個實驗數據也証明了我們利用呼叫圖方式的確可以有效的縮減 Linux 核心的規模。

表 1： 實驗組 1 的實驗數據

Linux kernel's organization	**Original size (byte)**	**call graph approach (byte)**	**Percentage**
arch/i386/kernel/head.o	60,617	60,617	0 %
init/main.o	8,314	6,598	20.6 %
init/version.o	639	639	0 %
arch/i386/kernel/kernel.o	43,339	32,286	25.5 %
kernel/kernel.o	73,000	50,730	30.5 %
arch/i386/mm/mm.o	3,767	3,714	1.4 %
mm/mm.o	37,079	32,336	12.8 %
fs/fs.o	92,121	71,006	22.9 %
net/net.o	117,098	115,606	1.3 %
ipc/ipc.o	22,472	21,355	5 %

在實驗組 2 裡，我們混和著使用 Linux 所提供的"config"指令和呼叫圖來縮減一個 Linux 核心，所得到的實驗數據如表 2 所示。在進行實驗組 2 的修改過程裡，一開始我們先利用"config"指令，藉由功能選項去安裝我們所想獲得的核心功能，接下來則是建構出呼叫圖並對核心進行更進一步的縮減，從表 2 可以發現到我們總計約從核心裡移除了 20.7%的程序。

表 2： 實驗組 2 的實驗數據

Linux kernel's organization	**Phase 1**	**Phase 2**	**Percentage**
arch/i386/kernel/head.o	60,617	60,617	0 %

init/main.o	8,314	6,598	20.6 %
init/version.o	639	639	0 %
arch/i386/kernel/kernel.o	36,710	32,478	11.5 %
kernel/kernel.o	58,054	46,288	20.7 %
arch/i386/mm/mm.o	3,660	3,660	0 %
mm/mm.o	32,336	32,336	0 %
fs/fs.o	70,567	70,567	0 %
net/net.o	15,311	15,311	0 %
ipc/ipc.o	221	221	0 %

四、結論

修改現有的Linux成為取得一個嵌入式作業系統對於廠商而言是一個符合經濟效益考量及具彈性的作法，相對於逐行縮減 Linux 核心程式，本研究所提出的方法利用呼叫圖技術可以適用於各種不同特定應用，而這一點對於嵌入式系統設計者來說是很有用的。另外，我們藉由製作 ACDC 播放器的實驗系統來驗証我們使用的方法是可行而且有效的。而透過最後的實驗數據可以發現利用呼叫圖方法來修改一個 Linux 核心的確可以移除約 20.5%的核心大小而且不會產生副作用或是發生錯誤。然而本研究並無法保証縮減過後的核心是最小的。因此，希望在未來的研究中，我們會列出對於現存的 Linux 系統中可以進行縮減的部分，進而能夠提出對於如何去產生一個小規模的Linux核心以及如何有效去監控核心執行效率之間的平衡點。

參考文獻

[1] http://www.ibm.com/products/gallery/linuxwatch.shtml

[2] Hecht, M. S., *Flow analysis of Computer Programs*, North-Holland, New York, 1977.

[3] B. Ryder, "Constructing the call graph of a program," *IEEE Trans. Software Eng., vol. SE-5*, pp. 216-225, May 1979.

[4] J. P. Banning, "An Efficient way to find the side effects of procedure calls and the aliases of variables," *Proceedings of the 6th Annual Symposium on Principles of Programming Languages*, ACM, 1979, pp. 29-41.

[5] Callahan, D., Carle, A., Hall, M.W., and Kernedy, K., "Constructing the procedure call multigraph," *IEEE Transaction on Software Engineering*, SE-16(4):483-487, April 1990.

[6] A. Cimitile and G. Visaggio, "Software Salvaging and the Call Dominance Tree," *The Journal of System and Software* 28, 1995, pp. 117-127.

[7] Elizabeth Burd and Malcolm Munro, "A Method for the Identification of Reusable Units Through the Reengineering of Legacy Code," *The Journal of System and Software* 44, 1998, pp. 121-134.

[8] http://www.hitsquad.com/smm/linux/CD_PLAYERS/