Linux 與 BSD 核心的比較:以各種角度去探討

羅榮鐘 林盈達

國立交通大學資訊科學系

新竹市大學路 1001 號

TEL：（03）5712121 EXT. 56667

E-MAIL：{ccluo, ydlin}@cis.nctu.edu.tw

摘要

Linux 與 FreeBSD 是兩套 Unix-Like 中應用最為普及的作業系統，兩者間的處理效率、穩定度、商業價值以及便利性的取捨是一個很好的研究議題。這些問題可以從各種角度下去分析，其中包含了 Unix 的發展歷史、設計理念、管理的方式、外部的物件、內部的方法等。另外，我們還選擇 Linux 內建的 IPtable 以及 FreeBSD 內建的 IPFilter 來進行封包傳遞的能力與效率測試。經過比較，我們發現 Linux 與 BSD 在訂定規格時，分別採用了 POSIX 與 BSD 的規範，以致於在實作時有不同的設計。另外 Linux 在封包傳遞的能力與效率上明顯的比 BSD 來的遜色，雖然如此 Linux 在 OpenSource 所獲得的資源卻遠超過 BSD 的所得。

關鍵字：核心比較，IPtable，IPFilter，設計理念，封包傳遞

1. 簡介

隨著 Linux 與 BSD 的發展，這兩套 Unix-like 的作業系統漸漸被眾多的使用者加以實用，也因此常常會有人在面對這兩者的時候會有不知如何取捨的狀況發生。大多數的人往往會選擇其中一套作業系統，熟悉之後就不再更換，往往對於另一套的特性不熟悉或者選擇了效率上較差的一方。這些狀況通常可以歸納為以下幾種：

狀況一：新手上路。以一個正在 CS 修業的學生來說，剛接觸這一類的作業系統的時候，常常會有一個重要的選擇題:選擇 Linux 或者 BSD。對於一個新接觸

Unix-Like 作業系統的使用者，最為困擾的問題就是選擇哪一個較適合的問題。這其中主要的問題是在便利性、管理的方式以及穩定度等。

狀況二：版本移植。對一個程式設計師而言，若要將 Linux 或 BSD 其中一套中所開發出的軟體移植到另外一套中正確執行的時候，必定會發生一些問題。這其中主要的問題可以從派別的分野、外部的設計物件、內部的設計方法等設計理念上來找到差異。

狀況三：商業價值。對一個商人而言，考慮的問題不外乎是產品的成本、效率；也因此在做選擇的時候，就是以商業價值當做優先順序。

綜合以上我們可以發現效率、穩定度、商業價值以及便利性的取捨問題。對於這些問題我們可以從各種角度下去分析，這其中包含了 Unix 的發展歷史、派別的分野、設計理念、管理的方式、外部的設計物件、內部的設計方法以及其中的應用程式的效率。因此，我們從管理者、程式設計師的角度來分析，並且從現行的版本上進行效率測試來比較。

2. 以管理者的角度比較

TCP/IP	LINUX	BSD
Network IP configuration	/etc/sysconfig/network-scripts/	/etc/rc.conf
Hosts IP addresses	/etc/hosts	/etc/hosts
Name service switch	/etc/nsswitch.conf	/etc/host.conf
Network parameters	sysctl -a | grep net	sysctl
Routing daemon	routed	routed
NIC Configurations	ifconfig -a	ifconfig -a
Secondary IP Address	modprobe ip_alias ifconfig eth0:1 IP	ifconfig x10 alias IP
Login prompt	/etc/issue	
Increase the # of pseudo-terminals	cd /dev ./MAKEDEV -v pty	
Maximum # of ptys	256	
Remote Shell	rsh	rsh
YP/NIS service binder	/sbin/ypbind	/usr/sbin/ypbind

表 1：部分常用的管理指令與檔案位置

以一個管理者或一般使用者的角度來說，其最為關心的內容就是如何使用作業系統所提供的功能。在 Linux 與 BSD 這兩套作業系統中，由於都是以 Unix 為基礎所發展而成的作業系統，因此在很多地方非常的相似，甚至是沿用 Unix 原始的設計方法，然而在某些部份卻依然有不同的設計。如表 1 所示，我們會發現 Linux 與 BSD 在一些地方是一樣的，而某些地方雖然不同但是卻相似。會出現這樣的狀況主要是因為 Unix 的發展過程中出現的兩大派別 AT&T 公司的 System V 以及 Berkeley 大學的 BSD，由於本篇重點不在於此，因此僅以小篇幅略微一提。

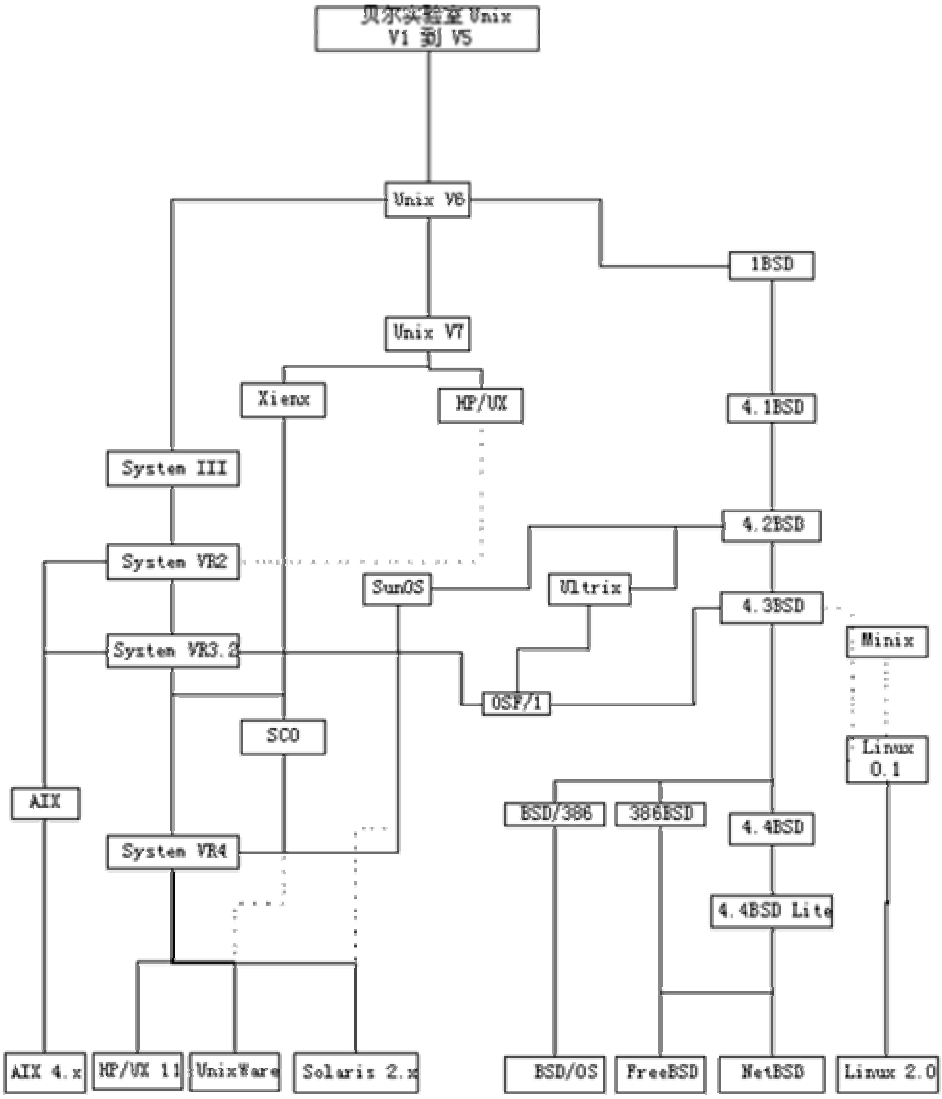
2.1 歷史

Unix 的第一個版本是 1969 年由 Ken Thompson 在 AT& T 貝爾實驗室實現的。到了 70 年代末，AT&T 成立 Unix 系統實驗室來繼續發展 Unix。幾乎在這個同時，Berkeley 大學電腦系統研究小組（CSRG）使用 Unix 對操作系統進行研究，大部分原有的程式都被重新寫過，因此 CSRG 中的研究人員把他們的 Unix 組成一個完整的 Unix 系統──BSD Unix（Berkeley Software Distribution），向外發行。這個 BSD 是 Unix 的一個分支，他的發展對於 Unix 有相當大的影響。而 AT&T 的 Unix 系統實驗室，同時也在不斷改進他們的 Unix 版本，直到他們吸收了 BSD Unix 中已有的各種先進特性，推出了 Unix System V 版本。從此以後，BSD Unix 和 Unix System V 形成了當今 Unix 的兩大主流。

雖然 AT&T 的 Unix System V 也是非常優秀的 Unix 版本，但是 BSD Unix 在 Unix 領域內的影響更大。AT&T 的 Unix 系統實驗室一直關注著 BSD 的發展，在 1992 年，Unix 系統實驗室指控 BSDI── 一家發行商業 BSD Unix 的公司，違反了 AT&T 的許可權。因此 CSRG 就對他們最新的 4.4BSD 進行了修改，刪除了那些來自於 AT&T 的原始碼，發布了 4.4 BSD Lite 版本（該系統是不完整的，尤其對於英特爾 386 體系的計算機系統）。386BSD 計劃由 Bill Jolitz 等研究人員發起，將 4.3BSD Net/2 移植到 80386 平台上。1993 年 12 月對於 FreeBSD 來講是非常重要的日子，FreeBSD 1.0 版本於這個月正式發布。然而 BSD 與 AT&T 的法律糾紛仍然威脅著 FreeBSD 系統的合法性。包括 FreeBSD 在內，都被要求立即轉換到 4.4 BSD Lite 上去。雖然 4.4 BSD Lite 只刪除了一小部分代碼，但尤其對於英特爾 80386 平台，缺乏這些代碼，系統就不能正常運轉。因此直到 1995 年 1 月他們才發布了 FreeBSD 2.0，這次就是一個完全的 4.4BSD Lite 的系統了。

而我們另外一個主角是有個芬蘭郝爾辛基大學的 Linus Torvalds—Linux 的

創立者，他有心的讀取 Unix 的核心，並且去除較為繁複的核心程序，將它改寫成可以適用於一般個人電腦的 x86 系統上面。到了 1991 年，他終於將 0.02 版的 Linux 放到網路上面供大家下載，並且由於受到大家的肯定，相當多的朋友一起投入這個工作中。終於到了 1994 年將第一個完整的核心 Version 1.0 釋出。整個發展的過程可以由圖一表示。



圖一：Unix 的發展過程(虛線表示並非繼承原來的版本而是獨立開發或參考)

2.2 影響

我們說完了歷史，但究竟這些歷史跟我們想討論的有什麼關係呢？其實是有的。我們可以從圖一發現 Linux 與 BSD 都是由 4.3BSD 發展而來(虛線表示並非繼承原來的版本而是獨立或參考開發)，因此 Linux 與 BSD 繼承了許多 4.3BSD 或整個 Unix 的特性，例如核心是集成式的作業系統核心 (monolithic kernel)。它

們將行程管理，記憶體管理和檔案系統包在一起，成為一個單一的可執行檔，而週邊硬體裝置管理則另外分開，成為一組驅動程式，每一個驅動程式的目的是控制某一類型的硬體裝置，例如控制軟碟機讀取磁片。這種設計是為了降低核心更動的頻率，不必為了新硬體裝置更改核心，而且驅動程式也比較好寫。但是 Linux 核心的進步非常迅速，這種設計反而不利於核心的實驗更新，為了克服這個缺點，Linux 提出模組 (module) 機制，這是種軟體容器，它和核心的介面要比傳統 Unix 的驅動程式來的有彈性，可以用來提供新功能給核心，當然也適用於寫驅動程式。相同的，BSD 也採用了模組 (module) 機制。對於一個管理者或者使用者來說，Linux 與 BSD 有許多管理的指令是一樣的（繼承了 Unix）。

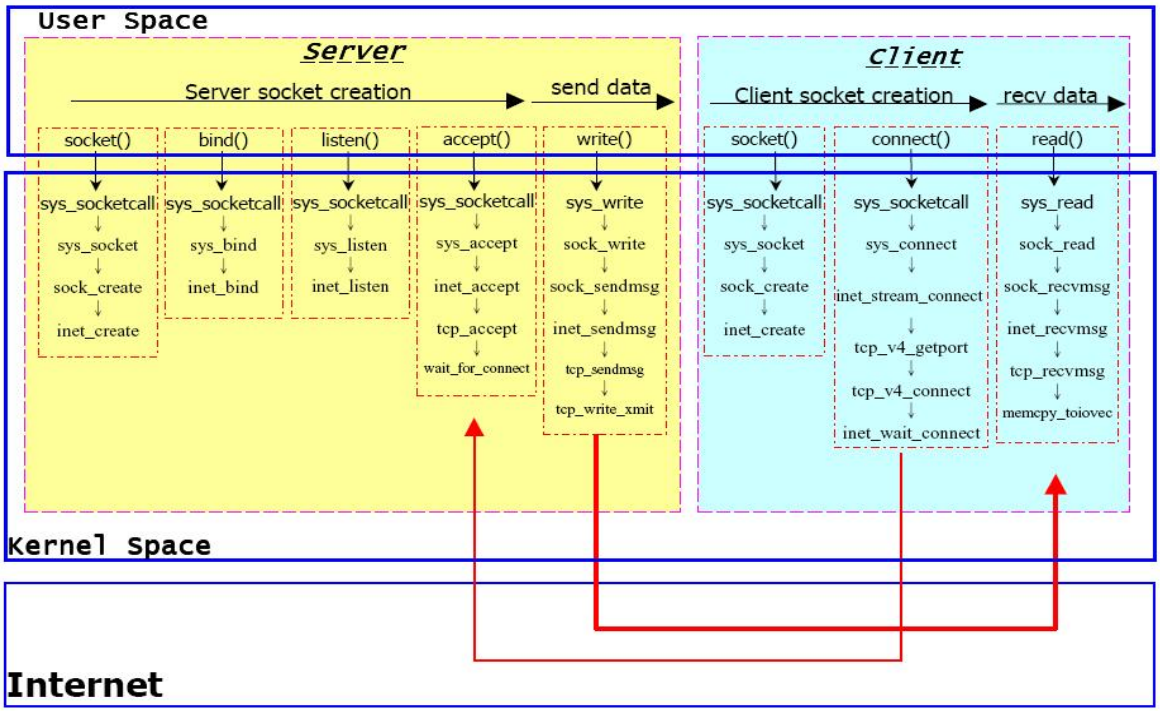
除此之外，Linux 與 BSD 雖然都繼承了 Unix 的特點，但由於 Linux 幾乎算是獨立發展的作業系統，與現行 BSD 繼承 4.4BSD 發展而來大不相同。因此 Linux 與 BSD 在神韻上雖然非常的相似，甚至以相似或相同的介面、函式庫，但在真正實作上卻已經是完全的不同了。再加上 Linux 是遵循 POSIX 標準而 BSD 則是繼續走 BSD 路線，POSIX 與 BSD 的差別會在程式設計師在設計程式的時候表現出不同的設計方法。

3. 以程式設計師的角度比較

3.1 網路程式的比較

圖二：作業系統架構圖

那麼究竟 POSIX 與 BSD 的差異在哪裡？為什麼這些差異會影響到兩者間的設計不同？討論這個問題之前，我們必須先從作業系統的架構談起。由圖二可以發現，一個程式的執行，主要可以分成三層—使用者、核心以及硬體。通常一個應用程式會用 system call 去呼叫核心來處理，而核心則是控制硬體或者利用驅動程式來對周邊設備進行操控。我們以 Linux 與 BSD 在於 networking 的部份來進行解說。

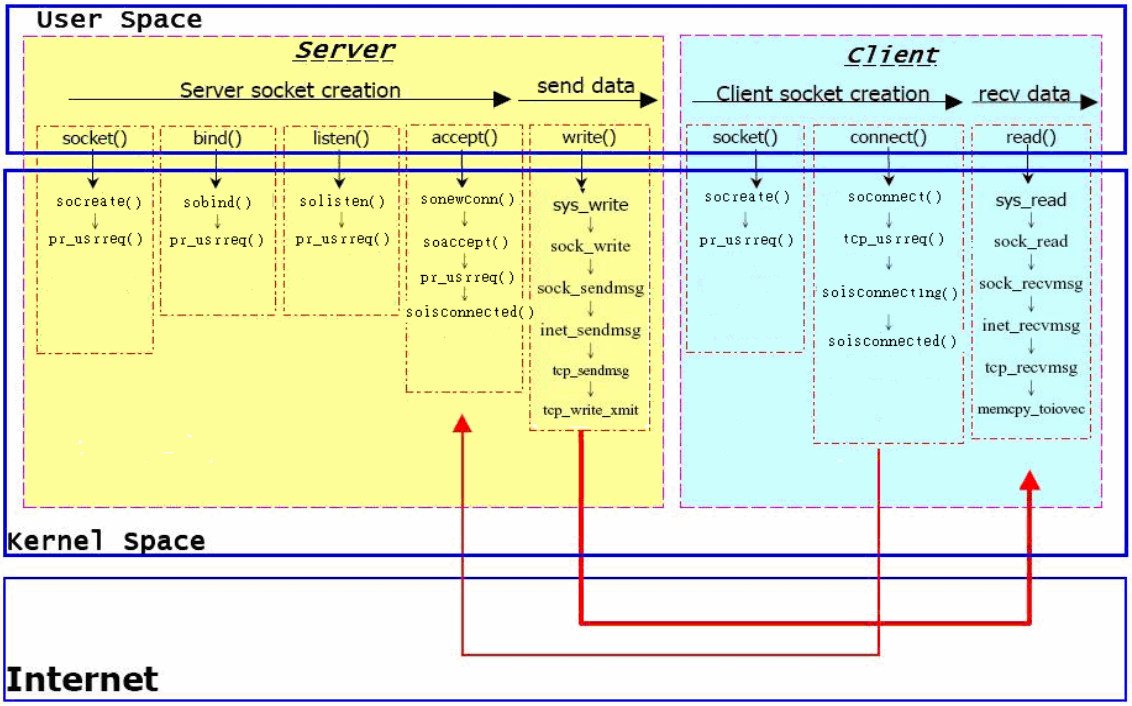


圖三：Linux 由 Server 送資料到 Client 的過程

圖三是我們對 Linux 進行追蹤後所得到的結果，由圖可以發現，我們建立一個網路程式會使用如 socket（）、bind（）、write（）等 system call 來呼叫核心去處理事情，而核心則是負責操控硬體。同時我們也可以看到整個流程，以及核心是如何處理事情，例如 socket（）程式的整個處理方法是當程式中呼叫 socket（）這個 system call 的時候，核心由 sys_socketcall 去處理，再往下由 sys_socket（）、sock_create（）、inet_create（）接續處理。

同樣的我們來看圖四，圖四是 BSD 上由 Server 送資料到 Client 的過程，如果我們拿圖三來比較，會發現在 systemcall 的部份是一樣的，都是使用如 socket（）、bind（）、write（）等 systemcall 來呼叫核心去處理事情，而內部的實作方法則是完全不一樣的內容，如在 Linux 上是 sys_socketcall 去處理，再往下由 sys_socket（）、sock_create（）、inet_create（）接續處理，而 BSD 上則是用 socreate

（）、pu_usrreq（）來處理。因此我們可以發現，對於 Linux 與 BSD 上所使用的 systemcall 是大部分是一樣的，而 kernel 則是用完全不一樣的內容去實做。



圖四：BSD 由 Server 送資料到 Client 的過程

然而，Linux 與 BSD 並不是只有這樣的差別。當我們想要從 Linux 上移植程式到 BSD 或者從 BSD 上移植程式到 Linux 上的時候，如果僅是以上面的結論，會認為直接移植不需修改就沒問題了。實際上雖然 systemcall 相同，但是在一些細節上還是有些不一樣。這裡的問題其實就是 POSIX 標準與 BSD 的不同。由於兩邊的標準並不一樣，所以在設計核心的時候會出現差異，以致於在最後會有不同的方法。

3.2 POSIX 與 BSD 的差別

POSIX 的系統檢查信號的次數，比起一些舊版的 Unix 是稍多了一點，如果是 Linux，發生以下狀況可能就會執行系統檢查信號了—非同步地(asynchronously)(計時器的滴答聲)、系統呼叫的傳回值(on return from any system call) 、在下列系統呼叫的執行期間：select()、 pause()、 connect()、accept()、read() on terminals、sockets、pipes or files in /proc、 write() on terminals、 sockets、 pipes or the line printer、 open() on FIFOs、PTYs or serial lines、ioctl() on terminals, fcntl() with command F_SETLKW, wait4(), syslog(), any TCP or NFS operations。因此當下

面這些系統呼叫(system calls)：creat()、close()、 getmsg()、putmsg()、 msgrcv()、msgsnd()、recv()、 send()、 wait()、 waitpid()、 wait3()、 tcdrain()、 sigpause()、semop()就有問題可能會發生。在系統呼叫期間，若有一信號(那支程式本身應準備好 handler 來因應了)產生，handler 就會被呼叫。當 handler 將控制權轉移回系統呼叫時，它會偵測出它已經產生中斷，而且傳回值會立刻設定成-1，errno 設定成 EINTR。程式並沒有想到會發生這種事，就會出現錯誤訊息。

當 read() 或 write() 在處理資料時，若剛好產生了一個 signal ，系統為了要處理這個 signal ，便會中斷 read() 或 write() ，將程序狀態切換到 signal 的處理動作中。而當 signal 的處理動作結束後，再將程序狀態切換到 read() 或 write() 的後續處理動作。這個後續動作的處理，會影響資料讀取或寫入的完整性。在 POSIX.1 中是這樣說的:

```
Section 6.4.1.2 of POSIX states, "If a read() is interrupted by a signal
after it has successfully read some data, either it shall return -1 with
errno set to EINTR, or it shall return the number of bytes read."
```

圖五: POSIX.1 的部分內容

在 POSIX.1 中，關於 read() 或 write() 被 signal 中斷時，系統所允許採取的兩種方法如下：

1.回傳 -1 並設定變數 errno 的值為 EINTR 。

2.回傳已處理的位元組數目。

由於兩種方法都有系統在使用，因此 POSIX 採用了折衷的方法，兩種方法都允許。如圖六所示，左邊是 Linux 中的寫法，左邊是 Linux 的寫法，如果我們將左邊的程式移植到 BSD 上那麼就會有問題發生，因為在 BSD 上並沒有定義 errno，如果我們硬把 Linux 上的程式(如圖六左方的程式)放到 BSD 上跑，除了要考慮 compiler 是否相容外，這些在 Linux 上有定義而 BSD 沒有定義的部份就會出錯，類似這樣的問題也需要去做考慮。因此如果今天我們要在兩套系統上移植軟體，就要小心這一類的問題。更多關於這方面的討論可以觀看 POSIX 的標準與 BSD 的標準得知。

```
int result;
while (len > 0) {
  result = read(fd,buffer,len);
  if (result < 0) { if (errno != EINTR) break; }
  else { buffer += result; len -= result; }
}
```

```
int result;
while (len > 0) {
  result = read(fd,buffer,len);
  if (result < 0) break;
  buffer += result; len -= result;
}
```

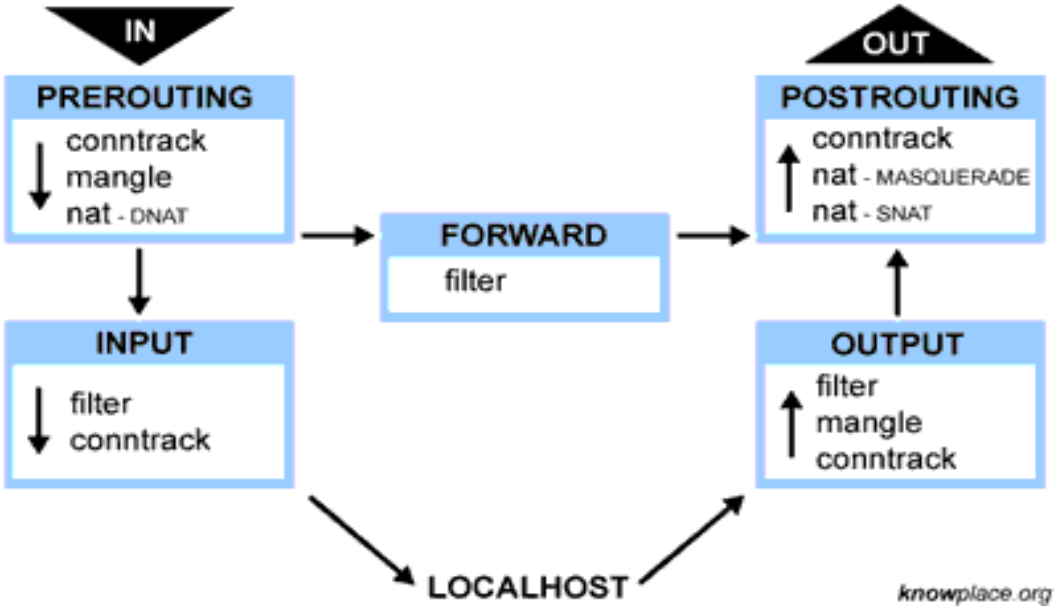
圖六：Linux(POSIX 標準)與 BSD 在實做的差異

4. 以現行的網路技術比較

討論完了差異性，接下來我們想要知道的就是在現行的版本下，兩者效率上的比較。由於效率問題牽扯的範圍非常的廣泛，如記憶體管理、行程排序、封包傳遞等都是效率問題的範圍之內，因此我們僅選擇了單純的封包傳遞來加以比較。在 Linux 上內建的封包傳遞的應用程式為 IPTable，而在 BSD 上內建的封包傳遞的應用程式為 IPFilter，我們將對這兩者進行比較以及效率測試。

4.1 Linux 的 IPtable

首先我們對 IPtable 做簡介，iptables 是 Linux 上內建的防火牆，當然也提供了許多技術，那就是對封包的連線狀態，作出更詳細的分析，例如：是否為新連線或回應封包、是否為轉向連線、連線是否失去回應，連線時間是否過長.....等等，另外也開發出真正的封包改寫能力，不需要透過其他程式的協助來模擬網址轉譯，除此之外，iptables 也獲得系統核心的直接支援。



圖七：iptables 的運作圖

圖七是 iptable 概念下的運作圖，可以知道 iptables 如何處理封包的流動，分述如下：

IP INPUT：只有要到達本機的封包才會由 INPUT 函式處理，所以會讓來自內部網路的封包無條件放行，來自外部網路的封包則過濾是否為回應封包，若合格則放行。

PREROUTING：需要轉送處理的封包皆由此函式負責處理，此函式用來做目的地 IP 的轉譯動作（DNAT）。

IP FORWARD：所有轉送封包都在這裡處理，這部分的過濾規則最複雜。

POSTROUTING：轉送封包送出之前，先透過這個函式進行來源 IP 的轉譯動作（SNAT）。

IP OUTPUT：從本機送出去的封包由這個函式處理，通常會放行所有封包。

iptables 可以自行定義規則群組（rule-set），規則群組被稱為規則鍊（chains），前面所描述的函式，也都有相對應的規則鍊（INPUT、FORWARD、OUTPUT、Prerouting、Postrouting）。iptables 則是採用規則堆疊的方式來進行過濾，當一個封包進入網卡，會先檢查 Prerouting，然後檢查目的 IP 判斷是否需要轉送出去，接著就會跳到 INPUT 或 Forward 進行過濾，如果封包需轉送處理則檢查 Postrouting。如果是來自本機封包，則檢查 OUTPUT 以及 Postrouting。過程中如果符合某條規則鍊將會進行處理，處理動作除了 ACCEPT、REJECT、DROP、REDIRECT 和 MASQUERADE 以外，還多出 LOG、ULOG、DNAT、SNAT、MIRROR、QUEUE、RETURN、TOS、TTL、MARK 等，其中某些處理動作不會中斷過濾程序，某些處理動作則會中斷同一規則鍊的過濾，一直到堆疊中的規則檢查完畢為止。透過這種機制所帶來的好處是，我們可以進行複雜、多重的封包過濾。

此外 IPtable 裡面主要的內建的規則表有三個，分別是：

Nat：此規則表擁有 Prerouting 和 postrouting 兩個規則鍊，主要功能為進行一對一、一對多、多對多、埠轉送等網址轉譯工作（SNAT、DNAT、PNAT）。由於轉譯工作的特性，需進行目的地網址轉譯的封包，就不需要進行來源網址轉譯，反之亦然，因此為了提升改寫封包的效率，在防火牆運作時，每個封包只會經過這個規則表一次。如果我們把封包過濾的規則定義在這個資料表裡，將會造成無法對同一封包進行多次比對。

mangle：此規則表擁有 Prerouting、FORWARD 和 postrouting 三個規則鍊。除了進行網址轉譯工作會改寫封包外，在某些特殊應用可能也必須改寫封包（TTL、TOS）或者是設定 MARK（將封包作記號，以便進行後續的過濾），這時就必須將這些工作定義在 mangle 規則表中。

filter：這個規則表是預設規則表，擁有 INPUT、FORWARD 和 OUTPUT 三個規則鍊，這個規則表顧名思義是用來進行封包過濾的處理動作（例如：DROP、LOG、 ACCEPT 或 REJECT），我們會將基本規則都建立在此規則表中。

4.2 BSD 的 IPFilter

至於 BSD 的 IPFilter 是套能提供 NAT 與防火牆功能的軟體，相較於其他防火牆，他內建 NAT 算是比較特別的(IPtable 也是內建 NAT)，但是 IPfilter 的 NAT user mode 需要由 IP NAT 這支程式去處理，而 IPtable 本身即可處理。他主要能夠依據來源/目的地位址，網路介面，IP 通訊協訂，19 種 IP options 或 8 種 IP 安全等級的服務種類(Type of Service, TOS), 是否為分段封包等多個項目來拒絕或允許封包的流入流出。IPFilter 在做規則比對的時候，是由最上面一條比對下來，但卻採用最後一個比對成功的規則。換句話說，他不會在第一個符合時，便結束比對工作。

由此我們可以發現，IPtable 與 IPFilter 各有自己的設計方法，以表二為兩者間的一些比較。

	Linux	BSD
Hook support	Y	Y
First handle function	Prerouting	fr_check
規則比對	按照流程持續進行下一個規則鍊的過濾（即不斷的比對規則並處理），一直到堆疊中的規則檢查完畢為止	由最上面一條比對下來，執行最後一個比對成功的規則。
快速路由	N	Y
處理方法	邊比對邊做	執行最後一個比對成功的規則
user space 的處理	IPtable 本身	須執行 IP NAT

表二：IPtable 與 IPfilter 的比較

4.3 效率比較

作完了理論上的比較，我們來看看實際上的測試數據，來比較兩者之間的差異，表三是我們實驗的時候用的機器設備。

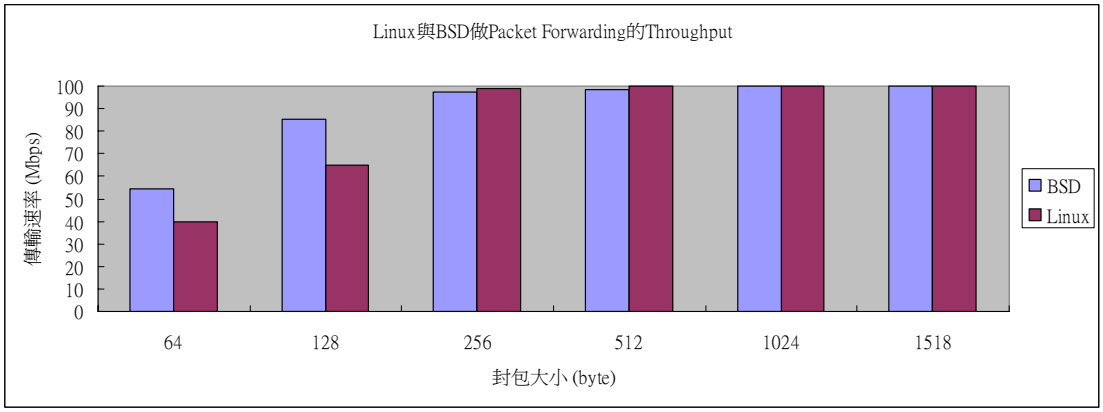
	設備	
CPU	Intel P-III 800	
Ram	256MB	
網路卡	Intel Pro 10/100M	
測試機器	SmartBit2000	
OS	Linux Rat Hat Fedora core 2	BSD FreeBSD4.10-RELEASE
軟體	IPtables	IPfilter

表三：實驗設備

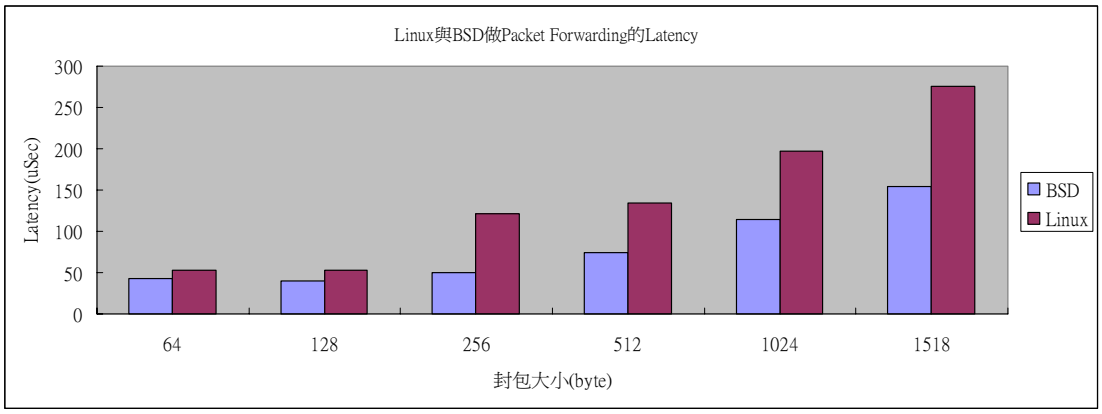
首先我們來看 Linux 與 BSD 在各種封包大小下的 throughput，圖八為實驗結果。我們可以發現，BSD 的 IPfilter 在封包大小較小時有著明顯的優勢，當封包大小為 64bytes 的時候，BSD 的 throughput 比 Linux 足足多了 10Mbps(其數據分別為 Linux 40Mbps 、BSD 54.2Mbps)。然而在封包大小為 256bytes 的時候，Linux 與 BSD 都是以接近 100Mbps(其數據分別為 Linux 99Mbps 、BSD 97Mbps)的 throughput。我們可以發現，當封包大小漸漸變大的時候，Linux 傳送封包的 Throughput 才追上 BSD，而在封包大小較小的時候，BSD 比 Linux 有更大的傳輸速度。

我們再來看兩者的 Latency 的比較，圖九是 Linux 與 BSD 在各種封包大小下的 Latency。我們可以發現，BSD 在各種封包大小下的 Latency 都明顯優於 Linux。不管是在小封包(64bytes)時，或者大封包 1518(bytes)時，BSD 處理封包的速度都比 Linux 更快，甚至大封包的狀況下有著非常明顯的差距。因此，由實驗數據我們可以發現 BSD 在作封包傳遞功能的時候效能明顯優於 Linux。

從以上兩個部分我們可以發現，BSD 的 IPfilter 不管是在 throughput 或者是 Latency，在單純的封包傳遞的效率比較下都勝過 Linux 的 IPtable。



圖八：Linux 與 BSD 做 Packet Forwarding 的 Throughput



圖九：Linux 與 BSD 做 Packet Forwarding 的 Latency

5. 結論

在我們做完一連串的比較之後，我們可以作出一些結論。首先我們可以發現 Linux 是獨立開發的作業系統，雖然在神韻上繼承了 Unix 的風格，然而內部實作的部分已經有自己的設計理念。反觀 BSD 的核心，BSD 是繼承了 Unix 的歷代版本直接改版而來，再加上 BSD 有 Core team 來開發，而 Linux 沒有(Core team 是負責專門開發核心的一個 team，在 Linux 上人人都可以參予這項工作，而在 BSD 上則是有專屬的 team)。因此我們可以感受到 Linux 活潑且更新快速的風格，而 BSD 則會令人覺得比較穩重的持續開發，更新速度也沒那麼快。然而相對的，Linux 的雖然有著活潑且更新快速的風格，但是也因此造成原始程式碼較 BSD 來的混亂，同時也造成穩定度不如 BSD。

然而我們從另一個角度來看的話，會發現其實在 Linux 上與 BSD 上開發程式其實是非常接近的。這同時也告訴我們要移植程式的話，通常會碰到的問題並不會太多，主要是因為兩邊所使用的 system call 幾乎是相同的，不同的地方是在

於其內部實做是兩個不同的方法，只要沒有牽涉到這一部份，兩者之間幾乎是沒有相異之處的。然而真正有相異之處的地方就是在 Linux 是遵循 POSIX 標準而 BSD 不是，這麼一來兩邊就會有細微的差異，通常這個問題可以觀看 POSIX 標準跟 BSD 標準，來解決兩者間的差異。

除此之外，在現行的 Linux 與 BSD 版本下，我們對於兩者間的效率比較有很大的興趣，這對於研究、商業都相當的重要。然而，效率這件事會牽涉到很多部分，若我們單純以兩者間的封包傳遞來比較，根據實驗的結果我們會發現 BSD 不管是在 Throughput 或者 Latency 上皆明顯的比 Linux 來的快。造成這樣的原因有值得我們深入探討的部份，對於研究兩者核心的人，則有著相當重要的價值存在。

6. 參考文獻

[1] Linux Kernel Development; Robert Love; Developer's Library,2004.

[2] Understanding the Linux Kernel Daniel P. Bovet & Macro Cesati ; O'Reilly, 2001.

[3] The Design and Implementation of the 4.4 BSD Operating System, Marshall Kirk McKusick, Keith Bostic, Michael J. Karels and John S. Quarterman, 1996.

[4]TCP/IP Illustrated vol.2 ,Richard Stevens, Gary R. Wright.

[5] Operating Systems : http://www.csie.nctu.edu.tw/~shieyuan/course/os/2003/csie/

[6]蔡孟甫、曹世強、林盈達; 「NetBSD核心網路安全模組: IPFilter及IPSec」 http://speed.cis.nctu.edu.tw/~ydlin/miscpub/hands-on_security_kernel_modules.pdf

[7]陳一瑋、林盈達； 「Linux網路卡驅動程式　追蹤與效能分析」； 網路通訊； 136 期，2002 年 11 月。 http://speed.cis.nctu.edu.tw/~ydlin/miscpub/linuxdrivers.pdf

[8]蔡品再、林盈達；『追縱 Linux 核心的方法』；網路通訊； 113 期，2000 年 11 月。 http://speed.cis.nctu.edu.tw/~ydlin/miscpub/remote_debug.pdf

[9] Firewalling with Netfilter / Iptables http://www.knowplace.org/netfilter/index.html

[10] 使用iptables建置Linux防火牆 http://www.spps.tp.edu.tw/documents/memo/iptables/iptables.htm