課程名稱：網路作業系統(NOS)
主題：網路作業系統導論
編撰者：王彥彬、陳世偉、林世唐
參考文獻：
Leland L. BECK (1997), SYSTEM SOFTWARE 3th 中文譯本, 林晉豐、林騰蛟等譯, Addison-Wesley, 文京.
Silberschatz, Galvin, Gagne (2000), Applied Operating Systems 中文譯本, 駱詩軒、駱詩富譯 Wiley, 東華.
Introducing Microsoft Windows 2000 Server, 林文德 譯, 碁峰
Microsoft Windows NT 4.0 核心技術, 鍾俊仁 編譯, 微軟.
Microsoft Windows NT 4.0 企業網路規劃與管理, 呂聰輝 編譯, 微軟.
Microsoft Windows 2000 升級篇, 呂聰輝 編譯, 微軟.
Windows 2000 Server 建構徹底研究, Harry M. Server 原著, 旗標.
Windows 2000 Server 系統實務, 施威銘研究室 著, 旗標.
Windows 2000 Server 網域建置與安全維護, 莊彥澤 著, 學貫.
Windows 2000 Server 企業網路建構實務-群組原則與安全規劃篇, 顏逸品 編著, 博碩.
Windows 2000 系統管理徹底研究, George Spalding 著, 麥格羅希爾.
Windows 2000 架站實務, 施威銘研究室 著, 旗標.
作業系統原理：Windows 核心剖析, 尤普元 等著, 麥格羅希爾.
作業系統概念的應用, Avi Silberschatz, Peter Galvin, Greg Gagne 著, 東華.
認識 Active Directory Service 目錄服務, 廖裕康 陳仁泰 編譯, 華彩

教材編撰年度：91 年度。

單元一 網路作業系統導論

第1章 作業系統簡介

1.1 概述

作業系統的主要目的在於使電腦更易於使用，提供使用者能夠執行程式的環境。作業系統負責管理電腦的各種資源，企圖達到整體的系統目標，雖然資源管

理的細節相當複雜，但通常作業系統都會隱藏起來，不讓使用者看到。

電腦系統約略可以分成為四種元件:硬體、作業系統、應用程式、及使用者)。而其中最重要的部份就是它的作業系統。

硬體主要負責了運算的部分，而應用程式藉著使用這些資源，來解決使用者的問題。而作業系統的作用，便是在管理控制各個應用程式對硬體資源的使用。所以，作業系統也可以當作一個資源分配者(resource allocator)。系統擁有許多的資源：如 CPU 的時間、記憶體的空間、檔案儲存空間、I / O 裝置等等。作業系統事實上就是一個管理者,並且分配這些資源給需要它們的某些程式與使用者。

網路作業系統(network operating system)提供了網路上檔案分享的特性，其中也包括了可以讓不同電腦系統上相異處理器溝通的技巧。在網路作業系統上運作的電腦能自主地運行，獨立運作於其他網路上，並且也能和其他的網路電腦溝通。

然而，作業系統至今仍沒有絕對的定義。作業系統的存在仍是因為它們可希望能提供一個環境能夠將問題有辦法解決，以便產生一套有用的電腦系統。電腦系統的功能其實就是執行程式，為了這個目的，我們把便結合了電腦硬體和作業系統，所以才發展出了應用程式來，不同的程式之間仍就需要一些共同的運作。而作業系統可說是就是由這些功能所共同集合在一起而成的軟體。

1.2 作業系統的型態

最常見的作業系統分類是依照使用者介面來區分。作業系統的分類的方法之一是依系統表示在同一時間內能支援的使用者數目來區分。單工(single-job)的作業系統表示在同一時間內只能執行一個使用者的工作。今天在微電腦和個人電腦中最常見的單件系統，是最早的作業系統型態。由於有限的記憶體容量及缺乏資料傳輸通道和其他資源,系統要在這樣的機器上支援一個以上的使用者實屬困難。

多工程式(multiprogramming)系統允許同時執行多個使用者的工作。作業系統負責在不同使用者工作間切換 CPU 的使用權。它同時提供一個合適的執行時期環境和其他支援性功能，使得各工作不會因此互相干擾。多處理器系統(multiprocessor)和多工系統相似，不同的是會有一個以上的 CPU 可供使用。在大多數的多處理器系統中，處理器共享記憶體，因此使用者亦可將此系統視為一個強而有力的系統。

1.3 個人電腦系統 (Personal-Computer System)

1970 年代才開始有個人電腦的出現。一開始 PC 受到了使用者程式的限制，

所以 PC 並無法多工也不會有多重使用者。但隨著慢慢進步，系統也增加了讓使用者較為方便使用的地方。例如微軟的一系列 WINDOWS 和 APPLE 的 Macintosh 都是如此。

由於現在 PC 電腦常會連線到其他的電腦,所以檔案保護對 PC 來說已經變得相當重要。當然，這可以藉著很多方法或設定來實施。當使用者能對其他電腦進行存取的同時，檔案保護系統就變成了 PC 上一個重要的項目。除了被他人惡意修改外，也有可能被病毒所入侵。

大致上來說，原先大型電腦上才有的特性，如今 PC 上也慢慢的具備。而個人電腦功能更強、更快的硬體也一直發展出來。

1.4 批次系統 (Batch Systems)

早期的電腦都由控制台整合控制，一般輸入裝置只有讀卡機和磁帶機，輸出裝置則是印表機、磁帶機。在這種系統運作下，使用者並沒有辦法直接與電腦溝通，只能準備好打孔紙卡，在一段時間之後就可以輸出。輸出的內容往往都是程式的結果。

在早期電腦中，作業系統都非常的簡單。它主要的工作是自動地控制轉換一個工作到下一個工作。為了提升處理速度，通常都會將相同或類似的工作整批集中在一起，並且一起交給電腦處理。因此將具有相似要求的程式安排成同一批，讓當電腦可以分批執行之，每個工作的輸出再送回給每一個使用者，這就是所謂的批次觀念。

1.5 平行系統 (Parallel System)

大多數的電腦系統仍是單一處理機的系統；也就是說這些系統都只有一個 CPU。不過現在電腦已慢慢傾向於多個 cpu 的系統，並且使用共同的資源。

使用多元處理系統的原因和好處有許多。如效能的提升，藉由處理器的增加，我們可以在較少的時間內完成較多的工作。但是多部處理機共同處理一件工作之時，都會產生一些額外的負擔來確保工作不會有失誤；另外還會有資源使用上的競爭問題。

另外還有一個好處：相對於單一處理機而言，多元處理可以達到節省經費的目的，因為多部處理機可以共用週邊資源。

目前最常見的多元處理機系統是使用對稱多元處理(symmetric multiprocessing)模型，這類系統中的每個處理器都是在同一個作業系統上執行，有些系統則採用非對稱多元處理，(asymmetric multiprocessing)，每個處理機都會被分配一個特殊的工作。此種方式有一種主從的關係，主處理機要分配工作給附屬處理機。

對稱多元處理的處理器彼此間沒有主從關係，由於CPU是分開獨立的，因此如果有一個閒置或故障時，其他的可能會負擔過大而沒有效率。

1.6 分時系統 (Time-Sharing System)

多元程式規劃整批作業系統提供了不同系統資源有效利用的環境,但是並沒有提供使用者和電腦系統的溝通。分時是CPU在數個工作之間輪流轉換使用以便執行這些工作，因為轉換發生頻率很高，所以使用者可以在每一個程式執行時與其溝通。

分時作業系統允許很多使用者同時共用一部電腦。因為分時系統中每個使用者只輪流分配需要一點點CPU時間，當系統非常快速地分配使用者時，使用者並不會覺得系統是共用的。

分時作業系統使用CPU排班和多元程式規劃。每個使用者在記憶體中都會有自己的程式。當一個程式執行時，通常在執行Ｉ/Ｏ時只需要短暫的時間，Ｉ/Ｏ即是輸出入的動作。因為Ｉ/Ｏ通常都要花較長的時間來執行，當有輸出入的動作時，作業系統會將CPU轉換到其他使用者的程式。

1.7 即時系統(Real-Time System)

即時系統為一種有特殊目的的作業系統。即時系統是使用者對於處理器的操作或資料的傳送在時間上很要求時所使用的，通常都是應用在控制裝置上，如一些武器系統、醫學系統都必須在正確時間內完成的系統。

即時系統分為兩種型態。一種為硬性即時系統(hard real-time)，它可以保證重要的工作能準時完成。但是這種限制會限定了系統中所能得到的資源，任何次要的儲存設備都會受到相當的限制。

另一類限制較少的為軟性即時系統(soft real-time)。在這種系統裡有優先權的機制，重要的工作比其他的工作有更大的優先權，並一直保持到工作完成。此種系統跟硬性一樣對核心所造成的延遲都要有限制。但是軟性即時系統的應用比硬性有限，因為缺乏了限期的功能，所以並不能應用在控制系統上，但是也有廣大領域的應用，如多媒體、虛擬實境等。

1.8 分散式系統 (Distributed System)

PC 在1970年代引進時，本是設計成個人使用，而且被當作為一台獨立的電腦。但在90年代引進了網頁和網路不斷發展，網路連接也被視為電腦系統的一個部分。

電腦系統中的新趨勢為將計算工作分散到多個處理機之中。但所有的處理機

並不共用記憶體或同一時脈。反之，每個記憶體都有其本身的記憶體。各處理機的溝通是經由各種不同的通訊線，如高速匯流排或電話線。

第2章 行程管理與排班

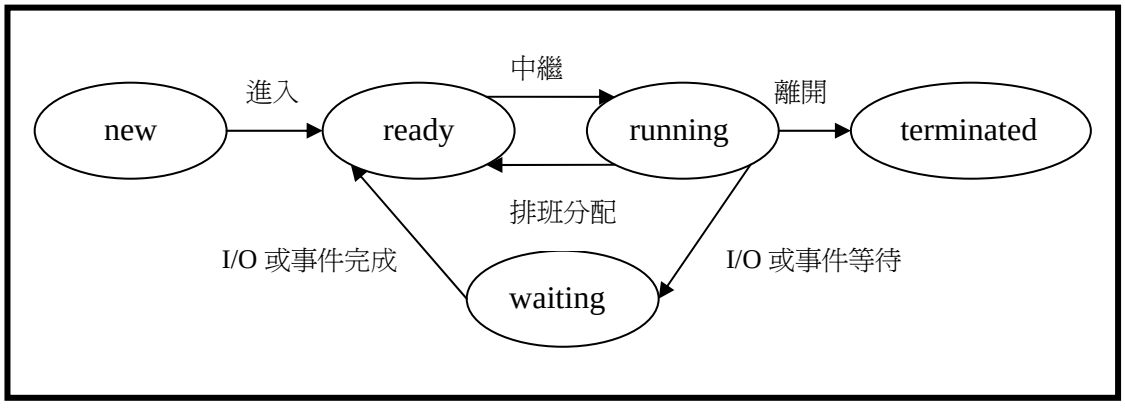
2.1 行程概述

早期的電腦一次都只能執行一個程式,但是今日的電腦系統已經允許多個程式同時執行。所以系統必須對程式做監控的動作，所以產生了行程(process) 的觀念。

根據比較大眾，行程指的就是正在執行的程式。行程在執行時會改變狀態。每個行程都會處於下列一種狀態中：

- new：產生行程
- running：執行指令
- waiting：等待某些工作完成或發生
- ready：準備完成等待一個處理機
- terminated：行程完成執行

這些狀態名稱可能因作業系統而異，但是狀況大約相同。下圖為示意圖：



系統對各個行程都會記載其相關資訊，裡面包括了行程狀態、程式計數器、CPU 暫存器、CPU 排班法則等相關資訊。

2.1.1 合作行程

在作業系統中行程可分為獨立行程和合作行程兩大類。如果一個行程不會影響人家也不會受人家影響就是獨立行程。也就是說，獨立行程並不會和其他行程

共用資源。反過來說，一個行程會受其他行程影響或影響別的行程的就是合作行程。

作業系統之所以有提供合作行程的機制有以下幾點理由：

- 資訊共享：因為可能有很多的使用者想使用相同的資訊，所以必須提供可以讓使用者同時使用的機制。
- 加速運算：如果要讓一些工作可以執行得快一點，可以分成一些子工作，每一個子工作可以平行的工作，這樣可以達到加速的目的。
- 模組化：以模組的方式建立系統，把系統功能分配到數個行程。
- 便利性：讓單一使用者也可以同時執行很多工作。

但是要注意的是，行程要同時執行且互相合作時，就需要允許行程之間互相能通訊。

合作行程之間可能直接分享一塊邏輯空間，或是只能經由檔案分享資料。但同時存取共同資料可能會造成資料前後不一致，所以必須要有一些方法以確保分享同一塊邏輯地址空間的合作行程或執行緒可以有秩序的執行，使資料可以維持前後的一致性。

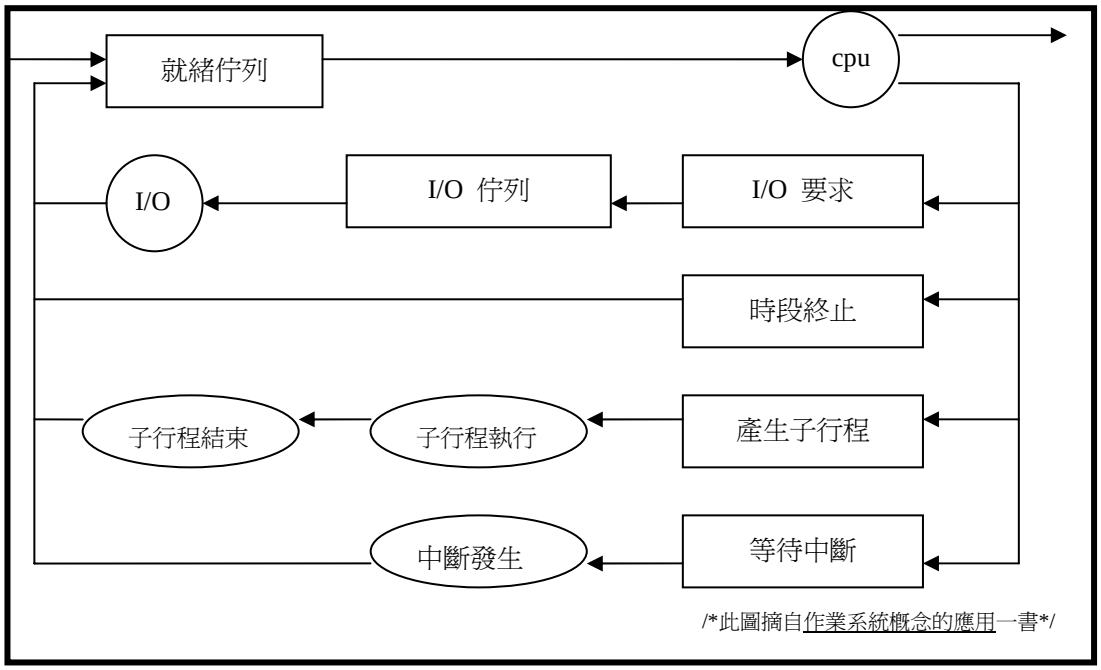
造成這種問題是因為允許數個行程同時存取和處理相同資料的情況發生，這就叫做競爭情況(race condition)。為了避免這種情況發生，我們必須確定一次只有一個行程來處理一塊資料區，而這就是行程必須同步的重要地方。

2.2 行程排班

多元程式規劃的主要目的，便是要提高 CPU 的使用率。而分時系統的功能是讓 CPU 在不同行程之間不停轉換，以便讓使用者可以在自己的行程時與它交談。在單一處理機系統裡，不能同時執行多個行程。如果有多個行程，就必須有一個排班的機制。

當行程進入系統時，它們是放在工作佇列之中。位在主記憶體中且就緒等待執行的行程是保存在一個所謂的就緒佇列(ready queue)裡。在系統中還有其他的佇列，當某行程配置到 CPU，它會執行一段時間並且最後會停下來、中斷、或等待某一特殊事件發生(譬如一項 I/O 要求的完成)。

行程排班的一般表示法，如下圖所示(每個長方形格子中是一個佇列，圓圈代表佇列服務的資源，箭頭表示行程在系統中的流動方向。)：



一個行程會在各個不同的排班佇列間遷移，而作業系統必須按排班次序從這些佇列中選取行程。行程的選取將由適當的排班程式來執行。長期排班程式(long-term scheduler)會選出行程並執行。短期排班程式(short-term scheduler)從記憶體中選出一個已經準備就緒的行程，並且將 CPU 分給它。

這兩種排班程式的主要區別就是它們的執行次數。短程排班程式必須經常為 CPU 選擇新的行程，一個行程可能只執行幾毫秒然後等待一項 I/O 要求。而且因為每一次執行所相隔的時間非常短，所以短程排班程式必須非常快速。在另一方面，長期排班程式執行次數少了很多。

一般來說，長程排班程式大多用 I/O 傾向或是 CPU 傾向來區分。一個 I/O 傾向的行程在做 I/O 的時間比在做計算的時間還多。反之，一個 CPU 傾向的行程不常產生 I/O 要求，而使用於做計算的時間比 I/O 傾向行程還要多。讓長程排班程式選擇一個適當的 I/O 傾向與 CPU 傾向的混合比率是十分重要的。

有些系統，如分時系統，可能會採用一種額外的方式來排班，這種的稱為中程排班程式(medium-term scheduler)。中程排班程式的觀念就是我們有時候可以將行程從記憶體中移開，由此減低多元程式規劃的程度。

2.3 CPU 排班

在單一處理機系統裡，不可以有多個程式在執行。而多元程式規劃的主要目的就是要一直保持有行程在執行，以提高 CPU 的使用率。因為如此，所以 CPU 排班自然是系統裡重要的一環。

2.3.1 概述

多元程式的概念非常簡單，就是許多行程同時存放在記憶體之中，當某個行程在等待時，其他的行程就可以先取得 CPU 的控制權。一旦 CPU 閒置，作業系統就從就緒佇列中選取一個行程來執行，而這個選取的動作是由短期排班程式（short-term scheduler）來執行。

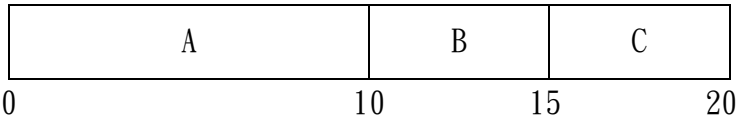
不同的排班方法有不同的特性，許多方法會對某一類的行程比較有利。有很多評估標準可以評估，但通常採用以下幾種：

- CPU 使用率（CPU utilization）
  這個方法主要是在看 CPU 的忙碌程度，
- 產量（throughput）
  產量是一種衡量工作量的方法，就是用每單位時間所完成的的行程數來計算。
- 回復時間（turnaround time）
  回復時間指的是一個行程到底需要多少時間才能完成，也就是一個行程從進入電腦，一直到它完成離開電腦的時間。當然回復時間愈短愈好。
- 等候時間（waiting time）
  行程在佇列裡等待的時間。
- 反應時間（response time）
  反應時間指的是一個行程提出第一個要求到第一個反應出現的時間間隔來計算

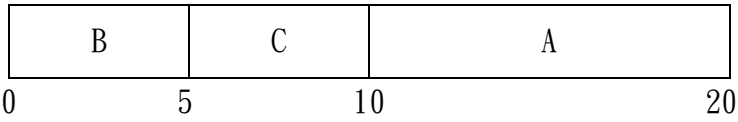
2.3.1.1 先來先做排班法（ First-Come First-Serve Scheduling）

最簡單的 CPU 排班法則就是 FCFS 演算法，也就是依照順序將 CPU 分配給提出要求的的行程。

但在 FCFS 方法之中平均等待時間通常都很長，舉個例來說，如果有三個行程分割時間分別為 A：10　B：5　C：5 ，那麼如果到達順序為 ABC 之時如下圖：



我們可以發現其等待時間為(0+10+15)/3 = 8.3
但是如果行程到達的次序是 BCA，那麼時間圖則為：



此圖就表現出另一種結果，其等待時間則為(0+5+10)/3 = 5
因此 FCFS 中的平均等待時間並非最小的，並且可能隨 CPU 分割時間的極大變化而差異甚大。

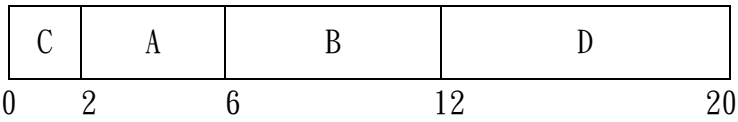
假設我們有一個 CPU 傾向的行程以及許多 I/O 傾向的行程。當這些行程在系統中操作的時候，就產生以下的情形：CPU 傾向的行程將得到 CPU 並且持有它。在這段時間內所有其他的行程都將做完 I/O 工作，並且移進就緒佇列中等待 CPU。當它們在就緒佇列中等待的時候，I/O 裝置都在閒置。最後 CPU 傾向的行程完成它的 CPU 分割並且移到一項 I/O 裝置。所有的 I/O 傾向的行程就只有很短的 CPU 分劃，很快執行之後再移回到 I/O 佇列之中。此刻 CPU 就閒置著，然後 CPU 傾向的行程再移回就緒佇列中並且配給 CPU。所有 I/O 行程都結束它們在就緒佇列之中的等待，直到 CPU 傾向的行程做完為止。當所有其他的行程都在等待一個大行程離開 CPU 的時候，就產生所謂的護送現象(convoy effect)。這個現象造成 CPU 和裝置的便用率降低，如果比較短的行程可以先做，裝置將可以獲得更高的使用率。

FCFS 排班演算法是一種不可搶先的演算法，一旦 CPU 分配給某個行程，該行程就一直佔住 CPU，直到它結束或要求執行 I/O 而釋放出 CPU 為止。

2.3.1.2 最短的工作先做之排班方式 (Short-Job-First Scheduling)

SJF 演算法就是將每一個行程的下一個 CPU 分割長度和該行程相結合。當 CPU 有空時，就指定給下一個 CPU 分割最短的行程。如果兩個行程具有相同長度的下一個 CPU 分割，那麼就採用先來先做(FCFS)的方法。

舉個例子，如果有四個行程分割時間分別為 A：4　B：6　C：2　D：8，到達順序為 ABCD，以 SJF 方法排班如下圖：



則我們發現其平均等待時間為(0+2+6+12)/4 = 5
但是我們如果使用 FCFS 的排班方式，那麼平均等待時間將為(0+4+10+12)/4=6.25。

SJF 已經證明為最理想的方法，採用這種方式我們可以得到一組行程的最小平均等待時間。將一個短行程移到長行程之前將使行程的平均等待時間減少。

SJF 的實際困難在於如何得知下一個 CPU 分割的長度。對於批次作業系統之

中的長程排班來說，我們可以使用行程時間限制，因此，使用者可以提高正確估計行程時間的限制，因為越低的估計值就代表越快的回覆。

雖然 SJF 非常理想，但它不能夠執行在短程 CPU 排班的層次。我們無法預知下一個 CPU 分割的長度。

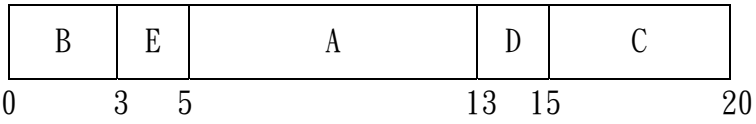
SJF 可以是不可搶先或者是可搶先。如果有一個行程正在執行，而另有一個新行程到達就緒佇列之中，問題就產生了。新的行程可以比目前正在執行的行程所剩部份有較短的下一個 CPU 分割。如果是可搶先的最短工作先做演算法，新行程就會搶在目前正在執行的行程之前執行。但是不可搶先的最短工作先做演算法，就會讓目前正在執行中的行程完成它的 CPU 分割。

2.3.1.3 優先權排班方式 (Priority Scheduling)

在 PS 演算法中每個行程都有自己的優先權，PS 演算法就是將 CPU 分配給具有最高優先權的行程，具有相同優先順序的行程則按照 FCFS 來排班。優先權一般是一些固定範圍的數字，例如 0 到 7 或 0 至 127。但是，0 是最高優先權還是最低優先權並沒有一致的看法。舉例說明，一組行程如下：

行程	分割時間	優先順序
A	8	3
B	3	1
C	5	4
D	2	3
E	2	2

則使用 PS 方法的時間甘特圖如下：



平均等待時間為(0+3+5+13+15)/5 = 7.2。

優先權可以由內部或外部定義。內部得到的優先權是使用一些可以測得的數值定義一個行程的優先權。如時間限制、記憶體需求、開啟檔案數量以及平均 I/O 分割與平均 CPU 分割的比率都可以用來計算優先權。外部得到的優先權是由作業系統外部的一些標準所決定，譬如行程的重要性、支付使用電腦費用所使用的基金類型與數量。

PS 也可以是可搶先或不可搶先。當某個行程到達就緒佇列之後，它的優先權會和目前執行中的行程之優先權比較。在 PS 中，如果新到達的行程的優先權比執行中的行程高，就會搶走 CPU 先做。一個不可搶先的 PS 演算法只能把新的

行程放在已就緒佇列的前端。

PS 演算法所遭遇的最大問題就是饑餓(starvation)。這個問題便是一個已經就緒要執行的行程，但是因為優先權太低一直搶不到 CPU。優先權排班演算法可能會造成一些優先權低的行程一直窮等 CPU。在一個工作繁重的電腦系統之中，一連串高優先權的行程會讓低優先權的行程始終得不到 CPU。

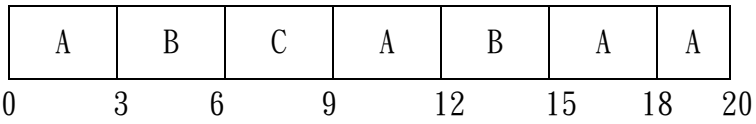
PS 的饑餓問題可以採用老化(aging)的方法解決。所謂老化就是逐漸提高停留在佇列之中的行程的優先權。舉例來說，如果優先權的範圍是由 0(低)到 127(高)，我們可以每隔十分鐘將某個等待行程的優先權加 1。最後，即使這個行程優先權原先是 0，最後也將獲得系統之中的高優先權並且執行。

2.3.1.4 依序循環之排班方式 (Round-Robin Scheduling)

RR 方法是特別為分時系統而設計的。這種排班方法和 FCFS 排班法相類似，但是多了加入可搶先的規則以便讓行程互相交換使用 CPU。我們定義一個小的時間單位，稱為一個時間量(time quantum)。就緒佇列為一個環狀佇列。CPU 排班程式繞著這個就緒佇列走，CPU 在時間間隔達到一個時間量時就會分配給下一個行程。

有兩種情況可能發生，行程的 CPU 分割可能比一個時間量小，這種情形下行程必須自動交還 CPU。但如果正在執行的行程分割時間比一個時間量還長，計時器將停止並產生一個中斷以執行內容轉換，並且將該行程置於就緒佇列的最後。

然而在 RR 方法下的平均等待時間通常是很長的。舉例來說，以一組行程 A：11　B：6　C：3 到達順序為 ABC，時間量為 3 單位，其甘特圖如下：



其平均等待時間為 (0+3+6+(9-3)+(12-6)+(15-12)) / 3 = 8

在 RR 演算法之中，沒有一個行程所分配的 CPU 時間會超過一個時間量。如果行程的 CPU 分割超過一個時間量，它的行程就被搶走並且放回就緒佇列之中。RR 是一種可搶先的排班演算法。

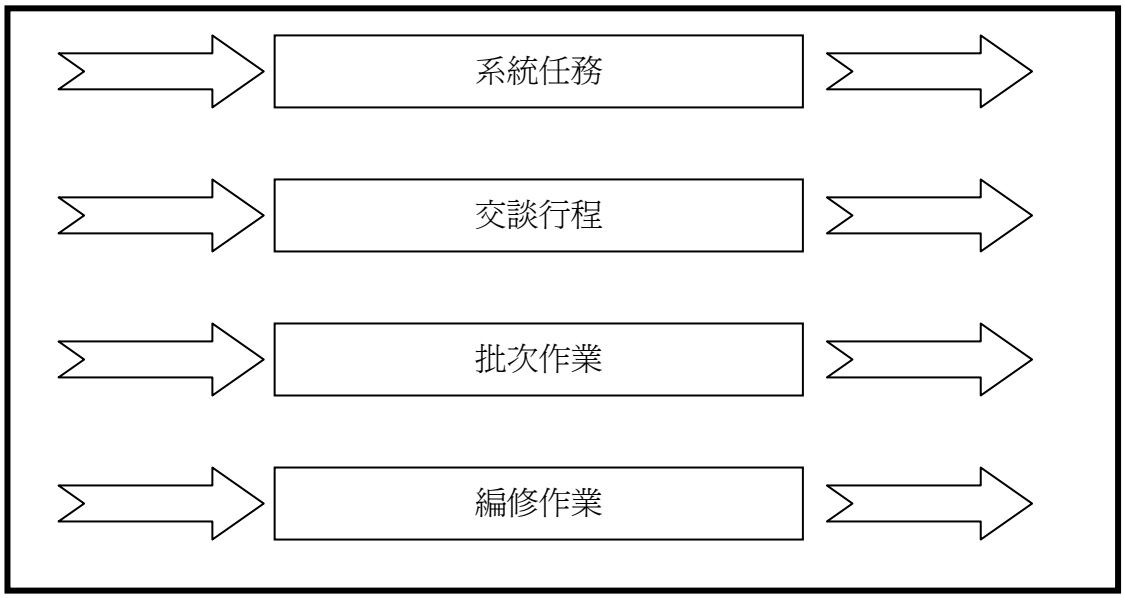
RR 方法的效能決定於時間量的長短上。在一種極端情況下，如果這個時間量非常大，那麼 RR 就變得跟 FCFS 排班一樣。另一種極端是時間量非常小，這種情況的 RR 又稱為處理機共用(Processor sharing)法則。在理論上這種方式使得系統中的數個行程都覺得自己擁有一個處理機。

然而在軟體方面我們也需要考慮在 RR 排班法則性能之中的”內容轉換(context switch)”之影響，也就是行程交接 CPU 所多花費的時間。因此我們要求時間的配額應該比內容轉換的時間長，如果內容轉換時間是時問量的 1/10，那麼就大約有 1/10 的 CPU 時間花在內容轉換的工作。

2.3.1.5 多屬佇列之排班方式(Multilevel Queue Scheduling)

MQS 排班法則是根據行程的性質將它們分成幾個不同的小組。例如，最典型的分類方法是區分為前景(foreground)和背景(background)，分別代表交談式和批次作業的行程。遺兩種行程有截然不同的反應時間，因此所需要的排班法則也完全不同。此外，前景行程的優先順序可能就高於背景行程。

MQS 將就緒佇列區分為多個獨立的佇列如下圖。行程永遠指定到某個佇列中，一般都是用行程的性質決定。每一個獨立的佇列都分別有它的排班法則，例如前景的行程和背景的行程佔不同的佇列，而前景佇列可能是用 RR 排班，背景佇列則可能是用 FCFS 方式排班。



此外，在這些佇列之間，仍然需要一個排班法則。通常都是採用固定佇列優先順序的方法。

舉個例子，四個不同佇列的多層佇列排班方式之例子如上圖，

- 系統任務
- 交談式行程
- 批次處理
- 編輯作業

每個佇列絕對都比較低佇列的優先順序高。例如批次作業佇列的行程，除非系統行程、交談式行程以及的佇列都已經空了，否則是輪不到它執行的。如果在批次作業行程的執行過程當中，有交談式行程進入就緒佇列，批次處理行程也會讓它

優先使用 CPU。

另外一種方法，就是分配每個佇列一段 CPU 時間，每一個佇列再各自安排佇列上工作的執行順序。

2.3.1.6 多屬回饋佇列之排班方式(Multilevel Feedback Queue Scheduling)

在多層佇列排班法則之中的每一個行程，在進入系統之後就分派到某一個固定佇列而都不能轉移到別的佇列。行程不能互相調移，因為每一個行程的性質是不會改變的。這個設定可以降低排班的時間負荷，但是相對比較沒彈性。

MFQS 法則允許行程在佇列中移動，它是利用不同 CPU 分割時段的特性，界分不同等級的佇列。如果一個行程需要太長的 CPU 時間，就會排到低優先權的佇列。高優先佇列通常排的都是 I/O 和交談式行程等。同樣地，在低優先佇列等候太久的行程，隨著時間的增長，也會漸漸地移往高優先佇列。這是一種避免飢餓的老化形式。

一般而言，多層回饋佇列的排班法則都是依據以下幾個參數決定：

- 佇列個數。
- 每一個佇列的排班法則。
- 決定什麼時候把行程提升到佇列的方法。
- 決定降低高優先佇列的行程到下層佇列時機的方法。
- 當行程需要服務時，決定該行程進入那一個佇列的方法。

MFQS 看起來似乎是最通用的一種 CPU 排班法則。理論上它可以設計來適應任何一個特定系統，但是我們仍然需要利用一些方法來決定這些重要參數的最佳值，以便設計最佳的排班法則，因此雖然多層回饋佇列是最通用的一種排班法則，但是它卻也是最為複雜的一種。

2.3.1.7 多元處理機的排班問題 (Multiple Processor Scheduling)

如果一套系統具有多個處理機，則其排班法則相對的就要複雜多了，而且跟單一處理機系統一樣，仍然沒有最佳法則。在系統中，處理機在功能上都相同時稱為同質系統 (homogeneous)，任何空閒狀態的 CPU 都可以拿來執行佇列中的任何行程。但是，即使在同質的多處理機上排班也有所限制，如果有一個 I/O 裝置連接到一個處理機的私有匯流排，那麼希望使用該裝置的行程就必須排去該處理機執行，否則永遠無法使用該裝置。

如果系統中有多個同質的處理機存在，則可利用負擔共攤〈load sharing〉的方法。我們一樣可以為每一個處理機準備一個獨立的佇列，但是這樣可能會造成有些佇列是空著。為了避免這種情況，可以另外利用一個共用的就緒佇列，所

有行程進來都先在這個佇列等候，那一個處理機有空，就安排一個行程過去。

我們可以有兩種排班方式。一種方法是對稱多元處理(Symmetric Multiprocessing, SMP)，也就是每個處理機自行解決排班問題，每個處理機檢查共同的就緒佇列並且挑選一個行程執行。非對稱多元處理(symmetrical multiprocessing)則是把所有排班的決定、I/O 處理，以及其他系統的動作都交由一個單一的處理機處理，他的處理機則執行使用者程式碼。

2.3.1.8 即時排班(Real Time Scheduling)

即時系統可分為兩種型式，一為硬式即時系統，另一則為軟式即時系統。硬式即時系統 (hard real-time system) 必須在一定量的時間裡完成任務。資源預約(resource reservation)意指排班程式接受一個行程，並保證這個行程可以準時完成；或是拒絕這個行程的要求。這種保證要求排班程式正確地知道每一種型式的作業系統函數要花多少時間，執行這種保證對於擁有次要記憶體或虛擬記憶體的系統絕不可能達到，因為遺些子系統會造成執行某一特殊行程不可避免和不可預期的時間量變化。

軟式即時系統 (soft real-time system)的限制比較少。它要求重要行程優先權高於其他行程。但是加入軟性即時功能到一個分時系統可能造成資源分配的不公平，而且可能造成某些行程較長的延遲，或者甚至於產生饑餓(starvation)的現象。所以實現軟性即時功能需要仔細設計排班程式以及作業系統。

首先，系統必須必須要有優先權排班的功能，而且即時行程必須列為最優先且不會隨著時間而改變。另外，分派程式的潛伏期也必須很短，讓即時行程的的開始執行時間更短。

我們可以不讓即時行程老化輕易地達成第一項性質，來保證這些行程的優先權不會改變。但是在大多數的作業系統裡，都是被迫等待系統呼叫完成或是 I/O 等待發生時才可以做內容轉換。這種系統的分派潛伏期可能很長，而有些 I/O 裝置速度也很慢。

為了降低分派的潛伏期，我們必須允許系統呼叫能夠被搶先。有一些方法能完成這項目標。一種方法是加入可搶先點(preemption point)，這些點會檢查是否有更高優先權的行程需要執行。如果是，內容轉換就發生。當高優先權的行程結束，被中斷的行程就繼續執行原先的系統呼叫。

另一種方法是讓整個核心都成為可搶先。使用這種方法，核心永遠可以被搶先，因為任何核心正在更新的資料都被保護，以防止被高優先權的行程修改。

2.4 執行緒的排班(Thread Scheduling)

在行程模式中導入執行緒的觀念可以讓單一行程可以有許多控制的執行緒。另外也區分了出使用者階層(user-level)和核心階層(kernel level)的執行

緒。使用者階層的執行緒是由一個執行緒程式庫所管理，系統核心並不會知情。使用者階層的執行緒要在 CPU 上執行時，最終都會對映到核心階層的某一相關執行緒，雖然此項對映可能是間接的並且只使用到輕量級的行程(LWP)。使用者階層和核心階層的執行緒之間的差別之一在於它們被排班的方式不同。執行緒程式庫安排使用者階層的執行緒在可以獲得的 LWP 上執行，這技巧稱為行程的區域排班法(process local scheduling)。反之核心系統使用系統全域排班法(system global scheduling)決定那一個核心執行緒被排班。

不同的執行緒程式庫是如何區域性地安排執行緒執行主要是軟體程式庫的工作，而非作業系統。

第3章 記憶體管理與虛擬記憶體

任何能支援同時一個以上使用者使用之作業系統，皆必須提供一種能在許多並行處理的行程間共用中央記憶體的方法。許多多工程式

及多行程處理系統將記憶體分成許多分割區，而每一個行程各被指定一塊不同的分割區，這些分割區的大小和位置是可以改變的。

為了要實現在執行性能上的增加，我們必須在記憶體中放入許多行程，而且必須共用記憶體。記憶體是系統一個很重要的部分。記憶體包含一個大型的字組或位元組陣列，各字組和位元組都有自己的位址，CPU 根據程式計數器的數值到記憶體的位址擷取指令。這些指令可能會對於特殊記憶體地址造成一些額外動作。

3.1 記憶體管理概述

使用者行程可能載入到實體記憶體的任何部份。電腦的位址空間是由 00000 開始計位，使用者行程的第一個位址也不見得就是 00000；這種安排方式對於使用者程式使用的位址會有很大的影響。在一般情況下，使用者的程式在被執行前要通過好幾個步驟。鏈結編輯程式(linkage editor)或載入程式(loader)再依次將這些可重新定位的位址鏈結至絕對位址。每一次定位的工作都是從一個位址空間對應到另一個位址空間。

由 CPU 所產生的地址通常稱為邏輯地址(logical address)，而記憶體單元所看到的地址(也就是載入到記憶體的記憶體地址暫存器之數值)通常叫做實體地址(physical address)。但是，一些地址鏈結的技巧卻會造成邏輯地址和實體地址不一樣。在這種情況下，在程式執行時，由虛擬地址對映到實體地址的工作就由記憶管理單元(memory management unit，簡稱 MMU)來執行，MMU 是一個硬體裝置。

行程的程式和資料都必須在實體記憶體執行。但是行程的大小受限於實體記憶體大小，要得到比較好的記憶體空間使用效率，有一種稱之為動態載入(dynamic loading)的方法。使用動態載入的時候，常式只在被呼叫的時候才會載入。動態載入的觀念是將所有程式，以可重定位載入的格式儲存在磁碟內。主程式儲存在主記憶體內，當需要呼叫其他程式時，如果此程式已經存在記憶體內，便可以直接使用；如果不是，便呼叫重定位鏈結載入程式(relocatable linking loader)，將所需要的程式載入記憶體內，並同時更新地址表的內容。動態載入最大的優點，便是不使用到的程式部份不會被載入記憶體內，所以不會

浪費空間。尤其是對於某些錯誤情況處理的大量程式碼相當有效。

動態鏈結與動態載入不一樣的是，鏈結被延後至執行時才發生。此項特色通常在系統程式庫中使用，像是語言副程式庫。若沒這項功能，在一系統內的所有程式都必須保有一份其所需程式的複製。如此便浪費許多磁碟和主記憶體的空間。若採用動態鏈結，在程式參用程式庫副程式時，會留下一個記號，該記號是用來指示如何去找尋適當的記憶體常駐程式庫副程式，或是如何載入程式庫副程式。

當執行到有記號的地方時，會先檢查所需要的程式是否已經存在記憶體中。如果該程式不在記憶體中，程式便會將它載入。也就是說，當該記號被執行時，便會將本身之程式以要載入的程式之位址取代並執行該程式。因此，下次再度過到該段程式時，便可直接執行該程式，而不用再度進行鏈結。在此種方法運作下，所有之行程就可以共用一份相同的程式庫便可。

動態連結與動態載入不一樣，它需要靠作業系統的一些幫助。如果記憶體中的行程是有保護的，彼此不能干擾，則作業系統就是唯一能檢查的便是，所需要的程式是否在另一行程的記憶體空間，以及允許數個行程能夠存取相同的記憶體空間。

3.2 記憶體空間置換(Swapping)

一個行程必須放在記憶體內中才能執行，然而一個行程可能會被置換出記憶體。多重規劃程式都是以依序循環排班方式安排 CPU。當時間量到了時，記憶體管理程式便會將剛剛結束的行程置換出去，並換入該區間等待中的另一個行程。在這個時候 CPU 排班程式可以同時安排時間量給其他的行程。當一個行程使用的時間量到了時，將置換另一個行程執行。理想上，記憶體管理程式能快速地置換行程，使 CPU 排班程式在重新安排排程時，仍然隨時有行程在記憶體內等著執行。

置換的過程需要有其他儲存體的配合，通常都是磁碟。這些備份儲存體必須具有相當大的容量，這樣才能放入所有使用者程式的拷貝。CPU 排班程式決定了下一個執行的行程之後，便會交由分派程式(dispatcher)來處理。分派程式首先查核這個等待執行的行程是否在主記憶體內，如果不是，而且記憶體中空間並不足夠它使用，分派程式就會將主記憶體中的一個行程置出(swap out)，再置入(swap in)所要執行的行程。

置換系統在內容轉換時都會浪費很多的時間，為了有效使用 CPU，我們當然希望每個行程執行的時間都比起置換的時間要長比較好。我們可以發現，就是絕大部分置換的時間都是在資料傳遞上。因此，如果我們能夠知道一個行程真正使用的位置和大小，而不只是知道用了多少記憶體的?空間，那麼在置換的過程中，便可以只置換實際要用到的部分而不需要全部載入，這樣就可以節省不少時間。

當然置換方式有一些限制，最重要的是，如果我們想要置換一個行程，我們必須確定它是完全的閒置著。

3.3 記憶體配置(Contiguous Memory Allocation)

主記憶體必須容納作業系統和不同的使用者行程。一般而言，如果有數個使用者的行程需要同時存放在主記憶體中，我們必須考慮如何將可用的記憶體分配給在等待被載入到主記億體的不同行程。

3.3.1 配置方法

有一種最簡單的記憶體配置方法，就是把主記憶體分成一些固定大小的分割(partition)。每個分割都有一個行程要執行，因此，多元程式規劃的程度往往會受分割的數目所影響。當一個分割使用時，會從輸入佇列中挑選出來一個行程並且載入此分割。當它執行結束時，此分割就可以讓其他行程使用。

在任何時間，都一個可用區間容量的串列和一個輸入佇列。作業系統會依照排班法的順序來處理行程，分配給每一個工作記憶空間，直到所剩下的空間不夠下一個行程的需求使用時。作業系統必須決定，是要先跳過這個行程，先處理其他的行程，還是等到有足夠用的空間出現。這個決定就會牽涉到 CPU 排班程式的問題了。通常在記憶體內，隨時都有一群不同容量的可用區間散佈在各處。當一個行程進入系統並要求記憶體空間時，我們便可在記憶體內找到一塊夠用的位置分配給該行程。

有幾個解決方法可以決定那個區間最適合拿來分配。最先配合(first-fit)、最佳配合(best-fit)及最差配合(worst-fit)是最常用來選出可用區間集合中一個空白區間策略。

- 最先配合：配置第一個可以使用的區間。搜尋的起始位置通常是第一個可用的區間或是前一個可用區段的結束位置。
- 最佳配合：在所有容量夠大的區間中，將最小的一個區間分配給行程。但是，我們通常必須搜尋完整個串列才能找到符合這項策略的區間。最佳配合法會找出最小的剩餘區間。
- 最差配合：將最大的區間配給行程。如果區間串列並未依照容量的大小順序排列，我們必須搜尋整個串列。通常我們也要搜尋完整個串列才能找到符合這項策略的區間。

根據測試，在時間和記憶空間的使用率方面，最先配合與最佳配合都比最差配合法要好。單就記憶體空間使用率來看，最先配合法和最佳配合法是差不多的，但是最先配合法的執行速度通常要快些。

3.3.2 斷裂 (fragmentation)

這些法則都會遭遇到記憶體斷裂的問題。由於行程的載入和置出，可用的記憶空間將被劃分為許多小區間。當有足夠的總記憶體空間得以滿足要求，而這些區間卻是非連續的時候，此時外部斷裂(external fragmentation)的現象便發生了，儲存體被分成了許許多多的小區間。

多重配置區域的另一問題，就是內部斷裂(internal fragmentation)。現在記憶體內剩下一塊 20000 位元組的區間，而接著的行程要求 19998 個位元組的位置，如果我們按其要求配置實際的位置，則會剩下兩個位元組的一個小區間，為了要記錄這塊小區間，我們所要花費的額外功夫，要遠大於這塊位置的價值，因此在這種情況之下，我們乾脆比實際需要多配置一點區間。而這之間的就形成了內部斷裂—小部份內部的記憶體沒有被使用到。

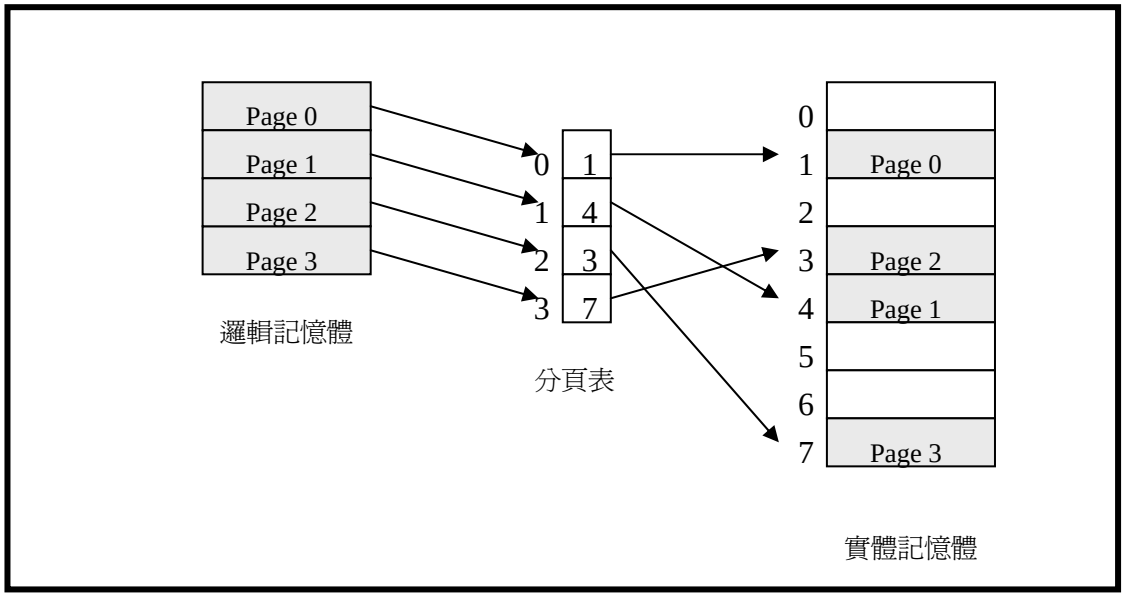
解決斷裂的問題，最常用的方法就是聚集記憶體內零零散散的可用空間，使成一大區間。但是聚集的動作並不是隨時可以進行的，如果重定位的動作是即採用靜態重定位的方式，則聚集的動作便不可行；只有在採用動態重定位方式時，才能適時地聚集分散的區間。外部斷裂的問題的另一個解決辦法就是允許邏輯地址空間不連續，而只要有足夠的實體記憶體就允許這個行程被配置。實現這種解決方法有兩種：分頁和分段。

3.3.3 分頁(paging)

分頁這種技巧允許行程之邏輯記憶體空間可以不連續。分頁法避免了把不同大小記憶體區塊放回到儲存裝置時所遇到的問題。因為分頁法優於一般的記憶體管理方法，所以在很多的作業系統中都被使用。

實體記憶體被分割為許多稱為欄(frame)的固定大小區段。邏輯上的記憶體也被分割為大小相同稱為頁(page)的區段。當一個行程要被執行時，它的頁就被由儲存裝置中載入到任何可用的欄中，而儲存體被分割為其小與記憶體的欄一樣固定大小之區段。

CPU 產生的位址會分為兩個部份：一個是頁數(p)、一個是頁偏移量(d)。頁數(page number)是用來指向分頁表的索引。在分頁表中存有每一頁在實體記憶體上的基底位址。這個基本位址再和頁偏移量結合在一起而定義出送往記憶單位中的實體記憶體位址。記憶體的分頁模式如下圖所示。



頁的大小(和欄的大小)是由硬體來決定。一頁的大小一般都是 2 的次方，其範圍可自每頁 512 字組至 16 百萬字組，依著電腦架構的不同而異。
分頁本身就是種動態重新訂位的型式。每一個邏輯上的位址都被分頁用的硬體對映到實體位址上。

外部斷裂的存可用下列的分頁方法解決：任何可用欄都可以分配給行程使用。但是這樣卻會有產生出一些內部斷裂。分配都是以欄為單位在做的，如果一個程式對記憶體空間的需求並不是剛好是一頁的倍數，那麼被分配的最後一欄就不可能全佔滿了。舉例來說，如果一頁有 1024 個字組，那麼 20504 個字組的行程就需要 20 頁外加 24 個字組。當然要配給它 21 個欄才可以，而造成了(1024 - 24) = 1000 個字組的內部斷裂。最差的情況就是，一個行程需要 n 頁外加 1 個字。那就得分配 n+1 個欄，造成幾乎等於一整個欄的內部斷裂。

如果行程大小和頁的大小無關，我們將可預料每個行程有半頁的內部斷裂。所以我們可以嘗試使用較小的尺寸來分頁。然而這會造成分頁表額外的負擔，而當每頁的大小增加時，負擔就降低了。另外，當傳送的資料量增加時，磁碟的輸入和輸出會比較有效率。通常，當行程、資料集、和主記憶體變大的時候，每頁的大小就必須跟著變大。現在，一般而言每頁都是 2K 和 8K 位元組。

通常每一個分頁表長度為四個位元，但是分頁表大小是可以改變的。32 位元分頁表可指向一個 2^{32} 實體分頁欄。如果一個欄為 4K，則 4 位元分頁表系統可以定址 2^{36} 位元的實體記憶體。

分頁法中有一項非常重要的觀念，那就是使用者對記憶體的看法與實際上記憶體實體之間會有區別。使用者的程式認為記憶體是一個連續的空間，並只保存了這一個程式。事實上，使用者的程式是散佈在整個實體記憶體上.並且擁有其他的程式。使用者對記憶體的看法與實體記憶體之間的差異是藉著硬體位址的轉換或對映，來使其歸於一致。邏輯上的位址轉換成實體上的位址。使用者無法看到這項對映的工作，它乃是由作業系統所控制的。根據定義，使用者行程無法使

用不屬於它的記憶體，它不能定址到其分頁表以外的記憶體，每一個行程的分頁表只包含自己所擁有的頁數。

由於是由作業系統管理實際的記憶體，因此它必須瞭解實際記憶體的分配詳情：那一些欄被分配出去了，那些欄還可以使用，一共有多少欄存在，等等。這些消息一般都保存在一種稱為分欄表(frame table)的資料結構中。每一實際頁欄在分欄表中都有一個單元，用來標示到底該欄是否可用或已被配置出去了，如果已被分配出去了，那麼分給那一頁那一個程序都會登記在其中。

此外，作業系統也必須瞭解在使用者空間中操作的程序，並且所有邏輯位址都必須被對映到產生實際的位址。如果使用者發出了一個系統呼叫(例如，做I/O)，並且以參數方式提供一個位址，那個位址就必須被對映以產生正確的實際位址。作業系統保存一份每個使用者分頁表的拷貝，正如同保存一份指令計數器及暫存器內容一樣。這項拷貝是當作業系統必須把一個邏輯位址對映成一個實際位址的時侯用來轉換的。它也就被 CPU 分派程式用來在一個行程將被配置 CPU 的時候定義硬體的分頁表。因此分頁法會增加內容轉換的時間。

大部份近代的電腦系統都提供了非常大的邏輯地址空間。在這樣的環境下，分頁表本身變得非常大。當然，我們並不希望把分頁表全部連續地放在記憶體中。這個問題有一種簡單的解決方法，就是把分頁表分成較小的片段。有幾個方法可以達成以上的要求。

其中一種方法就是使用兩層的分頁技巧，也就是分頁表本身也由分頁的方法得到。甚至有些系統還提供三層和四層的方法。

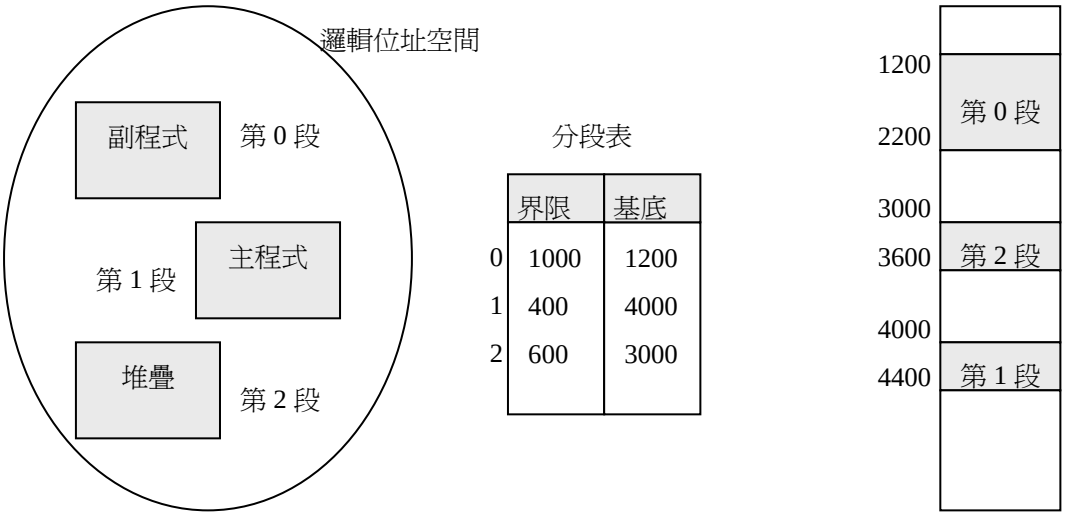
多層分頁對系統性能有何影響呢?假設每一層的分頁表皆存在記憶體中，則轉換邏輯地址為實體地址需要 4 次記憶體的存取。所以現在一次記憶體的存取需要原先 4 倍的時間。但是，快取記憶體可以增加效率，使整體性能保持在合理範圍。假設快取記憶體的命中率是 98%，則

有效存取時間 = 0.98 * 120 + 0.02 * 520
= 128 個十億分之一秒

因此雖然多加了額外層次的分頁表查詢，我們只讓記憶體存取時間慢 23%而已。

3.3.4 分段(Segmentation)

在分頁法中一項重要觀念就是使用者對記憶體的看法與實際記憶體之間是不一致的。使用者的觀點是對映到實際記憶體上。這項對映法允許在邏輯上記憶體與實際記憶體之間存有差異。



上圖為分段法的一個例子。一般人都認為一個系統的使用者或程式設計師並不會把記憶體當作一線性陣列的文字，相反地，使用者反而喜歡把記憶體看成是由許多不同大小的分段所組成的，其間並沒有什麼次序。試想你將會把一個正在寫的程式看成如何？你會認為它是由一個主程式和許多函數或模式所組成嗎？其中也有許多不同的資料結構：表格、陣列、堆疊、變數等等。這些模式或資料單元都由其名稱而定。

分段法就是一種提供這種使用者對記憶體看法的記憶體管理技術。一個邏輯位址空間就是許多分段的組合。每一段都有一個名稱和一個長度。位址指定了分段名稱和在分段中的偏移量。因此使用者用兩個量來指定每個位址：一個分段名稱和一個偏移量。

為了執行上的便利，每一段都有一個號碼，而且在提到某一段時要指明其號碼而不指其名稱。因此，邏輯地址由以下的所組成：<區段號碼，偏移量>。一般來說，使用者的程式被組譯，組譯程式會自動地為反映該輸入程式而構成一個分段。

分段法的另外一項優點就是分段上相關的保護措施。因為分段在定義上就代表著一個程式的一部份，而在分段中的所有單元都被以相同的方式使用。因此我們在某些分段中是指令，也有些分段中是資料。在近代的電腦結構中，指令是不可以被使用者修改的，因此指令分段可以定義成唯讀或者唯執行的方式。記憶體對映用的硬體將會核對在分段表中每個單元的保護位元，以防止對記憶體的不正當存取。譬如企圖對一個唯讀分段寫入資料或是將一個只執行的分段當做資料使用。管理記憶體的硬體將一個陣列放在它的分段中，並且會自動地檢查該陣列的指標是否合法以及是否沒有超出陣列的界限。因此許多程式的錯誤在它們引起非常嚴重的損壞之前都會被這些硬體偵測到。

分段法的另一種優點包括資料或程式碼的共用。每個行程都有一個分段表，當某個行程擁有 CPU 之後，分派程式就定義一個分段表。當在兩個不同行程的分

段表中的單元指向同一實體位址的時候，那一分段就被共用了。共用的部份發生在分段階層上，因此可以把任何要共用的資料定義成一個分段。既然許多分段都可以共用，那麼一個由許多分段所組成的程式就可以被共用了。

舉例來說，在一個分時系統中使用本文編輯程式，一個完整的編輯程式可能非常大，乃由許多分段所組成。這些分段可供所有使用者共用，限制實體的記憶體必須供應編輯的工作。我們只需要一份拷貝，而不是把編輯程式拷貝 n 份。對於每個使用者而言，我們仍舊需要不同的、單獨的分段來儲存內部變數。這些分段就當然是不可共用的了。

當然也可以只共用某些程式而已。舉例來說，一般共用副程式套裝程式庫就可以供許多使用者使用，只要把它們定義成可共用的唯讀分段就可以了。例如兩個 FORTRAN 程式可以使用相同的副程式 *Sqrt*，但是實際上只需要一份 *Sqrt* 常式的拷貝就可以了。

雖然這種共用看來十分簡單，但是卻有一些精細的問題必須考慮。程式碼分段一般包括了對其本身的參考。舉例來說，一個條件式的跳越都有一個轉換位址。這個轉換位址就是一個分段號碼和一個偏移量。轉換位址的分段號碼就是程式碼分段的分段號碼。如果我們要共用這一段，所有要共用的行程必須定義共用的程式碼分段都有相同的分段號碼。

唯讀的資料分段(沒有指標)可以用不同的分段號碼來參考。因為許多程式碼分段並不直接地參考其本身。舉例來說，條件式的分枝是以目前程式計數器的偏移量或是相對於一個存放有目前分段號碼的暫存器來定出分枝位址(branch address)，將會允許程式碼避免直接參考目前分段號碼。

長程排班程式必須為使用者程式找出其所有的分段並且配給它們記憶空間。這個情況和分頁法很相似，唯一不同的是分段的長度為可變化；而每一頁的大小卻都一樣。因此就如同可變分割一樣，記憶體的分配是一種動態儲存器分配，可以使用最佳適合或最先適合演算法來解決。

分段法在所有可用記憶體區段都太小了而無法提供成為一個分段的時候，會造成外部斷裂。在這種情況下，行程就只能等待到有更多的記憶體可以使用為止，或是使用聚集法來造成一個較大的洞。因為分段法在特性上就是一種動態重新定址演算法，我們可以在需要的時候將記憶體聚集在一起。如果 CPU 排班程式必須為了記憶體配置的問題而等待某個行程，它可以跳過 CPU 佇列，去找尋一個較小，且優先權較低的程式來執行。

分段方式所造成的外部斷裂有多嚴重呢?用聚集法的長程排班有助於此問題嗎？這些問題的答案主要決定於分段的平均大小。在一個極端的情況中，我們可以把每個行程定義成一個分段大小，這方法就是可變分割。另外一個極端的情形是將每個字組自成一個分段，並且分別地重新定位。這種方法可以把所有的外部斷裂都消除掉。然而每一字組需要一基底暫存器做為重定位，會要兩倍記憶體的使用。一般而言，如果分段的平均尺寸都很小，那麼外部斷裂也會很小。因為個別的分段要比整個行程小許多，所以它們更適合放在可用的記憶體區段中。

事實上，分頁法和分段法各有其利弊，所以產生了一種合併的方法稱做分頁式的分段(Segmentation With Paging)。一個行程的邏輯位址被分成兩部份。第一部份是由該行程私有的區段所組成；第二部分是由所有行程共同使用的區段。而其最主要的觀念就是將每一分段再予以分頁，而這個方法將更有效率。

3.4 虛擬記憶體概述

虛擬記憶體是一種允許行程可以不必完全存在記憶體中的情況下被執行的技術。這個辦法的主要可見優點就是使用者程式可以比實體記憶體大。除此之外，它將記憶體抽象成一個非常大且一致的儲存陣列，當使用者從實體記憶體觀察，它是和邏輯記憶體分開的。這項技術免除程式設計師對記憶體儲存限制的憂慮。但是，虛擬記憶體在製作上並不是很容易的事，並且可能在不注意的使用下顯著地降低了系統的性能。

記憶體管理演算法確實有其必要性，因為它有著一項基本需求：被執行的指令必須在實體記憶體中。滿足這項要求的第一種做法是，把整個程式的邏輯地址空間都放在實體記憶體中。重疊和動態載入可以減輕這項限制，但卻需要程式人員的特別注意，並付出額外的努力。事實上，在很多情況下，整個程式的載入記憶體是不必要的。

雖然有時我們也需要用到整個程式,但畢竟不會同時需要整個程式的任一部份。能執行只有部份程式在主記憶體中的能力有著多項的優點：

- 程式不再被可用實體記憶體的總量所限制。使用者可以為一很大的虛擬位址空間寫作程式，簡化程式寫作的工作。
- 各個使用者程式會佔有較少的實體記憶體，更多程式可以同時一起執行，相對的，CPU 的利用率和輸出量也隨之增加。
- 載入和置換各個使用者程式入記憶體的 I/O 較少。

因此，只執行記憶體中的部份程式，對系統和對使用者都有好處。

虛擬記憶體乃是使用者邏輯記憶體與實體記憶體之間的分隔。以這種方式我們可以在一個比較小的實體記憶體提供程式設計師大量的虛擬記憶體。

虛擬記憶體一般是以需求分頁法來執行，但也可以在分段法的系統中執行。許多系統中使用分頁式的分段法，也就是將一個分段再分為許多頁。需求分段法也可用來提供虛擬記憶體。可是分段替換演算法卻比頁替換演算法複雜得多，主要是因為分段具有可變大小的特性。

3.5 需求分頁 (Demand Paging)

需求分頁法就像一種使用置換法的分頁系統。行程存放在輔助記憶體。當我們要執行某個行程的時候，就把那個行程置換至記憶體中。但是，我們並不是把整個行程都置換進來，而是我們使用一個懶惰(lazy)置換程式。懶惰置換程式只有當需要某一頁的時候才把該頁置換進來。置換程式會避免將那些用不到的頁置換進來，而降低置換時間和所需的實體記憶體數量。

如果程式想要使用一頁尚未被載入記憶體的資料時會發生什麼情形呢？如果有程式想要存取尚未載入記憶體的某一項，就會產生分頁錯誤(page fault)。分頁用的硬體在經由分頁表轉換位址的時候，將會注意到不可使用的位元已經被設定了，這會引起因為無效位址的錯誤形式而對作業系統發出插斷。這個插斷是因為作業系統沒有把行程所需要的頁載入記憶體所造成的，而不是試圖去使用一個不合規定的記憶體位址所造成的無效位址錯誤。因此我們必須更正這個失察的地方。過程十分直接。

理論上，某些程式在執行每一個指令時都會存取一些新的分頁，每個指令都可能會造成分頁錯誤。

但是，程式通常具有參考之局部區域性，而使需求分頁有不錯的性能。需求分頁對於電腦系統的性能有很大的影響。

3.6 分頁替換 (Page Replacement)

如果我們增加多元程式規劃的程度，我們就過度配置記憶體了。當執行一個使用者行程的時候，出現了一個分頁錯誤，硬體就對作業系統發出插斷，由作業系統對其內部表格以瞭解這是一項分頁錯誤而不是一項非法的記憶體存取。作業系統確定所要的那一頁存在磁碟什麼地方，但是卻發現在可用空白欄的表中已無空白欄了；所有記憶空閒都已被使用了。作業系統在此刻有許多選擇方式，它可以終止該使用者行程。可是，需求分頁法就是作業系統用來改善電腦系統使用率與工作量的。

3.6.1 FIFO 法則 (FIFO Page Replacement)

最簡單的頁替換演算法就是先進先出 (First In ,First-Out ,FIFO)。與 FIFO 有關連的就是當每一頁被載入記憶體時的時間。當要替換一頁時，我們就去選擇最“老"的一個。要注意的是，我們並不一定要採用記錄某頁被載入時間這種方法。我們也可以建立一個 FIFO 的佇列來掌握在記憶體中的所有頁。要替換的時候就把佇列頭上的那一頁替換掉。當一頁被載入記憶之後，我們就把它插入佇列的尾端。

FIFO 替換演算法非常容易瞭解與設計。可是，它的性能並不是很好。替換頁可能是一個很早以前使用的創始模式，並且現在已經不需要了。此外，它可能包含了一個經常使用的變數，該變數很早就設定了並且始終在被使用。請注意，如果我們所選來要被替換掉的那一頁還正在使用中，該程式仍將正確地工作下

去。當我們把那一頁換成新頁之後，再對該頁參考就會立即產生錯誤了。此刻只有把別的替換出去，才能把那一頁放回記憶體中。因此，如果替換選擇上很差勁，那就會增加分頁錯誤的比率並且減緩了程式的執行速度，但是並不會造成不正確的執行。

3.6.2 最佳頁替換 (Option Page Replacement)

最佳頁替換演算法是所有演算法中分頁錯誤此率最低的一種。目前存在的最佳頁替換演算法稱為 OPT 或 MIN。它的說明很簡單：把未來最長時間之內不會被用到的那一頁替換掉。

使用這種頁替換演算法可以保證在固定欄位數量之下得到最低的分頁錯誤此率。很不幸的是，最佳頁替換演算法在執行上非當困難。因為靈它必須預先知道參考串的內容。因此，最佳演算法主要是用在比較性的研究中。

3.6.3 LRU 頁替換 (LRU Page Replacement)

如果最佳演算法並不是可行的，也許我們可以採用一種近似的方法。FIFO 和 OPT 之間最大的區別就是 FIFO 所使用的是某頁被載入記憶體的時間；OPT 所使用的是某頁被使用的時間。如果我們將最近的過去當作最近的將來，那麼我們將替換的是最久未被使用的那一頁。這就是近來最少使用(least recently used, LRU)演算法。

LRU 替換法與每一頁前一次被使用的時間極有關聯。當有一頁必須被替換的時候，LRU 就選擇最久未被使用的那一頁。這就是最佳頁替換演算法在時間上改為向後看，而不是向前看所得的結果。

- LRU 經常被當作一種頁替換演算法使用並且被認為非常良好。

3.6.4 LRU 近似頁替換法 (LRU Approximation Page Replacement)

很少電腦系統提供足夠的硬體來支援真正的 LRU 頁替換。有些系統根本不提供硬體支援，所以必須採用其他的頁替換演算法(例如 FIFO)。不過，許多系統都會以參考位元(reference bit)的方式提供一些幫助。任何時候當某頁被參考時，此頁的參考位元會被硬體設定。在分頁表裏面每一單元都有一個位元與其有關。最初，作業系統把所有的位元都清除掉了(定為 0)。當某個使用者行程執行的時候，凡是與被參考的頁有關的位元就在硬體上被設為 1。經過一段時間之後，我們就可以藉著核對那些參考位元來確定那些頁曾經被使用過了，那些並未用過。我們並不知道被使用的次序，但是知道那些被使用過了而那些還沒有被使用過了。這種部份排班資訊可使許多分頁替換演算法接近於 LRU 替換法。

另外，還有額外的參考位元，定期地記錄參考位元我們可以得到一些額外的次序資訊。每經過一段時間，一個計時器中斷就把控制權轉交給作業系統。作業系統把每一頁的參考位元遞增，那麼我們就可以知道哪一頁在都沒有被使用過、哪一頁用最多、哪一頁用最少。這樣我們就可以把所有具有最小值的頁都替換掉(置換出去)或是使用 FIFO 的方法來選擇其中一頁。用來記綠的位元數量可以改變的，當然，在選擇的時候當然是能促使更換作業愈快愈好。

3.6.5 計數基礎頁替換法 (Counting-Base Page Replacement)

還有許多其他演算法可用來處理頁替換。例如，我們可以使用一個計數器來記錄每一頁被參考過的次數，並發展出以下兩種方法。

- 最不經常使用的 (least frequently used, LFU)頁替換演算法讓次數最少的那一頁被替換掉。
- 最常被使用的 (most frequently used, MFU) 頁替換演算法認為次數最少的頁可能才被載入不久並且將要被使用。

MFU 和 LFU 這些演算法在執行上所花費的代價都相當大，而他們也不很近似 OPT 替換方法。

3.7 欄的配置法則 (Allocation of Frames)

我們如何把數量固定的可用記憶空間分配給不同的行程呢？如果我們有 100 個空白欄和 2 個行程，那麼每個行程可以得到幾個欄呢？

3.7.1 最少量的欄數 (Minimum Number of Frames)

分配上有許多限制，我們無法分配比整個可用欄總數還要多的欄。我們也只有最少量的欄數可被分配。很顯然的，當分配給每個行程的欄數減少之後，分頁錯誤比率就會增加，而降低了執行行程的時間。

除了為這少數欄的分配必須要執行一些工作之外，必須要分配的還有一個最少量的欄數。這個最少量的欄數乃是由指令組結構所定義的。當一個分頁錯誤在一項指令執行完畢之前發生的話，這個指令就必須重新執行一次。一般來說，我們必須有足夠的欄來保存任何單一指令可以參考的頁。

最少量的欄數是由電腦系統所定義。例如，PDP-11 需要在某些位址模式中移動的指令往往多於一個字，因此指令本身可能跨越兩頁之間。此外，兩個運算元可能都是間接參考，一共需要六個欄。在 IBM370 系統中最糟的情況可能就是移動字元的指令。因為這個指令是由儲存體至儲存體，它需要 6 個位元組並且可以跨於兩頁之間。要被移動字元的區段和要被移至的區域也都可以跨於兩頁之間。這個情形也需要 6 個欄。

每個行程最少的欄數乃是由電腦結構所定義，而最大的數量則是由可用的實體記憶體數量所定義。在這兩者之間，我們仍然留有在分配欄數上的很大選擇。

3.7.2 同等分配及比例分配

將 m 個欄分給 n 個行程的最簡單方法就是令每個行程都能分到一樣多的量，m/n 個欄。舉例來說，如果有 100 個欄和 6 個行程，那麼每個行程可以分到 16 個欄。剩下的 4 個欄可以當做一個空白欄緩衝區的庫存。這就稱為同等分配(equal allocation)。

另一種方法是因為瞭解不同的行程將會需要不同的記憶體而設立的。如果在一個擁有 62 空白欄的系統中只有一個 10K 的小型學生行程和一個 127K 的交談式資料庫，那麼把 31 個欄分給每個行程就毫無意義了。學生的程式只需要 10 個欄來使用，所以另外 21 個欄就浪費掉了。

為了解決這個問題，我們可以使用比例分配 (proportional allocation)。我們為每個行程分配可用記憶體時可以按照它的大小分配。在這種方式之下，兩個行程就都按照它們的需要去分享可用的欄，而不是平均分配。

在上述兩種情形(平均及按比例分配)中，分配給每個行程的方式將因為多元程式規劃的程度而有所不同。如果多元程式規劃的程度增加了，每個行程將失去一些欄來供給新行程使用。此外，如果多元程式規劃的程度降低了，那麼已經分配給要離開的行程的那些欄，現在就可以分散給各個仍舊存在的行程了。

3.7.3 全體和區域的配置(Global Versus Local Allocation)

另外一種影響分配給不同行程時所採用方法的重要因素是頁替換。在許多行程競爭欄數時，我們可以把頁替換法分成兩大類；全體替換法和區域替換法。全體替換法允許一個行程從所有欄數中選出一個替換欄，即使目前該正分配給其他某個行程使用中；換言之，一個行程可以從其他行程獲得一欄。區域替換法要求每一行程只能從它自己選出欄。

對於區域替換法而言，一個行程所配置的欄數不會改變。而對於全體替換法，一個行程可能選擇配置給其他行程的欄，因而增加配置的欄數。

全體替換法有一個問題是，一個行程不能控制它自己的分頁失誤率。一個行程在記憶體中所佔有的頁數不僅由該行程的分頁情況來決定，而且也由其他行程的分頁情形來決定。因此，相同的行程卻可能會有極端不同的執行情況(一次只需要 0.5 秒，另一次執行卻可能需要 10 秒)，完全視外在環境來決定。對於區域置換法就不會有此情形發生。在區域置換法下，一個行程在記憶體中所佔有的頁數僅由該行程的分頁情況所決定。區域置換有可能因為頁數不足而延遲行程的執行。因此，通常全體配置會產生比較好的系統效能，因此是較常用的方法。

3.8 輾轉現象(Thrashing)

如果分配給某個低優先權行程的欄數是在電腦結構所需的最少數量以下，我們就必須暫時擱置其執行工作。然後我們就將它所剩下的頁分出去，並且把分配給它的所有欄都釋回。這項規則引伸出了一項置入/置出(swap in/swap out)程次的 CPU 排班。

如果行程沒有得到足夠數量的欄，那很快就會產生分頁錯誤。此刻，它就必須替換掉一些頁。可是，因為它所有的頁都正在使用中，它就必須把稍後將需要的頁替換掉。一般來說，很快地它又出現錯誤，並且不斷地如此。程式不停地出現錯誤，不停地把稍後會引起錯誤的頁替換掉並且再立刻拿回來。

這種非常高速的分頁替換行為稱為輾轉現象。如果某個行程花在分頁上的時間比花在執行上的時間還要多，我們就稱它是處於輾轉狀態。

3.8.1 輾轉現象之原因

作業系統監督 CPU 的使用情形。如果 CPU 的使用率太低了，我們就藉著引入一個新行程來增加多元程式規劃的程度。我們使用一種全體性分頁替換演算法，在替換頁時並不考慮該頁屬於那一行程。現在假設某個行程在它執行過程中進入了一個新的狀況並且需要更多欄以供使用，它開始產生錯誤並且從其他行程處得到一些頁來使用。然而這些行程也需要這些頁，所以它們也將出現分頁錯誤，而又從其他行程處拿一些頁來使用。這些產生錯誤的行程必須使用分頁裝置來將頁置換進來和置換出去。當它們在排隊等待分頁裝置的時候，就緒佇列就空了。當行程都在等待分頁裝置的時候，CPU 的使用率就降低了。

CPU 排班程式發現 CPU 使用率降低的時候，它就增加多元程式規劃的程度。這些新的行程從其他正在執行的行程中搶來一些頁以開始工作，這樣就造成了更多的分頁錯誤，以及一個更長的等待分頁裝置的佇列。結果，CPU 的使用率就更降低了，而 CPU 排班程式就想要再提高多元程式規劃的程度。這樣輾轉現象就出現了並且使得系統的產量突然下降，分頁錯誤則大量地增加。結果造成有效記憶體存取時間也增加了。因為行程把所有的時間都花在分頁上，所以什麼工作都沒有完成。

當多元程式規劃的程度增加了之後，CPU 的使用率也增加了，雖然速度也慢了許多，但是一直增加到一個最大值為止。如果多元程式規劃程度繼續增加，輾轉現象就出現了並且使得 CPU 使用率急速下降，此時，若要使得 CPU 的使用率能增加並且停止輾轉現象，我們就必須降低多元程式規劃的程度。

3.8.2 輾轉現象解決方法

我們可以藉著使用一個區域性或(優先權)替換演算法來限制輾轉現象所造成的影響。若是使用區域替換法，如果某行程開始出現輾轉現象，它就無法從其

他行程處取得自己要用的欄，但這會引起其他行程也出現輾轉現象。如果有好幾個行程都出現了輾轉現象，那麼它們大部份時間將在等待分頁裝置的佇列中。處理分頁錯誤的平均時間將會增加，這乃是因為在等待分頁裝置的佇列中所費的時間平均變長了。因此，即使是沒有出現輾轉現象的行程，它的有效存取時間也會增加。

為了防止輾轉現象發生，我們必須提供行程所需的一切的欄。但是我們怎麼知道它需要多少欄呢？這就有許多方法可以採用了。工作組(working set)策略就是藉著觀察實際上一個行程正在使用那些頁而開始。這個方法定義了行程執行中的局部區域模式(locality model)。

局部區域模式認為當一個行程執行的時候，行程會從一個局部區域移到另一個局部區域。所謂一個局部區域就是一組同時被使用的頁。一個程式乃是由許多不同局部區域組成的，其中可能會有重疊。

3.8.2.1 工作組模式(Working-Set Model)

工作組模式乃是基於局部區域的假設而設定的。這模式中使用一個參數 Δ，來定義工作組欄框(working-set window)。其想法就是去檢查最近 Δ 頁參考。在最近 Δ 頁參考中的頁所組成的集台就是工作組。如果某頁正被使用中，它就會在工作組中。如果它不再被使用了，它將在最後一次被參考之後經過 Δ 個單位時間，從工作組中被剔除。因此工作組乃是一個程式的局部區域的近似狀況。

工作組模式的使用非常簡單。作業系統監督每一個行程的工作組，並且按照其工作組的大小分配給它足夠的欄使用。如果還有足夠的額外的欄可用，就可以啟始另一個行程。如果工作組大小的總和增加了，並且超過了整個可用欄的總數，作業系統就選出一個行程來讓它被暫時擱置。該行程的頁就被寫出去並且將它的欄重新分配給其他行程使用。被擱置的行程可以稍後再重新啟動。

在多元程式規劃的程度儘可能的提高下，工作組策略防止了輾轉現象。因此它嘗試要使 CPU 的使用率達到理想化。

3.8.2.2 分頁錯誤頻率(Page-Fault Frequency)

工作組模式非常有用，但是那卻是用來控制輾轉現象十分笨拙的方法，分頁錯誤頻率(PFF)策略則是一種更直接的方法。

我們要解決的就是防止輾轉現象發生。輾轉現象有高分頁錯誤率，因此我們要去控制住分頁錯誤率。當它太高的時候，我們就知道該行程需要更多的欄了。同樣地，如果分頁錯誤率太低，這就表示該行程擁有太多欄了。我們可以在想要的分頁錯誤率上訂定上限與下限。如果實際的分頁錯誤超過了上限，我們就把另外一個欄配給那個行程；如果分頁錯誤率降到比下限還低的時候，我們就從該行

程處移走一個欄。因此我們可以直接地測量與控制分頁錯誤率以防止輾轉現象出現。

如果是使用工作組策略的話，我們就必須把一個行程暫時擱置起來。如果分頁錯誤率增加了並且沒有空白欄可用，我們就必須選出某個行程並且將它暫時擱置一旁。然後被空出來的欄就可以被分配給分頁錯誤率太高的行程。

第4章 檔案系統

檔案系統提供了作業系統或電腦系統所使用者使用線上儲存和存取資料與程式的功能，檔案系統由兩個不同的部分所組成：一群檔案(每一個檔案都儲存相關的資料)，以及一個目錄結構(directory structure)，此目錄結構對於系統中的所有檔案加以組織，並提供相

關資訊。有些檔案系統還有第三部份，那就是分割(partition)，分割是用來分隔實體或邏輯上一大群成組的目錄。

4.1 檔案概述

電腦可以在一些不同的儲存媒體上儲存資料，譬如：磁碟、磁帶、和光碟。為了讓電腦便於使用，作業系統提供了一種對於資訊儲存的統一邏輯觀點。作業系統由它儲存裝置的物理特性抽象地定義了一個在邏輯上的儲存單位，那就是檔案，檔案是藉著作業系統應對到實體裝置上。

一般來說，檔案代表程式(無論是原始程式或目的程式)和資料。資料檔案可能屬於數字的(numeric)、文字的(alphabetic)、文數字(alphanumeric)或二進位數。檔案可能是自由格式，例如本文檔案(text file)，或是很嚴格要求的格式。總之，檔案就是由其建立者或使用者所定義，具有某種意義的一串位元、位元組、幾行文字或是紀錄。這是一項普通的觀念。

檔案中的資料是由建檔者去定義。許多不同類型的資訊可以存在檔案中，例如：原始程式(source program)、目的程式(object program)、數字資料(numeric data)、可執行程式(executable program)、本文、薪資帳冊紀錄、圖像、錄音等等。檔案根據其型態都有一定的結構。

4.1.1 檔案特性

檔案的命名是為了方便人類的使用，因此通常以檔案名稱來指定某一檔案。檔名通常是一串字元，譬如“config.sys”。有些系統對於檔名的大小寫有所區別，有些系統則視大小寫相同。當一個檔案被命名之後，它就獨立於產生它的行程、使用者或是系統之外。譬如，一個使用者可能會產生檔案“autoexec.bat”，而另一個使用者只要指定檔名，也可以修編同一檔案。檔案擁有者可能把檔案寫到軟碟或磁帶上，然後再從另一系統讀出來，然而檔名依然叫做“autoexec.bat”。

檔案還有一些其他特性，這些特性隨著作業系統不同而有所差異，但通常包括了：

- 名稱(name)　符號式的檔名是唯一用人看得懂的格式儲存。
- 型態(type)　這項資訊對於提供不同檔案型態的系統有需要。
- 位置(location)　一個指標指向該檔案所在裝置的位置。
- 大小(size)　該檔案目前容量的大小(以字元、位元組、或區段為單位)

以及允許以後擴增的最大範圍。

- 保護(protection)　存取控制資訊，控制誰能讀、寫、執行等程式。
- 時間、日期和使用者辨識　這項資訊可以確保(1)產生；(2)上次修改；(3)上次使用資料，以做為保護、安全、以及使用監督。

關於所有檔案的資訊都存在目錄結構中，這些目錄結構也存在輔助儲存體中。每一個檔案都可能佔用 16 到超過 1000 個位元組來記錄這項資訊。對於一個有許多檔案的系統而言，目錄本身可能佔用百萬位元。

4.1.2 檔案型態

設計一套檔案系統以及整個作業系統時的一項主要考慮是,作業系統是否該辨認和支援檔案型態。如果作業系統能辨識檔案型態，它就能以合理的方式操作檔案。譬如，使用者試圖列印二進位目的檔(binary-object)就是一項常犯的錯誤。

實現檔案型態的一種常見技巧是，包含檔案型態做為檔名的一部份。檔名可分成兩個部分—主檔名和副檔名，兩者之間以句點加以分隔。採用這種方式，使用者和作業系統可以從檔名就能分辨出檔案的型態。譬如，在 MS-DOS 下，檔名是由至多有 8 個字元的主檔名，跟隨著一個句點，最後在跟著至多 3 個字元的副檔名。作業系統使用副檔名來辨識檔案的型態，並判斷可對檔案做的操作。譬如，只有副檔名是 .com，.exe 或 .bat 的檔案才可以執行。其中 .com 檔和 .exe 檔是二位元的可執行檔，而 .bat 檔則是包含 ASCII 字元做為作業系統指令的批次檔。MS-DOS 只能辨識一些副檔名，但是應用程式也可以使用副檔名來辨識它們自己有興趣的檔案型態。譬如，組譯器處理副檔名是 .asm 的程式檔，而文書處理程式 Word Perfect 所處理的檔案則是 .wp 的副檔名。這些軟體並不一定非得處理以上副檔名的檔案，但使用者可以在設定檔名時省略檔案的副檔名，應用程式就會自動加上副檔名，並且尋找該檔案。

UNIX 系統不能提供這項特性，因為它使用了一個自然的魔數(magic number)，此魔數儲存在某些檔案的開頭以概略地指出檔案的型態是:可執行檔、批次檔(叫做 shell script)，postscript 檔等等。並非所有的檔案都有魔數，所以系統的特性不能只根據這項資訊。UNIX 系統不會記錄產生程式的名稱。UNIX 也允許用副檔名作提示，但此副檔名並不是強迫，它只是用來幫助使用者決定檔案的型態及內容而已。

4.1.3 檔案結構

檔案型態也可以用來指出檔案的內部結構。進一步而言，有些檔案必須符合一定的結構，作業系統才能夠了解其內容。譬如，作業系統可能會要求一個可執

行檔有一定的結構，以便決定將檔案載入到記憶體的什麼地方，以及第一個指令的位置。有些作業系統將此觀念延伸到一組系統支援的檔案結構，對於這些檔案結構都有特殊的操作來處理這些檔案。

我們可以發現讓作業系統支援多種檔案結構的一項缺點：作業系統的體積非常龐大。如果作業系統定義了五種不同的檔案結構，它就必須包含程式碼以支援這些檔案結構。除此之外，每一個檔案可能都必須定義成作業系統所支援的檔案型態之一。如果新的應用需要的資訊結構不是作業系統所支援，則會產生嚴重的問題。

例如有一套作業系統支援兩種型態的檔案：文字檔案(由 ASCII 字元所組成，並以 CR 和 LF 檔分隔)和可執行的二位元檔案。現在，如果我們希望定義一個密碼檔案，以保護我們檔案不會被未授權的人閱讀，我們會發現原有的兩種檔案型態皆不合適。密碼檔並不是一個 ASCII 文字檔，而是一個隨意的位元組合。雖然它很像是一個二位元檔案，但是卻不能夠執行。因此，我們只有克服作業系統的檔案型態，或者是放棄將檔案編成密碼。

對於一套作業系統而言，找到內部某一特定位置十分的複雜。磁碟系統通常會有一定義完善的區塊大小，這是由磁區(sector)的大小所決定。所有磁碟的輸入/輸出都是以區塊為單位來執行，此區塊就是實體紀錄(physical record)，所有的區塊大小皆相同。實體紀錄的大小不太可能會和邏輯紀錄(logical record)正好相同。邏輯紀錄甚至會有不同的長度。將一些邏輯紀錄組成實體紀錄是常見到的解決方法。

在任何一種情況下，檔案都可以視為一連串的區段，所有的基本輸入/輸出函數皆是針對區段來操作。但以區塊來配置磁碟空間通常都會造成每一個檔案中最後一塊區段的部分空間被浪費掉。如果每一塊區段都是 512 位元組，則一個 1949 位元組長的檔案將會被分配到 4 塊區段(共佔用 2048 個位元組)：最後 99 個位元組就浪費掉。浪費掉的位元組就成為內部斷裂(internal fragmentation)。所有的檔案系統皆有內部斷裂問題；區段愈大，內部斷裂就愈大。

4.1.4 檔案運作

檔案是一個抽象的資料形式(abstract data type)為了適切地定義檔案，我們必須考慮檔案上所表現的操作。作業系統提供系統呼叫以產生、寫入、讀取、重置、刪除、和縮減檔案。讓我們考慮，作業系統對於六種基本檔案操作必須做的事：

- 建立檔案　建立檔案需要兩個步驟。首先，為了這個檔案，其空間必須在檔案系統中被找到。其次，必須在目錄中為新檔案做一個項目。檔案項目記錄了檔案的名稱和其在檔案系統中的位置。

- 寫入檔案　為了寫入一檔案，要做一次系統呼叫，指定檔案名稱和欲寫入檔案之資訊。檔案名稱若已定，系統會搜尋目錄以找到檔案的位置。目錄項目將需要儲存一個指標以指向檔案目前的尾端。利用這個指標，下一個區段可被計算出來並且資訊可以被寫進去。此寫入指標必須被更新。用這個方式連續的寫入可將一序列的區段寫入一個檔案。
- 讀取檔案　為了從一檔案讀出，一個系統呼叫指定了檔案的名稱和檔案下個區段被放置的處所(在記憶體中)。同樣地，為了相關的目錄項目，目錄再次被搜尋了。並且，目錄將需要一個指標以指向下次被讀的區段，一但那個區段被讀出，此指標即被更新。一般說來，檔案不是被讀便是被寫，所以雖然它可能會有兩個指標，一個讀取指標和一個寫入指標，但大多數系統只有一個目前檔案位置指標。讀和寫二個操作使用相同的指標，將節省目錄項目中的空間，並且簡化系統的複雜性。
- 重置檔案　搜尋目錄找到相關的進入點，然後把目前檔案位置設定成某一固定值。重置一個檔案並不需要有任何真正輸出入。這個檔案操作也稱為檔案搜尋。
- 刪除檔案　為了刪除一個檔案，我們搜尋目錄以找此檔案。在找到相關的目錄後，我們釋放所有檔案的空間並且將此目錄項目作廢。
- 縮減檔案　有時候使用者希望檔案的特性保持現狀，但希望清除掉檔案的內容。使用者不必先刪除掉該檔案然後重新產生，他可以使用此功能使檔案特性保持不變(除了檔案長度)，但檔案恢復為長度零。

這六項檔案操作是組成所有必要檔案操作的最基本部分。其他常見操作包括了加入(appending)新資料到現存檔案的尾端，(renaming)為現存檔案更改名稱。這些幾本操作也可以結合以產生其他檔案操作。譬如，產生一個檔案的拷貝，或拷貝檔案到其他 I/O 裝置就可以先產生一個新的檔案，然後從舊檔案讀出，並寫入新檔案。

上述所有的操作均涉及到對檔案有關的目錄的搜尋，此目錄項目含有所有檔案操作所需要的重要資訊。為避免這種經常性的搜尋，許多系統當它第一次使用時，將開啟一個檔案。作業系統保存一個小表(開啟檔案表)以包容所有開啟檔案的資訊。當需要一個檔案操作時，只要搜尋這個小表，而不是整個目錄。當檔案不再使用時，它會被關閉並且從開啟檔案表中移除。

某些系統在第一次接觸檔案時便主動地開啟一個檔案，當開啟此檔案的工作或程式終止時便自動地關閉檔案。無論如何，大多數的系統在檔案被使用之前需要程式設計師以一系統呼叫(open)來開啟檔案。開啟操作接受一個檔案名稱並搜尋目錄，拷貝目錄項目到開啟檔案的表中。然後此 open 系統呼叫將傳回一指到開啟檔案表項目中的指標。此指標(不是真正的檔案名稱)被用在所有的輸出入操作，以避免任何多餘的搜尋，並簡化系統呼叫界面。

4.2 檔案存取

當檔案被使用時，資訊必須被存取並且讀入電腦的記憶體中。檔案中的資訊能用好幾種方式來存取。某些系統對檔案只有一種存取的方法，其他系統(例如IBM)則提供了許多不同存取的方法。

4.2.1 循序存取(Sequential Access)

這是最通用的檔案存取模式，檔案中的資訊是依著紀錄次序一筆接著一筆處理的。大部分檔案作業都是做讀出和寫入的工作。讀出的內容是目前所在位置接下來的部分，並且自動地將檔案指標向前轉。同樣地，寫入資料也是附加在現在檔案內容的尾端，並且前進至新寫入資料的末端(新的檔案結尾)。這類檔案也可以倒轉，在某些系統中，一個程式可以向前或向後跳過 n 個紀錄，n 是一個正整數(也許 n=1 而已)。這種存取方法就是所謂的檔案循序存取。循序存取是以檔案存在磁帶模式中為基本條件，但也可以在隨機存取裝置中運作。

4.2.2 直接存取(Direct Access)

另一種存取方法是直接存取。檔案是由固定長度的邏輯紀錄所組成，這可以讓程式不必以一定的順序，快速地讀寫紀錄。直接存取是使用在檔案存放在磁碟上的情況。為了要直接存取，檔案被視為一串編有號碼的區段或紀錄。直接存取的檔案允許讀出或寫入任意數量的區段長度。因此我們可以先讀出 14 個區段，再讀入 53 個區段，接著又寫入 7 個區段。對直接存取的檔案而言並沒有寫入和讀出的次序限制。

直接存取檔案對於立即要存取大量的資訊而言非常有用。它經常用來存取大型資料庫。當某個詢問要求立即送來某個標題的時候，我們計算出在那個區段裡存著那個答案，然後直接從那區段中讀出所要供應的資訊。

並不是所有作業系統都同時提供循序和直接存取兩種方法。某些只允許循序存取，另有一些則只允許直接存取。有些系統會要求在新建檔案的時候定義出該檔案是循序存取還是直接存取：這類檔案只能用被宣告的方式來存取。然而要注意的是，在直接存取檔案上模擬循序檔案是非常容易的事。如果我們用 p 來代表現在的位置，那我們就可以用(p = p+1)的方法來模擬循序檔案作業方式。但是反過來說，在循序檔案上模擬直接存取檔案卻是非常沒有效率和笨拙的。

4.2.3 其他存取方法

其他的存取方法也可以建立在一個直接存取方法的基礎上。這些方法一半都包含了對檔案所設的索引結構。索引(index)，就像書本前面的目錄一樣，包含了指向不同區段的指標(pointer)。要找到檔案中某個單元，我們先在目錄中尋找，然後使用指標去直接存取檔案和找到所要的單元。

針對大型資料而言，索引檔案本身可能會變得過大而無法存在記憶體中。於是有一種解決方法就是設定索引檔案的索引。這個主索引檔案(primary index file)將包含有指向副索引檔案(secondary index file)的指標，副索引檔案又含有指向實際資料項目的指標。

4.3 目錄結構

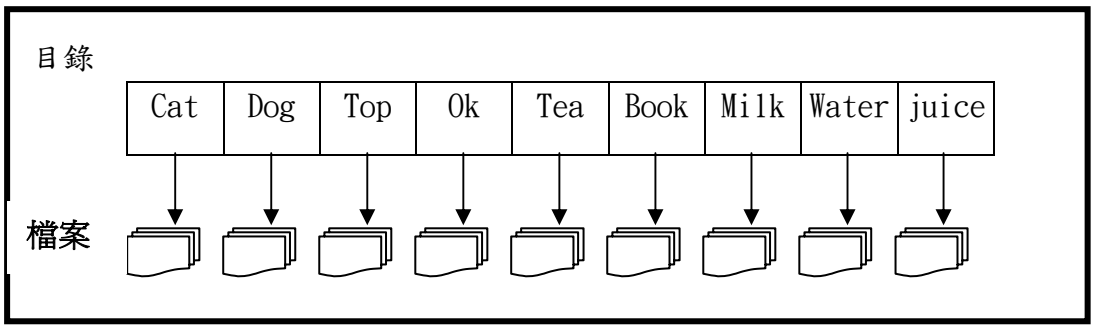
電腦的檔案系統可能非常龐大。有些系統儲存數千個檔案在幾百萬位元組的磁碟上。為了處理這些資料，我們必須將它們加以組織。這項組織化的工作通常必須以兩個部分完成。首先，檔案系統被分成數個“分割”(partition)，在 IBM 被稱為微型磁碟(minidisk)，在 PC 和 Macintosh 上則稱為卷(volume)。通常，一個系統的每一個磁碟至少包含了一個分割，這些分割檔案和目錄儲存在其中的低階結構。有時候，在同一磁碟機上可藉由不同的“分割”來提供分隔的區域，每一區域可視為一塊單獨儲存裝置，而有些系統則允許“分割”比單一磁碟機大，以便將數台磁碟機組成單一的邏輯結構。根據這種方式，使用者只須注意邏輯目錄和檔案結構，他可以完全不必理會在實際上如何分配空間給檔案的問題。因為這個原因，所以“分割”可以視為虛擬磁碟機。

接下來，每一分割皆包含了存放在其中檔案的資訊。這項資訊存放在裝置目錄(device directory)或卷表(volume table)之中。裝置目錄通常就稱為目錄，記錄在此分割裡所有檔案的資訊，譬如：檔名、位置、大小和型態。

目錄可以視為一個把檔名轉換成目錄進入點的符號表。若我們以這種觀點來看，則很明顯地，目錄本身可以用許多不同方式組成。我們希望能夠加入新的項目，刪除項目，搜尋某一指名的項目，或列出目錄中的所有項目。

4.3.1 單層目錄(Single-Level Directory)

最簡單的目錄結構就是單層目錄。一般裝置的目錄就是一種單層目錄。有的檔案都裝在同一目錄中，非常容易瞭解與使用(見下圖)。



然而，單層目錄在檔案數目增加以後，或是不只一個使用者的時候會有很大的困難出現。因為所有檔案都在一個目錄中，檔案只能使用獨一的名稱。如果我

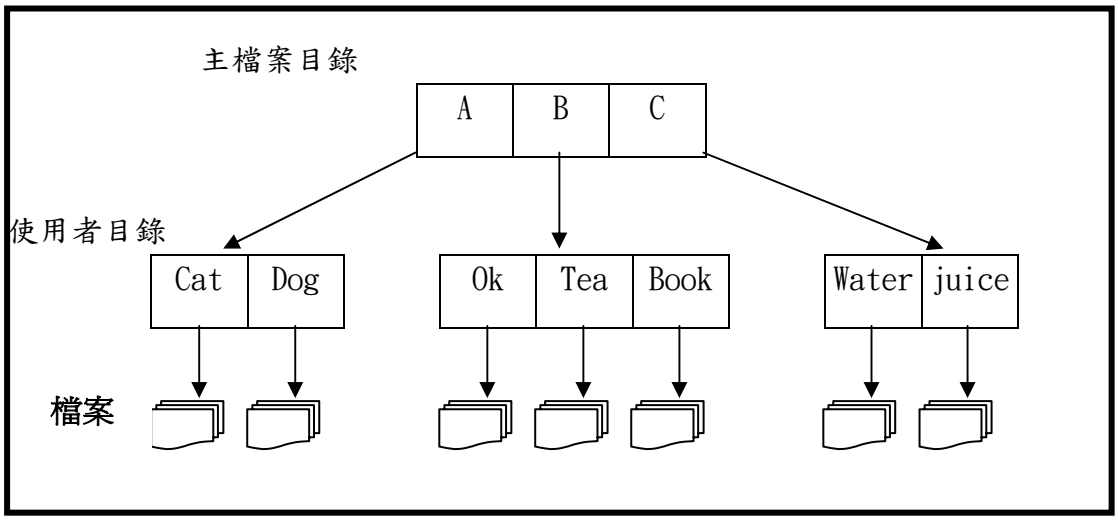
們有兩個使用者並且同時呼叫他們的測試資料檔案 test，那麼就違反了獨一名稱的規定。雖然檔案名稱一般都選用來反應檔案內容的名稱，但是它們大多是被限制長度的。MS-DOS 只允許使用 11 字元的檔案名稱；UNIX 則允許 255 字元。

即使只有一個使用者，當檔案數目增加的時候，那也會變得很難記住所有檔案的名稱來建立只使用獨一名稱的檔案。

4.3.2 雙層目錄(Two-Level Directory)

單層目錄的主要缺點就是在不同使用者之間檔案名稱的混亂。標準的解決方法就是建立每個使用者一個單獨的目錄。

在雙層目錄結構中，每個使用者擁有自己的使用者檔案目錄(見下圖)。



每個使用者都有一個相似的結構(線性的、二元的，或對應的)但是只列出單一使用者的檔案。當一個使用者的任務開始了或是一個使用者簽到(log in)之後，系統的主檔案目錄(MFD)就被搜尋一遍。主檔案目錄是用使用者姓名或是帳號來索引，並且其中每單元都指向一個使用者的使用者目錄。

當一個使用者想要某個檔案的時候，只有他自己的檔案目錄會被搜尋。這樣不同使用者就可以擁有相同名稱的檔案了，只要在每個使用者的檔案目錄中的所有檔案名稱是獨一的就可以了。

要為某個使用者建立一個檔案，作業系統只搜尋該使用者的目錄，以確定是否已有相同名稱的檔案存在了。要除掉一個檔案，作業系統將搜尋限制在該使用者的目錄中；因此不至於意外地將其他使用者具有相同名稱的檔案除掉。

使用者目錄在必要的時候要能除去和建立。有一種以適當的使用者名稱和帳號資料所定義的一種特殊程式可以應用於此。此程式建立一個新的使用者檔案目錄並且為它加上一個項目到主檔案目錄(master file directory)去。

在雙層目錄結構中仍舊存在著一些問題。這個結構很有效地將使用者分隔開

了。這是當使用者之間屬於完全獨立情況下的好處，但是一旦使用者想要在某個工作上和別人合作並且分享彼此檔案的時候，這就成了缺點。某些系統並不允許本地使用者的檔案讓其他的使用者存取。

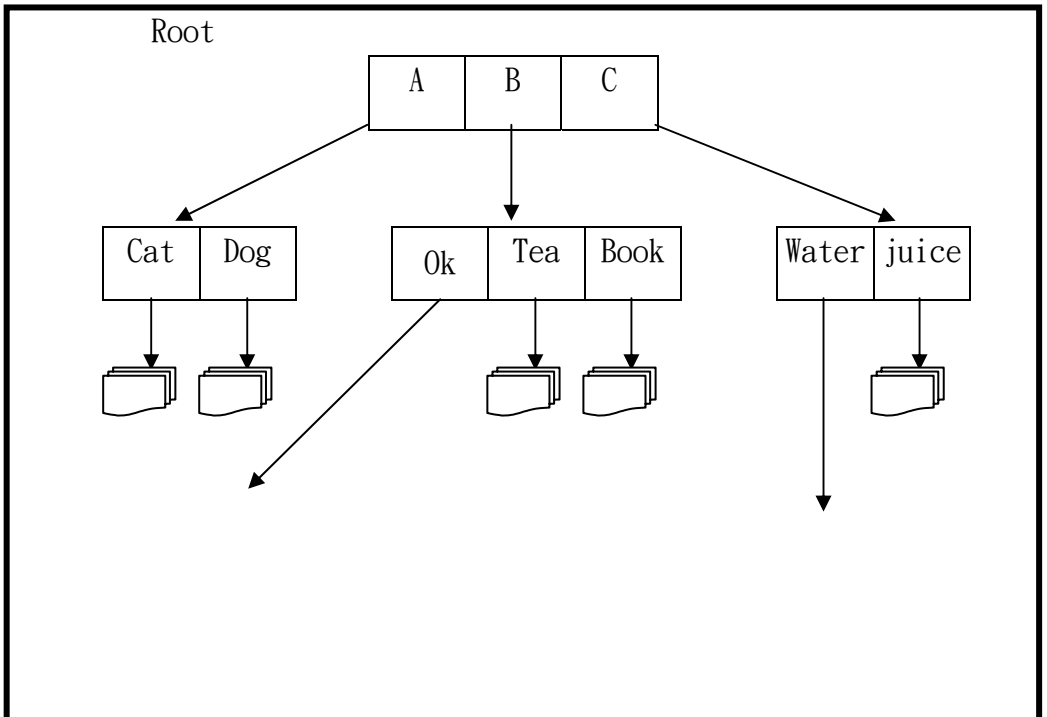
如果存取被允許了，一個使用者必須有能力在其他的使用者目錄中定出一個檔案的名稱。要確定某檔案在一個雙層目錄中的名稱是不是唯一的，我們必須同時給予使用者名稱和檔案名稱。一個雙層目錄中的名稱可以想成高度為 2 的樹。樹的根就是主檔案目錄。它的直接子孫(descendant)是使用者檔案目錄。而使用者檔案目錄子孫則是檔案本身。檔案是這棵樹的葉子。

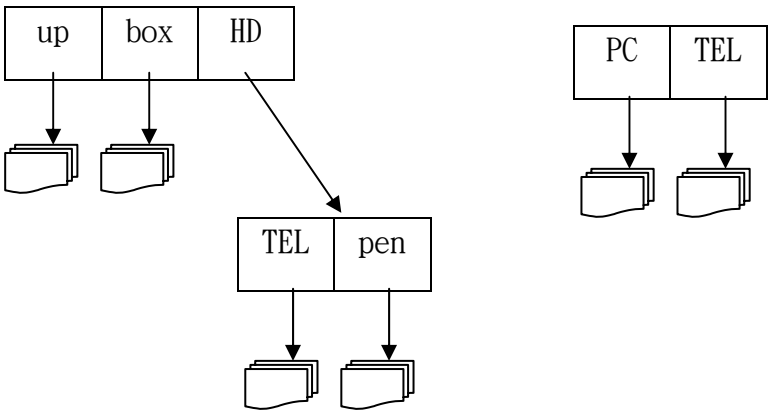
4.3.3 樹狀目錄(Tree-Structured Directories)

一旦我們把雙層目錄視為一棵二階的樹，由目錄結構發展成一棵任何形式的樹的理論就自然產生了多階的目錄樹(下圖)。這將允許使用者去建立它們自己的副目錄，並且可以適當地組合他們的檔案。例如 MS-DOS 檔案系統就是一個樹狀結構。這樹有一個根部目錄，在此系統中的每一個檔案都有一個單獨的路徑名稱。路徑就是由根部，經過所有副目錄，到某一個檔案的路徑。

一個目錄(或副目錄)存有一組檔案或副目錄。所有目錄的內部格式都一樣。在每一目錄項目中都用一位元來表示這項是個檔案(0)還是一個副目錄(1)。一些特殊的系統呼叫被用來建立和除去目錄。

在一般使用中，每個使用者都有一個現用目錄(current directory)。現用目錄應該包含使用者目錄最感興趣的大部分檔案。當要參考某個檔案時，先搜尋現用目錄。如果所需的檔案並不在現用目錄裡，那麼使用者就必須指定一個路徑名稱或是換掉那個現用目錄。有個系統呼叫可以將目錄名稱當做參數使用並且用它來重新定義一個現用目錄。因此使用者可以在需要的時候改他的現用目錄。從一個更換目錄的系統呼叫到下一個，所有開啟檔案的系統呼叫都搜尋現用目錄以找尋到
特定的檔案。





當一個使用者的工作開始或者最初簽到時,該使用者就會被指定一個現用目錄。作業系統會搜尋會計檔(accounting file)替使用者找到一個進入點。在會計檔中是一個指向使用者最初目錄的指標。為了這個使用者,指標被拷貝到一個區域變數,這個區域變數設定了使用者最出現用目錄。

路徑名稱可分為兩類:絕對路徑名稱或是相對路徑名稱。一個絕對路徑名稱是由根部開始並且循著一條路徑到達某特定檔案,而在一路上給定所要經過的目錄名稱。相對路徑名稱則是定義一條由目前現用目錄去的路徑。舉例來說,在上圖的樹狀結構檔案系統中,如果現用目錄是 Root/B/OK,那麼關於同一個檔案的相對路徑名稱是 HD/PC 而絕對路徑名稱是 Root/B/OK/HD/PC。

允許使用者定義他們自己的副目錄就是允許他們在檔案上面加上一層結構。這層結構可以讓檔案目錄與不同的主題發生關聯(例如,建立一個副目錄以保存本書的內容)或與不同型式的資訊發生關聯。

在樹狀結構的目錄中有一項有趣的策略性決定,那就是如何去控制目錄的消除工作。如果某個目錄是空白的,那麼這目錄當然可以除掉。但是,如果要除掉的目錄並不是空白的,而是存放了許多檔案,或是子目錄的話,要怎麼辦呢?有兩種方法可以採用。有些系統無法將非空白的目錄除掉。因此如果要除掉某個目錄,我們就必須先將其內部的所有檔案除掉。如果還有子目錄,就必須反覆使用這個程式,如此將它們也除掉。這個方法也可能造成大量的工作。

另一種方法是,當要求除去一個目錄時,那麼該目錄的所有檔案和子目錄都會一起被刪除;這種方法被應用在 UNIX 和 rm 命令上。這兩種方式都很容易做到;至於選用那一種方式則是策略上的應用。後者比較方便,但比較危險,因為只要一個指令,整個就被刪除掉了。如果下達錯誤的指令,則一大堆檔案和目錄都需要從備份磁帶中重新補救回來。

有一個樹狀結構的目錄系統,使用者除了可以存取自己的檔案外,也可以存取其他使用者的檔案。舉例來說,使用者 B 能夠定義他們的路徑名稱來存取使用者 A 的檔案。使用者 B 能夠定義一個絕對的或一個相對的路徑名稱。相對的,使用者 B 的現用目錄可以被改變到使用者 A 的目錄而且使用者 B 可以直接以它們的

檔案名稱來存取檔案。某些系統也允許一個使用者定義一個自己的搜尋路徑。在這情況下，使用者 B 可以定義一個搜尋路徑依序為(1)他的區域性目錄，(2)系統檔案目錄，以及(3)使用者 A 的目錄。只要使用者 A 的檔案名稱不和區域性檔案或系統檔案的名稱衝突，它就可以只用它的名稱來參考。

在樹狀結構的目錄中，要到達一個檔案的路徑可能比在一個兩層目錄中的路徑長。為了讓使用者在存取程式時不必記住這些冗長的路徑，Macintosh 作業系統自動地為可執行檔做搜尋。它維護一個檔案叫做“桌上檔案”(Desktop File)，其中包含了它所看到的所有可執行檔之名稱和位置。當一個新的硬碟或軟碟加入系統中，或者網路被存取時，作業系統會經歷目錄結構，搜尋在裝置上的可執行程式，並且紀錄下相關的資訊。

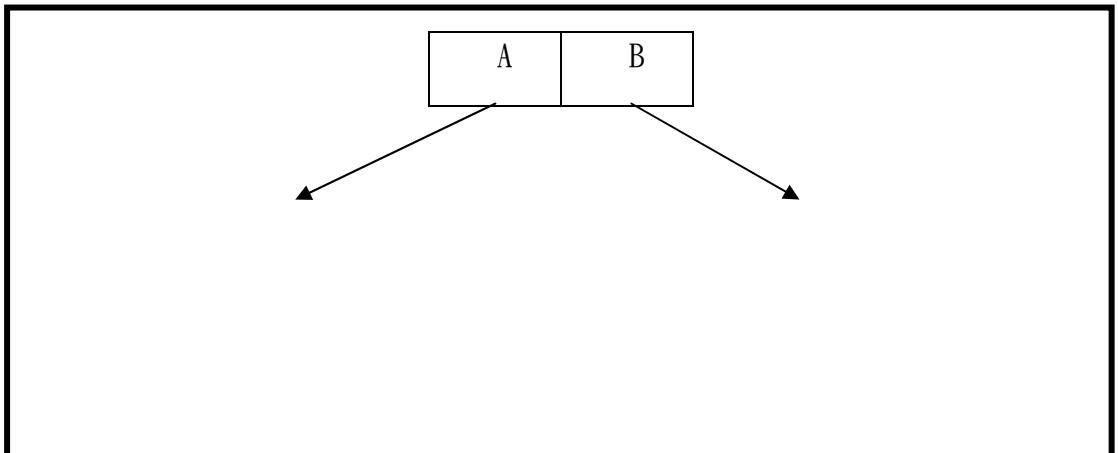
4.3.4 非循環圖型目錄(Acyclic-Graph Directories)

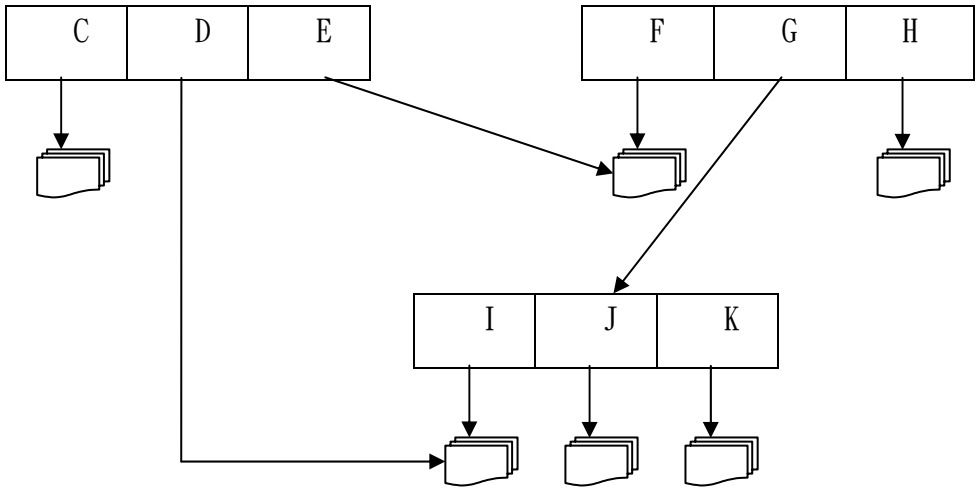
樹狀結構限制了檔案或目錄的共用。一個非循環圖型則允許目錄中的副目錄或檔案被共用(下圖)。相同的檔案或子目錄可以在兩個不同的目錄中。非循環式圖型是樹狀目錄法的自然發展結果。

在許多人共同以團隊方式工作的時候，所有的共用檔案就要放在同一個目錄中。每個團隊成員的檔案目錄將會把共用檔案的目錄視為它的子目錄。即使只有一個使用者，他的檔案結構也可能會要求將一些檔案放在許多不同的子目錄中。舉例來說，為一個特殊計劃而寫的程式應該在所有程式的目錄和那個計劃的目錄之中。

共用檔案和子目錄的操作方法有好幾種。最普通的方法是設置一個稱為鏈(link)的新目錄單元。鏈實際上就只是一個指向其他檔案或子目錄的指標罷了。舉例來說，一個鏈可以當做一個絕對路徑名稱(符號鏈，symbolic link)來用。當要參考某個檔案的時候，我們就搜尋目錄，目錄中的單元被標示成一個鏈並且被告知該實際檔案的名稱。我們用路徑名稱去找出實際檔案位置。鏈非常容易以它們在目錄單元中的格式去辨識出來，並且是一種非常有效且具有名稱的間接指標。

另一種執行共用檔案的方法是複製所有資料存放在兩個共用目錄中。因此兩個目錄的單元完全一樣。鏈當然和原來的目錄單元不同，因此這兩個目錄並不一樣。但是，複製目錄單元卻使得原來的和複製出來的無法區別。複製目錄單元所遭遇到最大困難就是當檔案被修改過後，如何去保持它們的一致性。





非循環圖型目錄結構比起簡單的樹狀結構要更具彈性些，但是也比較複雜些。許多問題都必須小心處理，檔案可能擁有許多絕對路徑名稱。因此，不同的檔案名稱可能都是指同一檔案。和這程式語言中的別名問題都很相似。如果我們想要詳細考察整個檔案系統，這個問題就更嚴重了，因為我們並不想使用共用的結構好幾次。

另外一個問題包括了刪除的問題。被分配給共用檔案的記憶空間什麼時候可以收回和重新利用呢？有一種可能就是任何人刪除掉一個檔案我們就立刻把它移開。但是這種舉動可能會留下一個指向現已不存在的檔案指標懸在那裡。更糟的是，如果剩下的檔案指標保存了實際磁碟位址，而記憶空間接著又被其他檔案使用了，而懸在那的指標可能會指到其他檔案的中間。

在使用符號鏈來執行共用檔案的系統中，這個情況比較容易控制。將一個鏈除掉並不會影響原先的檔案；因為只移動了鏈而已。如果檔案單元本身除去了，那麼該檔案的空間就收回，留下鏈懸置在那裡。我們也可以搜尋這些鏈並移去它們，但是除非是把每一個檔案相關的鏈放在一個串列中，否則這樣搜尋將所費不貲。此外，我們也可以留著這些鏈到下次來用。到那個時候，我們可以斷定由鏈定名的檔案已不存在並且無法解決鏈的名稱；這時的存取作業就像處理其他任何不合法的檔案名稱一樣。

另一種方法是直到某檔案的所有參考資料除掉之後，才把檔案除掉。要執行這個方法，我們必須有一些辦法來斷定該檔案的最後一項參考資料使否已被除掉。我們可以將一個檔案的所有參考資料都列出來(目錄單元或是鏈)。當一個鏈或目錄單元的拷貝建立之後，一個新的單元就加到檔案參考資料列的尾端。當一個鏈或是目錄單元被除掉了，我們就把它從串列中移去，當一個檔案的參考資料列成為空白的之後，該檔案就被除掉了。

這種方法的麻煩是一些變數和檔案參考資料列的潛在巨大體積。但是，我們可以發現我們實際上並不需要保存整個資料列，而只需要去計算參考資料的數目即可。一個新的鏈或目錄單元會增加該參考量的值；除去一個鏈或一個單元則將

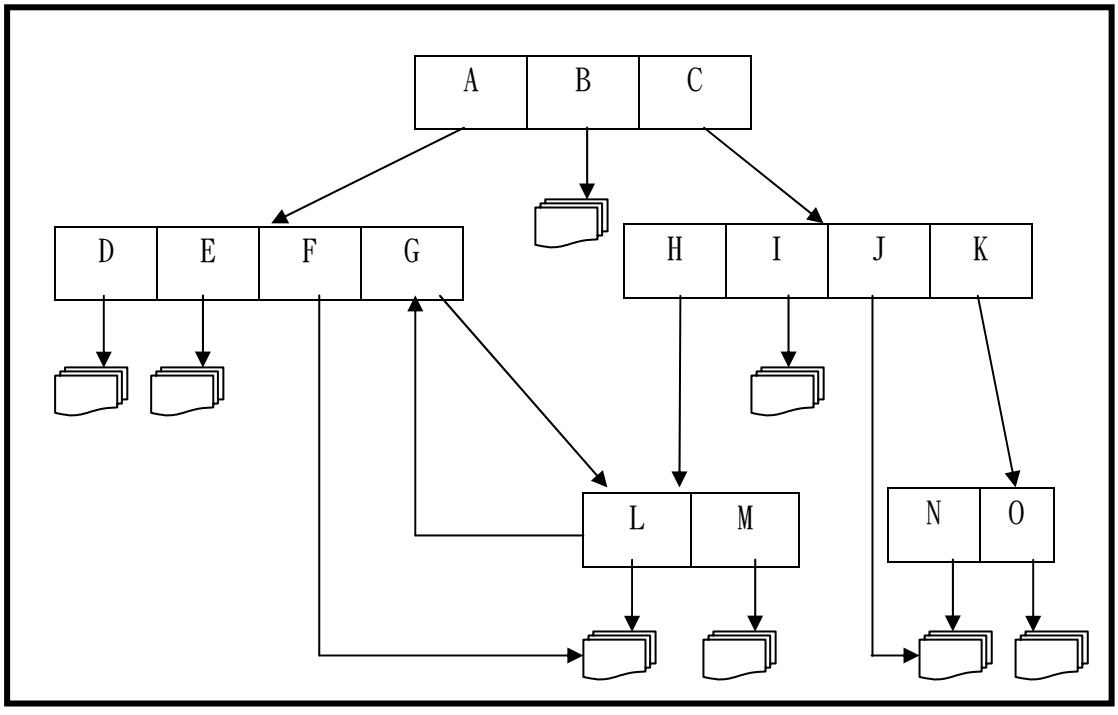
減少該值。當數值為 0 的時候，這檔案就可以除掉了；因為已沒有參考資料存在了。

為了避免這些問題，有些系統並不允許共用目錄或鏈。譬如，在 MS-DOS 中，目錄是樹狀結構，而不是非循環圖型目錄，這可以避免刪除掉檔案時所遇到的麻煩。

4.3.5 一般圖形的目錄(General Graph Directory)

使用非循環圖型結構時最嚴重的問題就是如何保證沒有循環存在。如果我們以雙層目錄開始，並且允許使用者建立子目錄，一個樹狀結構的目錄就形成了(下圖)。

非循環圖型主要的好處是檢視它的演算法在相對上十分簡便，並且在判定何時檔案參考資料沒有的方法也十分簡便。我們之所以想要避免檢視非循環圖型結構中共用部分兩次，主要是為了執行上的理由。如果我們剛搜尋過的一個主要共用的子目錄中的某檔案，但是沒找到，我們就要避免再次搜尋那個子目錄，因為那樣很浪費時間。



非循環圖型主要的好處是檢視它的演算法在相對上十分簡便，並且在判定何時檔案參考資料沒有的方法也十分簡便。我們之所以想要避免檢視非循環圖型結構中共用部分兩次，主要是為了執行上的理由。如果我們剛搜尋過的一個主要共用的子目錄中的某檔案，但是沒找到，我們就要避免再次搜尋那個子目錄，因為那樣很浪費時間。

如果在目錄中允許循環存在，我們同樣想要避免重複搜尋任一單元兩次，為

的也是執行上和正確性上的理由。一個差勁設計的演算法可能會造成一個無線迴路，不斷搜尋一個循環而永不終止。

在判定何時可以除掉某個檔案的時候也會遭遇類似的問題。若使用的是非循環圖型結構，參考資料計算值為零表示某檔案或目錄已無任何參考資料了，並且可以消除掉。但是，如果有循環存在，即使早就沒有檔案或目錄存在，參考資料計算值也可能不是零。這個異常結果由目錄結構中的自我參考(一個循環)的可能性所造成。在這個情況下，一般都需要廢置空間收集法(garbage collection)來判斷最後一個參考資料是何時被除掉的及磁碟空間可否重新分配。廢置空間收集法中包括檢視檔案系統，標示一切可以存取的東西。第二次再來將其餘沒有被標示的都收集到可用空間的列表中。但是在使用磁碟的系統中，廢置空間收集法非常費時間所以很少被使用。

廢置空間收集法之所以需要存在也是因為有可能在圖形中出現循環狀況。因此非循環圖型結構是非常容易使用的。主要困難就是避免循環，尤其是在將新鏈加到原結構上的時候。我們如何知道何時一個新鏈會造成一個循環呢？當然有演算法可以來偵測圖形中的循環，但是價格非常昂貴，尤其是當圖形是存在磁碟儲存體中的時候。

4.4 配置方式(Allocation Method)

磁碟直接存取的特性允許我們在執行檔案的時候更具有彈性。大部份的情況之下，許多檔案可同時存放在同一個磁碟上。在這方面的主要問題是如何分配記憶空間給這些檔案，使得磁碟空間能有效地被利用及檔案可以快速地存取。有三種主要的分配磁碟空間方法：連續的(contiguous)、鏈接的(linked)、索引的(indexed)。每種方法都是利弊互見。因而，有些系統(譬如 Data General 公司的 Nova 系列電腦上的 RDOS)則同時提供三種方法。但是大多數一個系統只使用其中一種方法來處理所有檔案。

4.4.1 連續式分配(Contiguous Allocation)

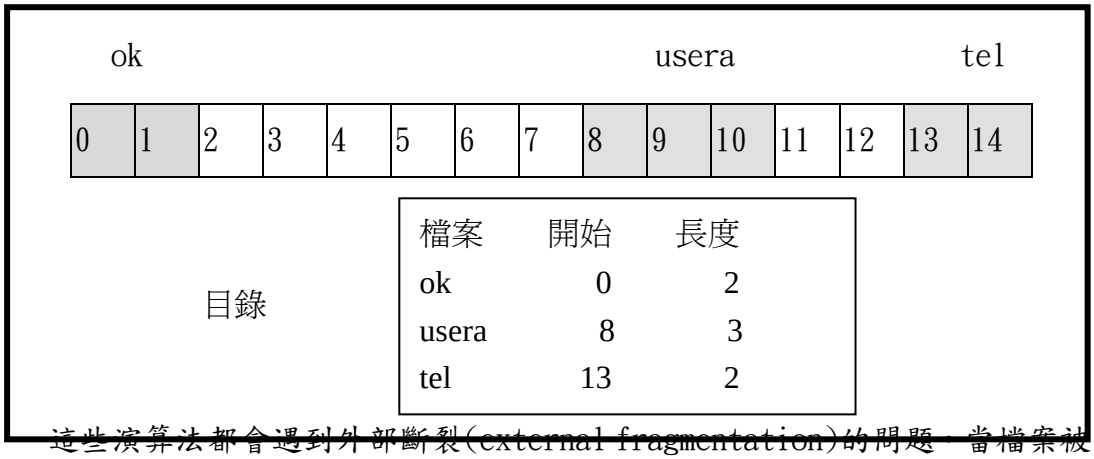
連續分配法需要令每個檔案佔用一組磁碟上的連續位址。磁碟位址定義了一個線性的磁碟上排列次序。此時的順序，在存取 b 區段之後存取 b+1 區段時一般不需要移動磁頭。若是要移動磁頭，也只是移動一條磁軌而已。因此，存取連續分配檔案的尋找次數將可達到最小，且搜尋時間亦為最短。

檔案的連續分配是藉第一區段的磁碟位址和區段長度來定義。如果檔案有個 n 個區段長，並且從第 b 處開始，那麼它就佔用了第 b、b+1、b+2、・・・、b+n-1 區段。目錄中每個檔案的項目指示出起始區段的位址，以及分配該檔案的區段長度(見下圖)。

要存取一個用連續分配的檔案非常簡單。對循序存取而言，檔案系統記得上

一個區段的磁碟位址，當要讀入資料的時候，就把下一個區段讀入。對直接存取來說，若要存取由第 b 區段開始的某檔案的第 i 區段，我們可以立即存取第 b+i 區段。因此循序存取和直接存取都可以用在連續分配上。

連續分配的困難是在為新檔案尋找可用空間的時候。這個問題可以視為一般動態儲存分配(dynamic storage allocation)問題的一個特例。最先配合(first-fit)最佳配合(best-fit)是最常用來選出可用空間的策略。模擬結果顯示最先配合和最佳配合在時間和儲存體二者的利用上比最差配合來得好。顯然地最先配合和最佳配合在儲存體的利用上都不是最好的，但最先配合通常是比較快。



這些演算法都會遇到外部斷裂(external fragmentation)的問題，當檔案被配給可用空間或是除掉之後，可用磁碟空間就被打散成很小的單位。當磁碟中有足夠用於某項要求的總空間量，但它們卻非連續的時候，此時外部斷裂現象就出現了；儲存器就被分成許許多多的小洞(hole)。外部斷裂現象可能是個很小的問題，也可能是一個很大的問題，這完全要看磁碟儲存量有多大及檔案的平均大小來決定。

有一些舊型微電腦系統在軟式磁碟系統中使用連續分配法。為了防止外部斷裂現象造成大量磁碟空間被損失掉，使用者必須執行一個重新集中常式來把整個檔案系統拷貝至另一份軟式磁碟或磁帶上。然後把原磁碟全部空出來形成一個大的連續可用區域；再把拷貝出去的那些檔案拷貝回原磁碟，並且連續地把空間分配給各檔案。這個方法有效地將所有夾雜在各檔案之間的可用空間聚集在一個區域裡，解決的斷裂的問題。聚集法所費的代價就是時間。對使用連續配置的大型硬碟而言，時間成本是特別地明顯，當它要聚集所有的空間時，可能花好幾小時，也可能好幾天。在這段停止時間(down time)，通常不能允許正常的系統操作，就量產的機器成本而言，這些聚集無論如何也要避免。

連續分配還有一些其他問題。最主要問題是在判斷一個檔案需要多少空間的問題。當建立一個新檔案的時候，必須找到並且分配一切必須的空間。使用者(程式或人員)如何預知所要建立的檔案有多大呢？有些情形下要做判斷是件很容易的事(例如拷貝一個現有的檔案)，但在一般情況下輸出檔案的大小都十分難以預

測的。

如果我們分配給某檔案的空間太小，我們會發現它無法擴充。尤其是採取最佳配合分配方法的時候，檔案兩側的空間可能都在使用中，我們卻無法將檔案加大一點點。有兩種可能存在。第一種是把使用者程式以適當地錯誤訊息將它故意終止掉。於是使用者可以重新分配更多空間來再執行該程式一次這樣重複執行可能所費不貲。為了防止這種狀況使用者一般都要超估所需空間的數量造成可想而知的空間浪費。

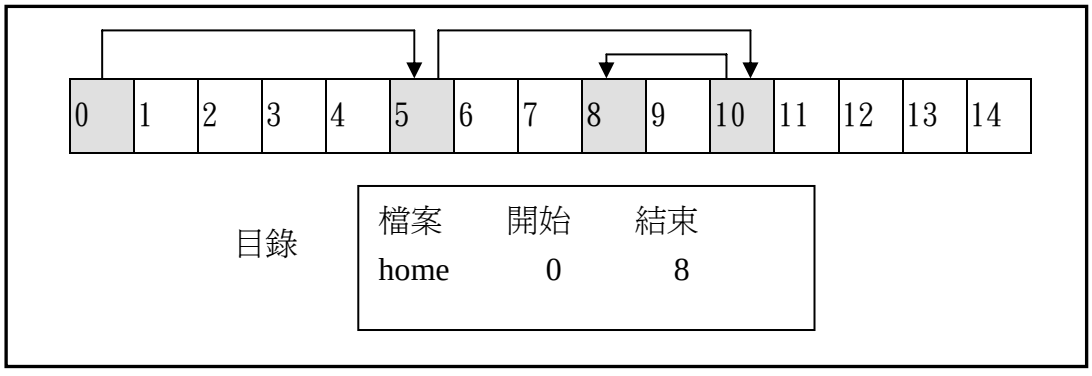
另外一種可能是去找出一個較大的區域，將檔案內容拷貝至這個新的空間並且把原來的空間釋開。只要有空間存在這動作就可以不斷重複，雖然這也是很耗時間的。但是請注意在這情況之下，根本不必通知使用者現在正發生了什麼事；系統一直忽視的問題就是其速度上變慢了。

即使已事先知道一個檔案所需的空間總量，預先分配其空間可能是沒有效率的。對一個長期(經年累月)成長緩慢的檔案而言，必須為其最後大小分配到足夠的空間，縱使大部分的空間長久都沒有用到。所以最後檔案有大量的內部斷裂。

為了避免這些缺點，有些作業系統使用修正的連續分配法，這種方法一開始分配一段連續的空間，然後當此空間不足夠大時，另一段連續空間(叫做 extent)就加到原先的配置。此檔案的位置則紀錄成某一位址，和一個區段計數，再加上一個鏈結到下一段延伸開頭的指標。

4.4.2 鏈結分配(Linked Allocation)

用鏈接的分配法可以解決斷裂的問題。使用鏈接分配，每個檔案都是磁碟上區段的鏈接串列(linked list)；磁碟上的區段可能散佈到磁碟的任何地方。



目錄中存放著指向檔案第一個(和最後一個)區段的指標。舉例來說，由第 0 區段開始的一個擁有 4 個區段的檔案，可能由第 0 區段接到第 5 區段，然後接到第 10 區段，，最後到第 8 區段(見前圖)。

每個區段都含有一個指向下一個區段的指標。這個指標對使用者並沒有用處。因此如果每個區段是 512 個位元組，而每個磁碟位址(指標)需要 4 個字組，

那麼使用者可用的有 508 個位元組。

建一個檔案很簡單；只要在裝置目錄中建一個新的單元就可以了。若使用鏈接的分配，目錄中每個單元都有一個指標指向檔案的第一個磁碟區段。指標原先是 nil(表尾指標的值)用來表示一個空檔案，大小欄位亦設為零。若要寫入一個檔案，就將可用空間串列中的第一個可用區段移去，並且將資料寫在上面。這個新的區段於是鏈接到原檔案的尾端。若要讀某個檔案，我們就讀跟在指標後面的區段，循著指標由一個區段到另一個區段地讀下去。

在鏈接的分配方法中沒有外部斷裂的現象。任何可用空間串列上的可用區段都可以用來滿足一些要求，因為所有區段都是鏈接在一起的。同時要注意的是我們並不需要建檔的時候先宣告檔案的大小。只要有可用的區段存在檔案就可以繼續擴大。因此，根本就不必將磁碟空間聚集在一起。

然而鏈接分配也有缺點。最大的問題就是它只能在循序存取檔案中有效地使用。若要找出檔案第 i 區段，我們必須由該檔案的起點開始並且循著指標找下去，直到我們找到第 i 區段為止。每次對指標的存取都需要做磁碟讀入。因此，對使用鏈接分配的檔案我們無法提供直接存取的能力。

鏈接分配的另一項缺點就是指標本身所要佔的空間。如果一個指標就佔用一個 512 位元組區段中的 4 個位元組，那麼磁碟空間的百分之 0.78 被利用在指標上了，而不是用於存放資料。因此每個檔案就需要更多的空間。

這個問題常見的解決方法是將區段組在一起成為 cluster，並且用 cluster 而不是用區段做配置。例如，檔案系統可能定義一個 cluster 是 4 個區段，而且磁碟上的操作以 cluster 為單位。因此指標將使用掉磁碟空間最小的比例。這個方法允許邏輯對實際區段的對映保持簡單，但卻增進了磁碟的產量(較少的磁頭搜尋)，並且降低了區段配置和可用空間串列管理的代價，因為如果一個區段只有部分填滿會比全部填滿浪費空間。cluster 可以用來對許多其他地方的磁碟存取最佳化。所以它們用在大多數的作業系統。

更嚴重的是可信度(realiability)的問題。因為檔案是由散佈在磁碟中的指標鏈接起來的，一但有一個指標搞錯了或被破壞了，那會發生什麼事呢？作業系統軟體中的錯誤或是磁碟硬體上的損壞都會造成指標錯誤指向。這樣會造成鏈接到可用空間表內的單元或是鏈接到其他檔案上。特殊的解決方法就是用雙重鏈接表(doubly linked list)，或者將檔案名稱及相對的區段號碼存在每個區段中；不過這些方法將令每個檔案佔用更多空間。

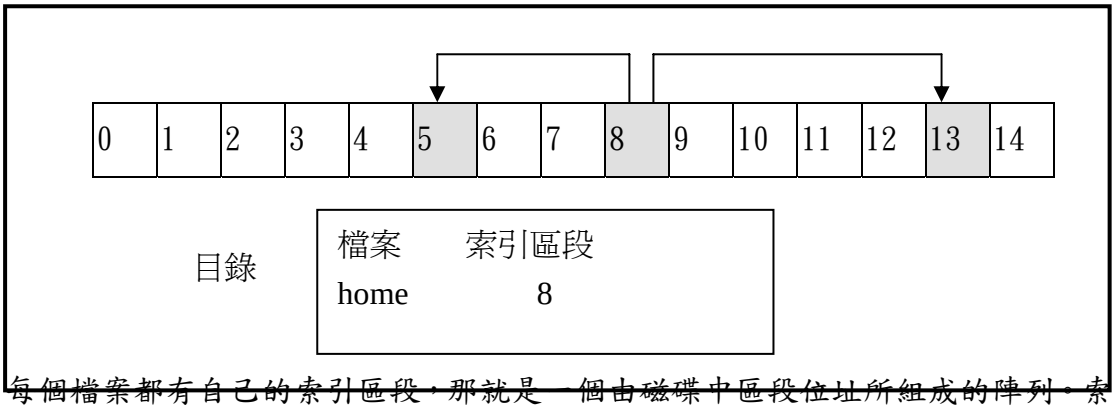
鏈接式配置方法的一項重要變化方式是採用“檔案配置表格”(file-allocation table，即 FAT)。這種簡單但有效率的磁碟空間配置法用在 MS-DOS 和 OS/2 的作業系統。每一分割的開頭磁碟部分都用來存放此表。該表格為每個磁碟區段保有一份紀錄，並且以區段號碼當作索引。FAT 用起來很像鏈結串列。目錄中包含檔案第一個區段的區段編號。以區段編號為索引之表格紀錄包含了檔案中下一個區段的區段編號。如此不斷連接下去，直到最後一個區段，其表格紀錄是一特殊之檔案結束(end-of-file)數值。未使用之區段以零的表格數

值來表示。分配一個新的區段給檔案只不過是找尋第一個零值的表格紀錄，然後以新區段位址來取代先前檔案結束數值，而零值便以檔案結束值來替代。

但是，FAT 配置方法可能造成大量磁頭尋找，除非 FAT 以快取方式存放。磁頭必須移到分割的開頭以讀取 FAT，並尋找中區段的位址，然後再移到區段本身所在的位置。在最糟糕狀況下，對每一區段都要移動 2 次。有一項優點是，隨機存取可以最佳化，因為磁頭可以藉由讀取 FAT 的資訊而找到任何區段位置。

4.4.3 索引分配(Indexed Allocation)

用鏈接分配可解決外部斷裂和連續分配必須做檔案大小宣告的問題。但是，因缺乏一個 FAT，鏈接分配無法使用直接存取，因為檔案的各個區段散佈在磁碟中。更重要的是，指向各區段的指標也是散佈在整個磁碟中而且需要依序取出。索引分配可以解決上述的問題，因為它把所有的指標都集中起來放在一個地方：索引區段(index block) 。



每個檔案都有自己的索引區段，那就是一個由磁碟中區段位址所組成的陣列。索引區段中第 i 個單元指向檔案的第 i 個區段。而在目錄中存放著索引區段的位址(上圖)。

當要建立檔案的時候，再索引區段中的所有指標都設成 nil(無)。當第 i 區段首先被寫入資料以後，就從可用空間表中移去一個區段，並且把它的位址放入第 i 個索引區的單元中。

索引分配法可以提供直接存取，而且不會有外部斷裂現象發生。再磁碟中任何地方的任何可用區段都一定可以用來滿足某些要求。

索引分配法會遭遇到浪費空間的問題。索引區段中指標佔用的空間一般比鏈結分配指標佔用的還要多。大多數的檔案都很小。假設我們有一個只有一兩個區段大的檔案。若用鏈接的分配，那麼每個磁區我們只損失一個指標(一個或兩個)所佔用的空間。若是使用索引分配即使只有一、兩個指標不是 nil(被使用了)，我們一樣要分配整個索引區段給它使用。

這個觀點引申出到底索引區段該有多大的問題。每個檔案都必須有一個索引區段，所以我們希望索引區段愈短愈好。如果索引區段太小的話，對於一個大檔案來說恐怕無法容納它的所有指標。因此我們需要一些技術來處理這種情況：

- 鏈結方式　索引區段一般是一個磁碟區段。因此它可以直接由它自己來讀或寫。為了允許大型檔案使用，我們可以把幾個索引區段鏈接在一起。例如，一個索引區段可以保存一個提供檔案名稱的小引子，並且將前 100 組區段的位址存在裡面。下一個位址(索引區段的最後一個字)是 nil 或是指向另一個索引檔案的指標(對某大檔案來說)。
- 多層索引　一種不同的鏈結表示法是使用第一層索引區塊指向第二層索引區塊組，然後再指向檔案區塊。為了存取一個區塊，作業系統使用第一層索引找出第二層索引區塊，再用這個區塊找出想要的資料區塊。這個方法可以延續到第三或第四層，這是依照想要的最大檔案之大小決定。使用 4096-位元組的區塊，我們可以儲存 1024 個 4-位元組指標在索引區塊之中。兩層的索引允許 1,048,576 個資料區塊，則它允許一個檔案最大到 4G bytes。
- 組合方式　另一種使用於 BSD UNIX 系統中的方式為保存裝置目錄中的前 15 個(比如說)索引區段指標。這些指標的前 12 個指向直接區段；也就是說，它們直接包含保有檔案資料的區段位址。因此較小檔案(不超過 12 個區段的檔案)的資料便不需要分離的索引區段。如果每個區段的大小為 4096 個位元組，那麼就有 48K 位元組的資料可供直接存取了。而接下來的三個指標則是指向間接區段，單一的間接區段位址事實上就是第一個間接區段指標，但它的內容卻不是資料，而是包含著資料的區段位址，因此，就會有雙重間接區段(double indirect block)的指標出現，區段的位址所包含的資料為區段位址，這些位址才是指向實際資料區段的指標的包含者。最後一個指標則包含著一個三重間接區段(triple indirect block)的位址。在這種方法之下，可以配置給一個檔案的區段數目超過作業系統或系統呼叫時所用的 4 位元組檔案指標。32 位元的檔案指標只可達到 2^{32} 位元組(也就是 4G 位元組)。

索引配置法遭遇到和鏈接配置法一樣的效率問題。特別是，索引區段可能以快取方式存在記憶體，但資料區段卻可能散佈在磁碟中的各個地方。

4.4.4 分配效能差異

在各種不同分配方法之間儲存效能差異很大。同時對磁碟上區段的存取所花費的時間也各有不同。這個因素在作業系統製作時該選擇的正確方法是非常重要的。

在比較效能的時候有一項困難，就是如何去判斷這些系統到底是如何使用的。無論是用哪一種存取方式，連續分配只需要用一次存取來得到一個磁碟區段。因為我們可以將檔案的起始位址保存在磁蕊(core)中，所以我們可以立刻算

出第 i 區段的磁碟位址(或下一個區段的位址)並且將它直接讀出。

對於鏈接分配來說,我們也可以將下一個區段的位址保存在記憶體中並且直接讀取。這種方式適用於循序存取,但是對於直接存取,對第 i 個區段的存取可能需要做 i 次讀磁碟的工作。所以鏈接分配不該用在需要直接存取的事例上。

索引分配就更複雜了。如果索引區段早已存在記憶體中了,那麼就可以直接做存取的工作。但是,將索引區段保存在記憶體中需要大量的記憶空間。如果記憶空間並不夠用,那我們就必須先讀索引區段然後再去讀想要的資料區段。對於兩層索引來說,我們最起碼要讀兩次索引區段。對於非常大的檔案來說,若是要存取的區段是幾乎在檔案尾端的,那麼在最後讀取該區段之前,我們必須要讀過所有在指標鏈接上的索引區段。因此索引分配的性能要完全看索引結構、檔案大小和所要找的區段來決定。

在某些系統結合了連續分配與索引分配,而在小的檔案(最多三或四個區段)上使用連續分配並且當檔案成長較大時自動地改變為索引分配。由於大多數的檔案都很小,而且連續分配對小檔案是很有效率的,所以平均的效能很好。

4.5 空間管理(Free-Space Management)

因為僅有有限的磁碟空間可用,因此新建的檔案就必需使用被除掉的檔案之前所佔用的空間。為了要保留可用磁碟空間的磁軌,系統採用了一種可用空間串列(free-space list)。可用空間串列記錄了所有可用磁碟的區段(block)(也就是尚未分配給任何檔案的記憶空間)。若要建一個檔案,我們就搜尋可用空間串列來找尋所需的空間量,並且將它分配給新的檔案。於是這些空間就從可用空間串列中移掉。當一個檔案被除掉之後,它的磁碟空間就加到可用空間串列上。

4.5.1 位元向量(Bit Vector)

通常來說,可用空間串列用位元映像(bit map)或位元向量(bit vector)來執行工作。每個區段用一位元來表示。如果某區段是可用的,該位元為 0;若某區段已被配置,該位元為 1。

例如,假設有一磁碟的第 2、3、4、5、8、9、10、11、12、13、17、18、25、26 和 27 區段是可用的;那麼可用空間位元映像就是

001111001111110001100000011100000…

這個方法的主要優點在於簡單,而且,它在磁碟連續 n 個可用區段的搜尋上相當具有效率。Macintosh 作業系統循序地檢查位元映像中的每一字元組以找出第一個可用區段,它去查看是否為非 0 的數值,因為一個所有位元為 0 的字元組就代表了一組以配置的區段。第一個非 0 字元組是檢查第 1 個位元。區號號碼的計算如下:

(每個字元組的位元數目)x(內容為 0 的字元組數目)+第 1 個位元的偏移量
但是，除非整個向量保留在主記憶體做為大部分存取，否則位元向量是無效用的。對較小的磁碟保持位元向量在主記憶體是可行的，就如微電腦上一樣，但不能在大電腦上。一個 1.3G 磁碟(以 512 位元組為區段單位需要 320K 的位元映像以追蹤可用的區段。若將 4 個位元向量群聚成一個 cluster 則可以降低為 80K 位元組。

4.5.2 鏈接串列(Linker List)

另外一種方法是將所有可用磁碟區段鏈接在一起。由一個指標指向第一個可用區段。這一個區段又包含有指向下一個區段的指標，如此下去。在上面那個例子中，我們將保存一個指向第 2 區段的指標，把它當作第一個可用區段。第 2 區段指向第 3 區段，第 3 區段指向第 5 區段，第 5 區段指向第 8 區段。這種方法並不是非常有效率的，因為當我們檢視這個表的時候，必須把所有區段讀入，這將需要很龐大的時間。不過，通常作業系統只需要一個可用區段，以便將該區段分配給一個檔案，因此可用串列中的第一個區段可以用上。

4.5.3 群組(Grouping)

修正可用串列方法是將 n 個區段的位址存放在第一個區段中。前 n-1 個是實際可用的區段。最後一個是存放另一個含有另外 n 個可以使用的區段在磁碟上的位址。這樣執行的重要性是使得大量可用區段的位址可以很快地尋獲。

4.5.4 計數(Counting)

另一種方法乃是利用一般上都是需多連續區段同時為可用或同時被分配掉的事實，尤其是在使用連續分配的時候。因此，並不需要使用含有 n 個可用磁碟位址的表，我們只需要記住第一個可用區段的位址及其後面所連接的 n 個連續可用區段的數量即可。於是在可用空間表中的每個項目中只包含了一個磁碟位址和一個數目值。雖然每個項目比起單只磁碟為只要佔用較多的空間，但是整個串列反而變的短些，只要該數值平均上比 1 大就可以了。

第5章 輸出入系統

個人電腦最主要的兩個工作為輸入/輸出與資料處理，在許多情況下，主要的工作為 I/0，而資料處理只是偶發事件。作業系統在電腦 I/0 方面扮演管理及控制 I/0 操作與 I/0 裝置的角色，對於作業系統設計者而言，最關心的問題莫過於如何控制與電腦相連之裝置，由於 I/0 裝置在功能與速度方面變化極大(如滑鼠、硬碟與光碟機)，因此需要許多不同的功能來加以控制，這些功能構成核心的 I/0 子系統，將核心其他部份與管理 I/0 裝置可能涉及的複雜度區隔開來。

現在軟體與硬體介面逐漸趨於標準化，這個趨勢有助於改良後之裝置整合至既存電腦與作業系統之中，另一方面我們也看到更多 I/0 裝置的出現，有些新裝

置和以往的裝置差異極大，以致於將它們整合至既存電腦與作業系統的工作變得不太容易。I/O 硬體元件(例如運接埠、匯流排與裝置控制器)都包含在廣泛的 I/O 裝置種類之中，為了將不同裝置的細節與相異之處隱藏起來，作業系統核心設計成使用裝置驅動器模組(device driver module)，裝置驅動器則代表一個與 I/O 子系統相通之統一裝置存取介面，大部份與系統呼叫一樣，提供應用程式與作業系統間的標準介面。

5.1 I/O 硬體

電腦負責操作許多不同種類的裝置，包含儲存裝置、傳輸裝置(網路卡、數據機)，以及介面裝置(螢幕、鍵盤、滑鼠)。不管電腦之 I/O 裝置有多少種類，我們只需要了解該如何安裝這些裝置以及軟體如何控制硬體的概念即可。

裝置透過纜線或甚至在空中發送訊號即可與電腦系統通訊，藉由所謂的連接埠(port)與機器互通訊息，若一至多個裝置都使用相同纜線，這樣的連接方式就稱為匯流排(bus)。匯流排由一組纜線組成，並規定一組可在纜線上傳送的訊息格式；這些訊息都以電壓的形式以及定義好的時間間隔傳至纜線之中，當裝置 A 與裝置 B 相連，而裝置 B 與裝置 C 相連，裝置 C 連至某電腦之連接埠時，這樣的安排稱之為串鏈(daisy chain)，通常以匯流排的方式操作。

控制器(controller〉為可操控連接埠、匯流排或裝置的電子零件組合；序列埠即為簡單裝置控制器的一個例子，它為電腦中的獨立晶元，控制在序列埠纜線上的訊號，相反地，SCSI 匯流排控制器就不這麼簡單了，由於 SCSI 協定較為複雜，因此 SCSI 匯流排控制器常製作成可插入電腦的獨立電路板，基本上包含處理器、微程式碼及用來處理 SCSI 協定訊息的私有記憶體，某些裝置擁有自己內建的控制器，若你有機會看到磁碟機的話，將會發現另一端與電路板相連，此板即為磁碟控制器，它根據特定連接(例如 SCSI 或 IDE)之協定加以製作，本身也擁有微程式碼與處理器可以執行許多工作，例如毀壞區域映對(bad-sector mapping)、預先載入(prefetching)、暫存(buffering)與快取(caching)。

控制器擁有一至多個暫存器可以儲存資料與控制訊號，處理器透過讀取暫存器之中的資料與將資料寫入暫存器的動作跟控制器通訊;通訊的方法可以藉由使用特殊 I/O 指令來傳輸要送往某 I/O 連接埠位址的位元或字元，I/O 指令驅動匯流排線選擇適當的裝置，再將位元輸入或寫出到裝置暫存器；另一個方法為使用裝置控制器支援記憶體映對(memory-mapped)I/O，在這種情況下，裝置控制暫存器映對至處理器的位址空間，CPU 透過使用標準資料傳輸指令讀寫裝置控制暫存器的方法來執行 I/O 要求。

某些系統兩種方法都使用，例如，PC 皆使用 I/O 指令控制某些裝置，並用記憶體映對 I/O 控制其他裝置。下表為部分一般 PC I/O 連接埠位址，圖形控制器的 I/O 連接埠作為基本控制操作之用，但控制器擁有大型記憶體映對區以便保存螢幕內容，此控制器透過將資料寫入記憶體映對區域的方法，將輸出送至螢

幕，此控制器依據記憶體中的內容產生螢幕影像，此方法極為簡單。

I/O 地址範圍	裝置
000-00F	DMA 控制器
020-021	中斷控制器
040-043	計數器
200-20F	遊戲控制器
2F8-2FF	串列埠(次要)
320-32F	硬碟控制器
378-37F	平行埠
3D0-3DF	圖形控制器
3F0-3F7	磁碟片裝置控制器
3F8-3FF	串列埠(主要)

圖形記憶體要比發出數百萬命令要來得快速許多；但是這個方法有一個缺點，因為一般軟體發生之錯誤類型大多為使用指標不當，即指向不被允許的記憶體區域，所以記憶體映對裝置暫存器在遇到偶爾被修改的情況時，極易受到影響，當然，不當使用(protected memory)可減少風險。

I/O 連接埠基本上包含四個暫存器，分別為狀態(status)、控制(control)、資料輸入(data-in)及資料輸出(data-out)暫存器。狀態暫存器包含可被主機讀取的位元資料，這些位元指出目前執行指令是否完成、在資料輸入暫存器中否可讀取，以及是否有裝置錯誤發生之類的狀態；控制暫存器則可由主機起始指令或改變裝置模式，在序列埠控制暫存器中的特定位元可以半工通訊，而另一位元則啟動同碼檢查，第三個位元則將字元長度設為 7 或 8 位元，再由其他位元選擇序列埠支援速度的其中之一。主機由資料輸入暫存器讀取輸入，再將資料寫入輸出暫存器送至輸出裝置；基本上，資料暫存器為 1 組，某些控制器尚擁有 FIFO 晶元，於是可以保留輸入或輸出資料數個位元組大小，使得除了資料暫存器本身容量大小之外，可以再擴展控制器的容量；可保有小量資料，直到裝置或主機能夠接收為止。

5.1.1 查詢(polling)

要找出主機與控制器之間的完整協定是一項挑戰，但基本概念卻極簡單。假設有 2 個位元協調控制器與主機之間的生產-消費關係，控制器透過設定在狀態暫存器內的忙碌位元表示本身狀態。設定(set)位元表示將此位元值設為 1，而清除元表示將此位元值設為 0，當控制器忙於工作時，即設定忙碌(busy)位元，而當準備好可以接收下一個指令時，即清除忙碌位元。主機藉由設定在指令暫存

器(command register)中的指令就緒 (command-ready)位元來表示本身狀態，當有可供控制器執行的指令時，主機即設定指令就緒位元值下列為主機將輸出寫至一連接埠，並且協調與握制器之間的動作：

1. 主機將重複讀取忙碌位元，直到位元值被清除。
2. 主機設定在指令暫存器中的寫入位元(write bit)，並將位元組寫入資料輸出(Data-out)暫存器。
3. 主機設定指令就緒位元。
4. 當控制器發現指令就緒位元已經設定完成，即設定忙碌位元。
5. 控制器讀取指令暫存器並發現寫入指令，則自資料輸出暫存器讀取位元組，並且執行必須的裝置 I/O 處理。
6. 控制器清除指令就緒位元，並清除在狀態暫存器中的錯誤位元(error bit)，表示裝置 I/O 已經成功，再清除忙碌位元以便表示動作完成。

每個位元組皆重複此迴路。

在步驟 1 之中，主機處於忙碌等特(busy waiting)或查詢(polling)的狀態：這是一個迴路，它一再讀取狀態暫存器內之值，直到忙碌位元被清除為止；若控制器與裝置速度夠快，這個方法就足夠了，但是也可能因為主機切換到另一個工作，而使得等待時間過長。對某些裝置而言，主機必須提供裝置快速的服務，否則可能造成資料遺失，例如，當資料正流入序列埠或自鍵盤取得時，若主機未能及時讀取這些資料，在控制器內的小型緩衝器將會滿溢而造成資料遺失的情形。

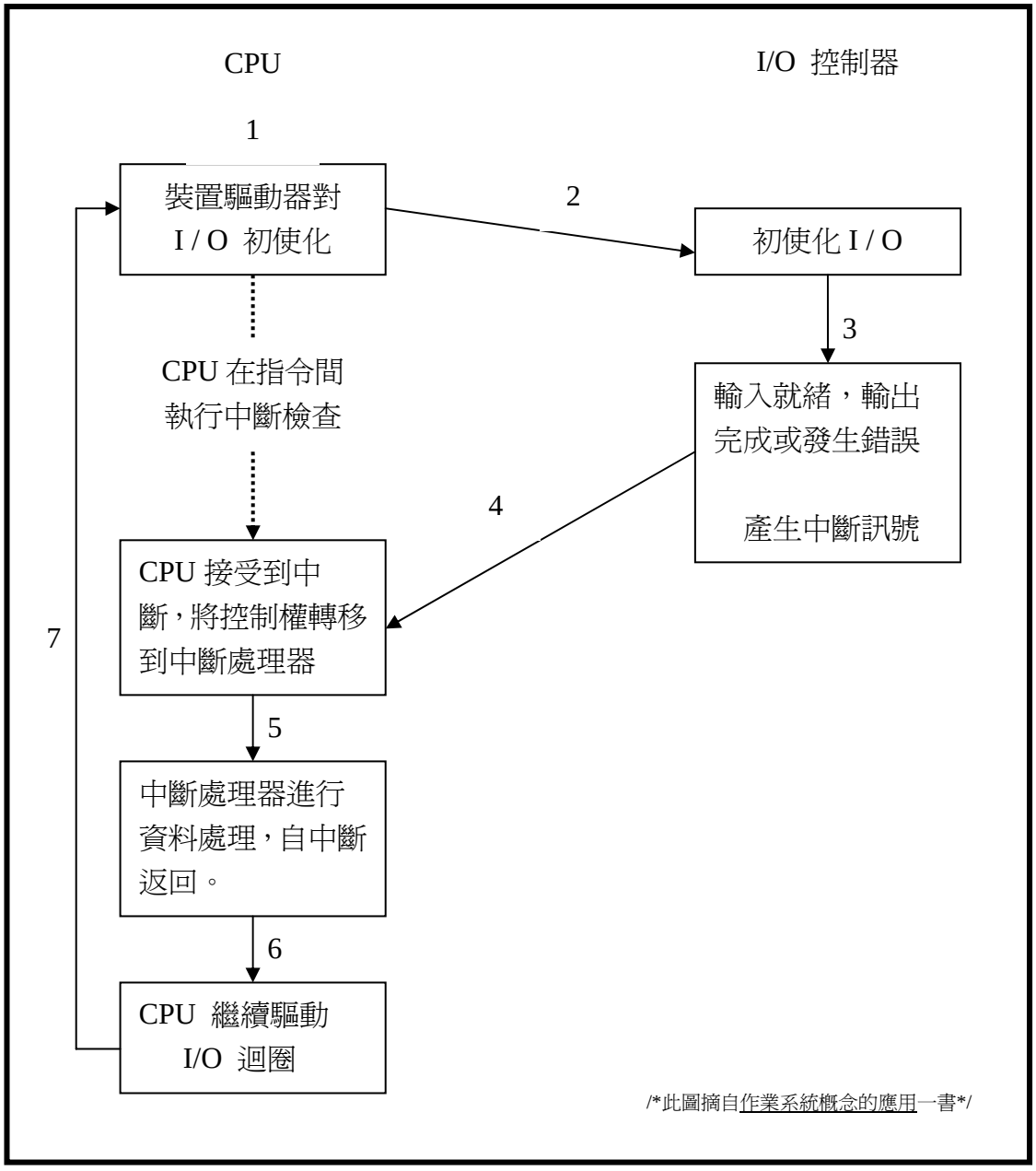
在許多電腦架構之中，用三個 CPU 指令循環即已足夠用來查詢一個裝置之狀態：讀取(read)裝置暫存器、利用邏輯-和(logical-and)運算抽取狀態位元，以及若不為零時產生分支(branch)。更清楚地說，基本查詢操作就已足夠，但當查詢需重複執行時，就會顯得沒有效率。發生找不到已就緒的裝置可提供服務，同時其他 CPU 程序仍未獲得解決的情況；像這種情況，若能在裝置就緒可提供服務時，安排硬體控制器通知 CPU 可能會較有效率，如此就不需 CPU 為了完成 I/O 動作，一再重複地查詢各裝置。這種允許裝置通知 CPU 的硬體機制稱為中斷(interrupt)。

5.1.2 中斷(Interrupt)

中斷是導致電腦變換其正常指令順序執行的訊號，許多不同的情況都會產生這樣的訊號，例如，輸入輸出作業的結束、超過事前設定時間間隔，或者試圖除以零等。

電腦上有四種中斷類型。SVC 中斷(第一類)是當 CPU 執行監督者呼叫指令時產生。這個指令是程式用來要求作業系統做某些功能的。程式中斷(第二類)是當程式執行期間某些情況發生時產生的，例如執行一些不合法的指令。定時器中斷

(第三類)是由 CPU 的時間間隔定時器產生。此定時器含有暫存器，CPU 每使用一毫秒，暫存器的值就減一；當值變為零時，引發定時器中斷，此時作業系統用來管理一使用者程式控制機器的時間。輸入/輸出中斷(第四類)是由輸入/輸出通道或設備產生。大部分這種中斷是當某些 I/O 操作正常時引發。然而，I/O 中斷也可能表示各種不同的錯誤情況。



上圖為基本中斷機制的運作，CPU 硬體擁有一個稱為中斷要求管線(interrupt request line)的纜線，CPU 會在每執行一個指令後檢查，當 CPU 偵測出某控制器已將一訊號加至中斷要求管線時，CPU 將會先儲存少量狀態值(例如目前的指令指標值)，並跳至記憶體中某固定位址，執行中斷處理器常式(interrupt handler routine)；中斷處理器決定中斷發生的原因，執行必要的

處理，並產生自中斷返回的指令，以返回 CPU 在發生中斷前之原執行狀態；像這種由裝置控制器將訊號加至中斷要求管線的方法，我們稱為引發(raise)中斷，CPU 捕捉這個中斷並分派給中斷處理器，而且處理器會在提供裝置服務之後清除中斷，下圖為中斷-驅動 I/O 迴路之整理。

此基本中斷機制使得 CPU 得以回應非同步事件，如當裝置控制器已呈就緒狀態時，即可提供服務；但在現代作業系統之中，需要更複雜的處理要求。首先，我們必須要在中心處理(critical processing)執行時，亦能夠服從中斷處理器之要求的能力；第二，我們需要可以將裝置中斷訊息分派至適當中斷處理器的有效方法，而不需查詢所有裝置，以便得知到底是哪個裝置引發中斷；第三，我們需要多層(multilevel)中斷，使得作業系統可以分辨高優先權(high-priority)與低優先權(low-priority)中斷的不同，並可依適當的緊急度加以回應。在現代電腦硬體之中，CPU 與中斷控制器(interrupt controller)硬體已經提供這三項特點。

大部份的 CPU 有兩種中斷要求管線，一為無遮罩中斷(nonmaskable interrupt)，保留給不可回復記憶體錯誤之事件使用；第二種為遮罩〈maskable)中斷：CPU 可以在執行不可被中斷之中心處理之前先關閉這類中斷，像裝置控制器都使用遮罩中斷來要求服務。

中斷機制接受一位址，此位址為一可以自某集合中選擇特定中斷處理常式的數字。在大部份的架構之中，此位址為中斷向量(interrupt vector)表格的起始位址，此向量含有特定中斷處理器的記憶體位址。中斷向量方法的目的在於讓中斷處理器不需搜尋所有中斷發生的可能來源，即可決定到底那一個裝置需要服務。事實上，電腦擁有遠較中斷向量表中含有位址元素為多的裝置，一般解決方法為使用中斷串鏈(interrupt chaining)的方法，在中斷向量表中的每個元素都指向由中斷處理器組成之串列前端。當中斷發生時，則會一個個呼叫對應串列內之處理器，直到可回應要求之中斷器找到為止。此架構為下列兩個問題的折衷處理，一為大量中斷向量表造成多餘負擔，另一為分派至單一中斷處理器較無效率。

中斷機制也製作系統的中斷優先權層(interrupt priority levels)，此機制使得 CPU 能夠在不關閉所有中斷的前提下，處理低優先權的中斷，並使高優先權中斷能夠先於低優先權中斷執行。

現代作業系統透過幾種方法與中斷機制通訊，在開機時作業系統先探查硬體匯流排，以得知目前存在哪些裝置，並將相關中斷處理器載入至中斷向量表中；在執行 I/O 期間，所有就緒的裝置控制器都可能引發中斷，這些中斷意味著輸出已經完成、輸入資料就緒，或已測得某個錯誤發生；中斷機制也用來處理許多不同的例外(exception)，例如被除數為零、存取違法或不存在之記憶體位址，或企圖在使用者模式執行特權指令。

作業系統也可以使用其他有效率之硬體方法：儲存處理器狀態值，並接著呼叫核心之特權常式。例如，許多作業系統使用虛擬記憶體分頁的中斷方法，分頁失誤(page fault)是一個會引發中斷的例外，此中斷將會先懸置目前之處理程

序，並跳至核心分頁失誤處理器；此處理器可儲存處理程序狀態、將處理程序移至等待佇列，執行分頁快取(page cache)管理、並安排 I/O 操作取得分頁、再安排另一處理程序繼續執行，最後自中斷返回。

另一例子為系統呼叫的製作，所謂的系統呼叫為一個可以被應用程式呼叫以便啟用核心服務的功能，系統呼叫會先檢查應用程式傳入的參數，建立資料結構並將它傳給核心，接下來再執行一個稱為軟體中斷或跳脫(trap)的特殊指令，此指令擁有可確認特定核心服務的運算子；當系統呼叫執行跳脫指令時，中斷硬體將儲存使用者程式碼的狀態值，切換至監控者模式，再分派至實作要求服務之核心常式。此跳脫指令與裝置中斷比較起來，屬於較低的中斷優先權，表示在裝置控制器的 FIFO 佇列尚未滿溢與遺失資料之前，應用程式執行之系統呼叫與裝置控制器服務兩者相較之下較不緊急。

中斷亦可用來管理核心內的控制流程，如要完成磁碟機讀取的處理程序，有一個步驟需先將資料自核心記憶體複製到使用者緩衝器，這個複製動作得花點時間，但卻不緊急。也就是說，它不會阻礙其他具高優先權的中斷處理；另一個步驟則是此磁碟機啟動下一個懸置 I/O，這個動作具有高優先權，亦即若能有效使用磁碟機，我們即可盡快在前個指令完成時，立刻開始下一個 I/O；因此，完成磁碟讀取的核心程式碼由一組中斷處理器製作，具高優先權之處理器先記錄 I/O 狀態、清除裝置中斷、開始下一個懸置 I/O、再引發一低優先權之中斷完成工作；稍後，當 CPU 已不處理高優先權之工作時，低優先權中斷就會被分派，映對之處理器藉由將資料自核心緩衝器複製至應用程式空間，完成使用者層次 I/O，接下來再呼叫排班程式將應用程式置於就緒佇列。

總之，現代作業系統使用中斷來處理非同步事件，並跳至核心中的監控者模式，為了讓緊急的事件得以優先處理，現代電腦使用一套中斷優先權方法，裝置控制器、硬體錯誤，及系統呼叫皆會引發可驅動核心常式的中斷，由於中斷處理佔去極多時間，因此為了良好的系統執行效率，需要有效的中斷處理方法。

5.1.3 直接記憶體存取(Direct Memory Access)

對於一個需要做大量傳輸的裝置(例如磁碟機)而言，若使用昂貴及一般性用途的處理器來監看狀態位元，以及將資料以一次一位元組大小的方式送入控制暫存器之中，似乎有點浪費，這樣的程序稱為程式化 I/O(programmed I/O，PIO)。許多電腦利用將這類工作交由特殊用途處理器負責，來避免加重主要 CPU 的負擔，這種特殊用途處理器稱之為直接記憶體存取(DMA)控制器；當要對 DMA 控制器做初始化時，主機會將 DMA 指令區塊寫入記憶體，此區塊包含指向傳輸來源的指標、指向傳輸目的地的指標，與傳輸資料數目。CPU 將此指令區塊位址寫至 DMA 控制器之中，接下來就去處理其他工作，這時 DMA 控制器會直接操作記憶體匯流排，將位址置於匯流排上，不需主要 CPU 的輔助即可執行傳送工作。

在 DMA 控制器與裝置控制器之間的程序透過一組稱之為 DMA 要求

(DMA-request)與 DMA 回覆(DMA acknowledge)的纜線完成，當有資料需要傳送時，此裝置控制器即將一訊號置於 DMA 要求纜線上，此訊號會讓 DMA 控制器抓取記憶體匯流排，將目的位址置於記憶體位址纜線上，再將訊號置於 DMA.回覆纜線。當裝置控制器收到 DMA 回覆訊息時，即將資料傳至記憶體，並移除 DMA 要求訊息；當整個傳輸結束時，DMA 控制器會中斷 CPU。當 DMA 控制器取得記憶體匯流排時，CPU 即不可存取主記憶體，而只能存取主要與第二快取(cache)中的資料項；雖然這樣可能會降低 CPU 的計算速度，但將資料傳輸的工作交給 DMA 控制器處理，一般的確可改善整個系統的執行效率；某些電腦架構讓 DMA 使用底層實際記憶體位址，但其他則執行直接虛擬記憶體存取(direct virtual memory access, DVMA)，使用需經過虛擬至實際記憶體位址轉換的虛擬位址，DVMA 可在兩記憶映對裝置之間直接進行傳輸，而不需 CPU 的介入或使用主要記憶體。

當系統核心為保護模式時，作業系統通常會防止處理程序直接發出裝置命令，這個方法可以防止資料遭受存取控制衝突(violation)，而且可以使系統免於裝置控制器的錯誤使用而導致系統當機；除此之外，作業系統讓具有足夠優先權的處理程序，得以使用可存取底層硬體之低層操作功能；當核心沒有記憶體保護時，處理程序可以直接存取裝置控制器，使用此直接存取方法可以獲得較高執行效率，因為它可以避免核心通訊、內容切換、及核心軟體之分層。但是，它卻會影響系統保密與穩定度。一般作業系統的作法是保護記憶體與裝置，因此系統會試著防止有錯誤或懷有惡意的應用程式執行。

5.2 I/O 介面

我們需找出一般性特徵才可以將 I/O 裝置的細節相異性抽離開來，這些一般性特徵可由一組標準化功能加以存取，我們稱之為介面(interface)，真正的相異點都被封裝在所謂的裝置驅動器(device driver)這個核心模組之中，由系統內部為每種裝置量身訂做，但外觀皆為標準介面

裝置驅動器層的目的在於將各裝置控制器間之相異點隱藏起來，不讓核心的 I/O 子系統觸及，正如 I/O 系統呼叫將裝置行為以少量且一般性的類別包裝起來，以達到隱藏應用程式硬體相異點的目的。將 I/O 子系統獨立於硬體之外，可簡化作業系統發展者的工作。但是，對裝置硬體廠商而言，每種作業系統都會擁有自己適用的裝置驅動器介面標準，一個特定裝置可被多種裝置驅動器驅動，例如，MS-DOS、Windows、Windows NT 與 Solaris 上的驅動器。

就應用程式存取目的而言，最希望的就是作業系統能夠將眾多相異點隱藏起來而且裝置設計能夠集中在少數傳統類型。大部份的裝置存取它的最後型態也趨向於普遍性與可用性，雖然真正的系統呼叫可能因作業系統而不同，但裝置類的介面卻已標準化，主要的存取功能含有區段(block)I/O、字元串列、I/O 記憶體映對檔案存取，以及網路承接口(socket)。作業系統亦提供特殊的系統呼叫以便存取其他如時鐘與計時器這類的裝置。某些作業系統更提供一組可用於圖形顯

示、視訊，及聲音裝置上的系統呼叫。

大部份的作業系統也將來自某應用程式之任意指令傳給裝置驅動器的跳脫(escape)或後門(back door)系統呼叫；在 UNIX 之中這類的系統呼叫為 ioctl ，ioctl 系統呼叫讓應用程式可以存取任何裝置驅動器上的功能，而不需加入新的系統呼叫，ioctl 系統呼叫共有三個參數，一個是透過指向由裝置管理的硬體裝置方法與應用程式連結的檔案描述子;第二個參數可選定將使用裝置中的哪個指令；第三個參數為一指標，可指向記憶體之中的任一資料結構，因此應用程式與驅動器可透過任何必須的控制資訊相互傳訊。

5.3 核心 I/O 子系統(Kernel I/O Subsystem)

5.3.1 I/O 排班程式(I/O Scheduling)

所謂安排一組 I/O 要求，意指如何找出好的執行順序，應用程式發出系統呼叫的順序幾乎都不是最好的安排，使用排班程式可以改進整個系統之執行效率，允許執行程序們公平地享有裝置存取權力，而且可以降低 I/O 完成指令的平均等待時間；假設磁碟讀取臂目前位置接近磁碟的起端，應用程式 1 要求一接近磁碟尾端的區塊，應用程式 2 要求一接近磁碟開頭的區塊，而應用程式 3 則要求一位於磁碟中央的區塊。很明顯地，此作業系統若能依 2、3、1 的順序為應用程式提供服務，就可以減少磁碟讀取臂移動的距離；重新安排服務順序就是 I/O 排班的主要工作，作業系統發展者透過保有每個裝置之要求佇列進行排班工作，當應用程式發出阻隔 I/O 系統呼叫時，立刻將要求置於裝置佇列之中，I/O 排班程式將重新安排佇列的順序，以改善整個系統效率以及應用程式之平均回應時間；作業系統本身也可以試著講求公平性，這些應用程式即可公平享受系統提供之服務，或讓緊急要求馬上獲得優先服務。

I/O 子系統改善電腦效率的其中一個方法為安排 I/O 操作順序，另一個方法則為使用主記憶體或磁碟機上的儲存空間，即所謂的暫存、快取及 spooling。

5.3.2 緩衝(Buffering)

所謂緩衝器是在兩個裝置或裝置與應用程式之間傳輸資料時，可以儲存資料的記憶體區域。要使用緩衝的原因有三點：一是為了解決資料串列之生產者與消費者在速度上不相等的問題；假設，有一個檔案想要透過數據機傳輸之後儲存於硬碟之中，數據機的速度大約此硬碟慢了一千倍，因此在主記憶體中有一個暫存器可以暫時累積來自數據機的資料，當整塊暫存器資料送達時，暫存器即在單一操作內將資料寫回磁碟機；由於磁碟寫入動作並不立刻發生，而且數據機仍需要一塊可以儲存額外接收資料的空間，而使用兩個暫存器；自數據機送達資料填滿第一個暫存器後，即發出磁碟寫入動作，接下來數據機開始再趁第一個暫存器執

行寫入動作時，將資料填滿第二個暫存器，在數據機將第二個暫存器填滿之前，第一個暫存器的磁碟寫入動作應已完成。因此數據機可在第二暫存器進行磁碟寫入時，即切回第一個暫存器；此雙向緩衝(double buffering)的方法降低生產者與消費者之間的資料量，並放鬆它們之間的時間需求限制。

緩衝的第二個用途是用在不同資料傳輸大小的裝置之間作調整，這些特別的不一致，通常是在電腦網路連結之中發生，其中的緩衝器是廣泛地使用在分段和訊息的重組。在傳送端，大型的訊息是分割成許多小型的網路封包，這些封包傳送在整個網路，接收端將它們放置在一個重組緩衝器之中，以便組成來源資料的影像。

緩衝的第三種用途是用來提供應用 I/O 的複製語法(copy semantics)。假設應用有一個資料緩衝器，而這些資料想要寫入磁碟之中，它會呼叫寫入(write)這個系統呼叫，提供給緩衝器一個指標以及一個用來指定多少位元組的整數，以便作寫入的動作。在系統呼叫返回之後，假如應用改變緩衝器的內容，那將會發生什麼事呢?採用複製語法時，寫入磁碟之資料版本保證是應用系統呼叫的時候，與在應用的緩衝器之中任何後來的改變有關之版本。作業系統可以保證複製語法是一種簡單的方式，對於寫入(write)這個系統呼叫而言，在控制權回到應用之前，複製應用資料到核心緩衝器之中。磁碟寫入是由核心緩衝器完成，因此對於應用緩衝後來的改變沒有影響。在核心緩衝器與應用資料空間之間的資料複製，在作業系統之中是很平常的，由於有明確的語法，就可以忽略由這些操作所導致的額外負擔。

5.3.3 快取(Caching)

快取是一個持有資料複製的快速記憶體，存取已經快取住的複製是比存取原有的資料要來得有效率。例如，目前正在執行之行程的指令是儲存在磁碟之中，已經快取在實際的記憶體之中以及再複製到 CPU 的次要和主要快取之中。緩衝器和快取之間的不同，在於緩衝器可能只有持有資料項的現在備份，而快取只持有一個存在於其他位置的項目之快速儲存的複製。

快取和緩衝是兩個不同的功能，但是有時候這兩個目的可以使用同一塊記憶體範圍。例如，為了保存複製語法和使得磁碟 I/O 有效地排班，作業系統在主記憶體之中使用緩衝器來保存磁碟的資料。這些緩衝器也像快取一樣地使用，以便提升這些由應用所共用的檔案或者它是經常地寫入和再讀出的檔案之 I/O 效率。當核心接收到一個檔案 I/O 要求，核心首先存取這個緩衝器，來看看是否這個檔案的區域已經存在於主記憶體之中。如果是，則實際磁碟的 I/O 就可以避免或延後。因此，組合大量的傳送以便允許有效的寫入排班。

5.3.4 Spooling

Spool 是一個緩衝器，它是一個用來保留不能接收揮入之資料串列的裝置之輸出。雖然一部印表機一次只能為一個工作服務，但是許多應用可能同時想要列印它們的輸出，而且不能將它們的輸出混在一起。作業系統採用攔截所有到印表機之輸出的方式來解決這個問題，每一個應用的輸出都被 Spool 在個別的磁碟檔案之中。當一個應用完成列印之後，將 Spooling 系統的佇列之中符合的 Spool 檔案輸出到印表機，Spooling 系統一次複製一個已經在等待的 Spool 檔案到印表機。

某些像磁帶、印表機這類的裝置，無法應付來自多個並行應用程式的 I/O 要　求，Spooling 為作業系統解決協調並行輸出的方法之一，另一個解決並行裝置存取的方法是提供明白的協調功能。某些作業系統(包括 VMS)都提供互斥裝置存取支援，讓處理程序配置一個閒置裝置，當不需要時再將此裝置釋回。其他作業系統則強迫這類裝置最多只能開啟以某個數目為上限的檔案描述子，許多作業系統提供可以讓處理程序彼此協調互斥使用的功能，例如，Windows NT 提供等待(wait)系統呼叫，直到裝置物件變成為可用狀態，它也在開啟(open)系統呼叫之中加入參數，用來宣告允許存取其他並行執行緒內容的類型；在這些系統之中，由應用系統負責避免死結(deadlock)的情況發生。

5.3.5 錯誤處理(Error Handing)

使用保護記憶體的作業系統，能夠防止許多硬體與應用程式發生錯誤，因此真正的系統錯誤並非一般小型機械錯誤引起。裝置與 I/O 傳輸發生錯誤的情況有很多，有些可能是暫時性原因，例如網路負載過高，或永久性原因，例如磁碟控制器毀壞，作業系統大部份可以有效修復暫時性原因所發生的錯誤。例如，磁碟讀取錯誤可再重新讀取，而網路傳送錯誤則可視協定規格加以重送，但是，如果重要元件遭受永久錯誤，則作業系統就無法復原。

I/O 系統呼叫的一般作法為送回 1 位元的資訊，以便說明此呼叫的執行狀態為成功或失敗。在 UNIX 作業系統之中，有一個稱為 errno 的額外整數變數，可以用來回傳錯誤碼，大約有 100 個資料值用來說明錯誤發生的情況(例如，參數數目過多，不當指標，或尚未開啟檔案)；相反地，雖然某些硬體能夠提供相當詳細的錯誤資料，但作業系統並不能將這些資料傳送給應用程式。例如，當 SCSI 裝置發生錯誤時，SCSI 協定將會透過感應鍵值(sense key)得知錯誤狀況為何，例如硬體錯誤或非法要求；至於額外感應碼(additional sense code)可以說明像無效指令參數或自我測試錯誤之類的錯誤類別，而額外感應碼修飾子(additional sense code qualifier)則可提供更詳細的資料，例如哪一個指令參數發生錯誤，或哪一個硬體子系統在自我測試時發生錯誤等。甚至還有許多

SCSI 裝置會保有可供主機參考的內部錯誤記錄資訊。

5.4 I/O 要求

應用程式透過檔案名稱才能得以參考其中內容，在磁碟機之中，這是檔案系統的工作，透過檔案系統目錄之中的檔案名稱取得檔案的空間位置；例如，在 MS_DOS 之中，檔案名稱與一數字映對，指出位於檔案存取表格內的哪個項目，而且在這個項目說明此檔案佔有磁碟機的哪個區段；在 UNIX 之中，檔案名稱則與 inode 數字映對，內容為空間配置資料。

檔案名稱如何與磁碟控制器(硬體連接埠位址或記憶體映對控制器暫存器)發生關聯?

首先，先從 MS_DOS 作業系統來說明，在冒號前面的 MS_DOS 檔案名稱，其第一部份為指出特定硬體裝置之字串，例如，C：為主要硬碟之中每個檔案名稱的第一部份，原因是 C：表示此作業系統的主要硬碟為 C，C：映對至裝置表之中特定的連接埠位址，由於分號隔離子(separator)的緣故，每個裝置內的裝置名稱空間與檔案系統名稱空間分隔開來，此分隔作法讓作業系統在處理裝置其他功能關聯時相當容易；例如，可以容易地呼叫 spooling，讓檔案寫至印表機。

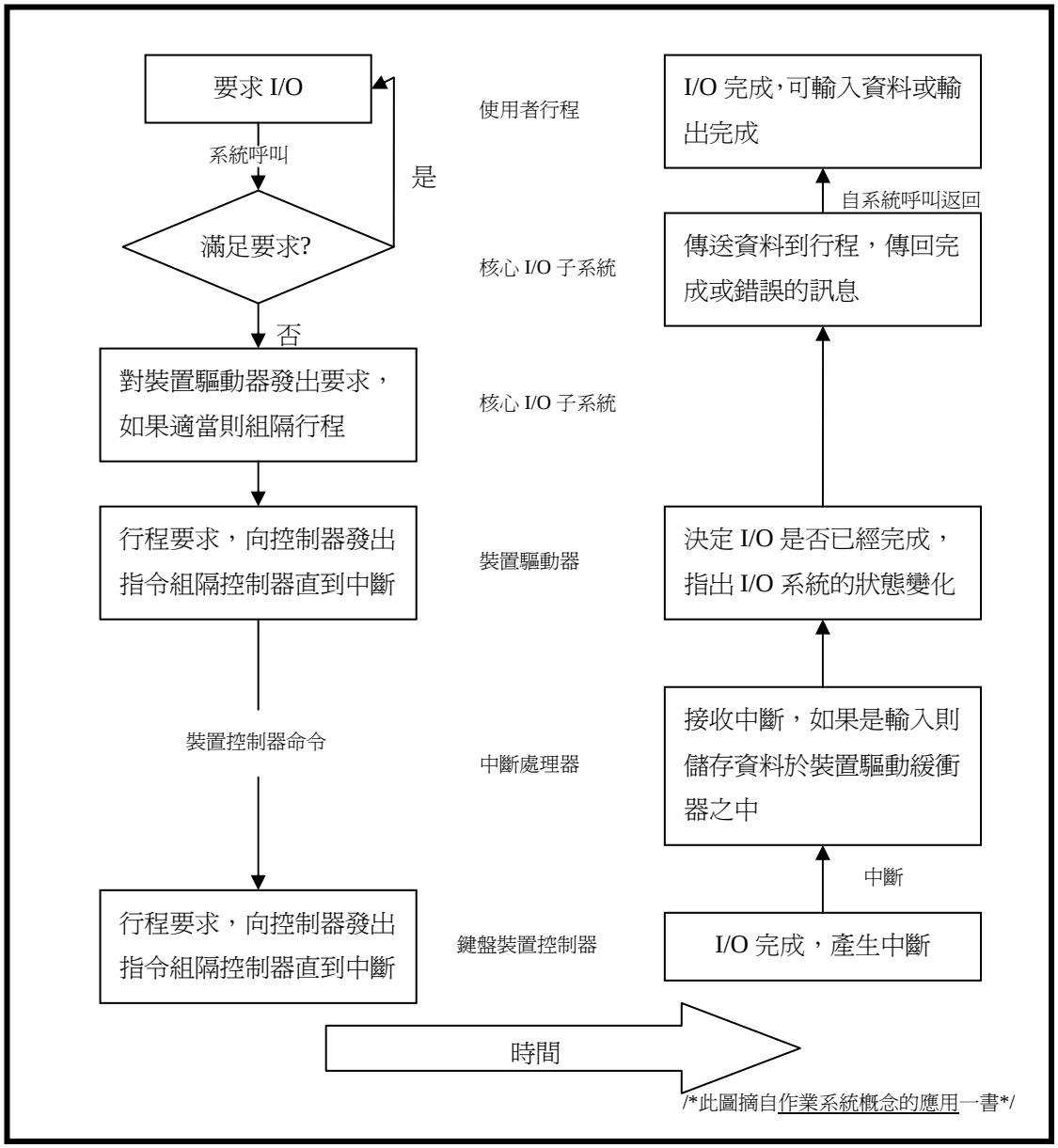
如果裝置名稱空間與一般檔案系統名稱空間一起使用，則一般自動提供檔案系統名稱服務；如果檔案系統對所有檔案名稱都提供擁有者與存取控制能力，則裝置即可具備擁有者與存取控制功能，因為檔案可儲存於裝置內，因此這種界面會提供兩層 I/O 系統的存取。名稱可用來存取裝置本身，或用來存取儲存於裝置內的檔案。

UNIX 一般以檔案系統名稱空間表示裝置名稱，不像 MS_DOS 檔案名稱使用冒號隔離子，UNIX 之路徑名稱在裝置部份並無清楚分隔；事實上，路徑名稱之中並無裝置名稱的部份，UNIX 擁有一個鑲嵌表格(mount table)，可對前段路徑名稱與特定裝置名稱加以關聯；為了解決路徑名稱的問題，UNIX 會搜尋鑲嵌表格中的名稱，找到最長之相符字集，於是鑲嵌表格中的映對欄位即代表裝置名稱。此裝置名稱的命名方式與檔案系統名稱空間相同，當 UNIX 搜尋檔案系統目錄結構中的名稱而非找尋 inode 數字時，UNIX 會發現一組〈主要，次要〉的裝置號碼，主要裝置號碼指出可處理 I/O 的裝置驅動器；次要裝置號碼則將裝置驅動器做為索引值傳入裝置表之中；相關裝置表項提供連接埠位址或裝置控制器的記憶體映對位址。

現代作業系統在搜尋表格時，在要求與底層裝置控制器之路徑兩者間提供多種狀態相當具有彈性，在應用程式與驅動器之間傳遞要求的方法也極普遍。因此，我們將不需重新編譯核心程式碼，即可在電腦之中加入新的裝置與驅動器。而且，某些作業系統具有因應需要隨時載入裝置驅動程式的能力。在開機時，系統首先檢查硬體匯流排決定目前存在哪些裝置，接下來系統再載入必須的驅動器，至於方法則不管是立即載入，或是在 I/O 要求發出之後再執行都可以。

UNIX 系統 V 有一個有趣的機制，稱為資料串列(stream)，讓應用程式能夠動態組合驅動器程式碼之管線(pipeline)，所謂的資料串列為一個在裝置驅動器與使用者層處理程序之間的全工(full-duplex)連接，它包含一個可以作為使用者處理程序介面的資料串列標頭(stream head)、一個可控制裝置的驅動器尾端(driver end)，以及零或多個存在它們之間的資料串列模組(Stream modules)；模組可以被加到資料串列之中以增加分層特點(layered fashion)方面的功能。例如，處理程序能夠透過資料串列開啟序列埠裝置，而且可以在模組上執行輸入編輯工作。資料串列可以用在內部程序與網路通訊方面，事實上，在系統 V 之中，承接口機制即由資料串列製作完成，如下圖所示。

執行 I/O 操作需要相當多的步驟如下所示，整個過程需花費極多 CPU 時間：



/*此圖摘自作業系統概念的應用一書*/

1. 處理程序發出阻隔讀取系統呼叫給先前已被開啟之檔案描述子。
2. 核心的系統呼叫程式碼檢查參數之正確性，在輸入的時候，若資料已存於緩衝快取器之中，則直接將資料回傳給處理程序，I/O 要求完成。
3. 若資料不存於緩衝快取器中，需要執行底層 I/O，如此行程才會自執行佇列移至裝置等待佇列並且安排 I/O 要求。最後，I/O 子系統會將要求送給裝置驅動器。根據不同的作業系統，要求會藉由子常式(sub routine)呼叫或核心訊息傳送。
4. 裝置驅動器會配置核心緩衝空間以便接收資料，並安排 I/O。最後，驅動器透過寫入裝置控制暫存器的方法，將命令送至裝置控制器。
5. 裝置控制器操作裝置硬體執行資料傳送。
6. 驅動器可能會查詢狀態與資料，或者可能會建立 DMA 傳輸到核心記憶體之中，當傳輸完成時產生中斷。
7. 正確的中斷處理器透過中斷向量表接收中斷、儲存任何必要資料、發訊號給裝置驅動器，並自中斷回傳。

5.5 I/O 與效能

I/O 是影響系統效率的主要因素，在執行裝置驅動器程式碼以及當行程被阻隔(block)與非阻隔(unblock)時，公平有效地安排其執行順序將會為 CPU 帶來極大負擔，導致內容切換在 CPU 以及其硬體快取之間；I/O 在核心中斷處理機制方面較無效率，而且當在控制器與底層記憶體之間進行資料複製時，會降低記憶體匯流排的速度，這個情況在核心緩衝器與應用程式資料空間兩處進行資料複製時也會發生。如何妥當處理這些問題，成為電腦架構的主要考量之一。

雖然現代電腦每秒能夠處理數百個中斷，中斷處理執行起來卻相當麻煩，每個中斷會使系統進行狀態改變，執行中斷處理器，以及儲存狀態。如果花費在忙碌等待上的 CPU 時間不會太多，程式化 I/O 與中斷驅動 I/O 相較之下來得有效率得多。在 I/O 完成時，基本上會繼續行程的執行，此時的內容切換則為主要的額外負擔。

網路上的交通量也可能造成較高的內容切換率(context-switch rate)，試想，當自某機器登入至另一部機器時的情況，每個在本地機器上鍵入的字元必須被傳至遠端機器；亦即在本地機器上鍵入字元，產生鍵盤中斷，而字元就透過中斷處理器依序傳給裝置驅動器、核心、最後是使用者行程。使用者行程發出網路 I/O 系統呼叫，將字元送給遠端機器，接下來字元會傳入本地核心，透過網路層建立網路封包，再進入網路裝置驅動器。網路裝置驅動器將封包傳給網路控制器，送出字元並產生中斷，中斷再傳回核心，引發網路 I/O 系統呼叫完成指定工作。

現在，遠端系統網路硬體已經收到封包並產生中斷；網路協定解開字元封包，並送達適當之網路 daemon，網路 daemon 決定到底要加入哪一個遠端登入對

話層，並將封包傳給所屬對話層之適當子 deamon（sub daemon)。經過內容交換與狀態交換之後，通常接收端會將字元回傳給發送端，重複相同步驟。

某些系統在終端機 I/O 方面，使用獨立前端處理器(front end processors)以減少主 CPU 上的中斷負擔。例如，終端集中器(terminal concentrator)可將來自數百部遠端終端機的交通量，轉至大型電腦的某一個連接埠處理；I/O 通道(I/O channel)為在於主電腦以及其他高層系統之中固定而且具特殊用途的 CPU，通道的工作是為了減少來自主 CPU 的 I/O 工作，方法是當主 CPU 能夠處理資料時，通道中的資料能夠順暢流動；和在小型電腦之中的裝置控制器與 DMA 控制器比較起來，通道可以處理較一般性與較複雜的問題，因此通道可以依特殊工作需要而調整。

我們可以利用下列數種方法改對 I/O 之執行效率：

- 減少內容切換次數。
- 減少在裝置與應用程式之間傳輸時，必須複製的記憶體資料次數。
- 減少使用大型傳輸、智慧型控制器與查詢方法之中斷發生頻率。
- 使用 DMA 控制器的並行處理，或者可以減少來自 CPU 簡單資料複製次數的通道。
- 將處理初始條件移至硬體之中，使它在裝置控制器之中的操作可以與 CPU 以及匯流排的操作同時進行。
- 平衡 CPU、記憶體子系統、匯流排與 I/O 之執行效率，因為任一區域的額外負擔都可能造成其他系統的不便。

裝置的複雜度變化極大，例如，滑鼠就很簡單，滑鼠移動與按鈕敲擊皆會被轉換為數字資料，這些數字傳自硬體，再到滑鼠裝置驅動器，最後是應用程式。相反地，NT 磁碟裝置驅動器提供的功能就很複雜，它不只是負責管理個別磁碟機，還要製作 RAID 陣列。為了達到這個目的，它將應用程式的讀取或寫入要求，轉換成已經協調完成之磁碟 I/O 操作組，而且它還製作了複雜錯誤處理與資料回復演算法，同時，因為輔助儲存(second-storage)執行效率對整個系統執行效率相當重要，因此必須試著最佳化磁碟的執行效率。

第6章 安全保護機制

在作業系統中各個不同的行程必須有保護措施的隔開，以免互相影響。因此，為了有良好的保護作用，檔案、記憶體、CPU 以及其他的資源，都只能在行程獲得特定的權利下，才准許被取用。

保護涉及到控制程式、行程、或使用者對電腦系統所定義之資源的存取機能。此機能必須提供加入控制，以及某些強制性方法的規定。我們將保護和保密加以分別，保密是系統完整以及資料保存的信賴度測量。而保密保證是比保護更廣泛的課題。

6.1 保護的目的與範圍

由於電腦系統變得更複雜，而且在應用上更普遍後，保護其完整性的需求必須提高。原先此保護功能是被視為“多元程式"(multiprogramming)作業系統所具備的，使得它能夠使一些不可靠的使用者，能安全地共用相同邏輯命名空間，如檔案目綠或共用相同實體命名空間。目前保護觀念已擴充到各種複雜的系統中共享各種資源的可靠性問題。

設立保護的動機很多，最明顯的就是防止被使用者惡意地去破壞系統上的存取限制。再進一步來說，是為了確保系統內每一個程式都能在指定的狀況下使用這些資源。這對一個可信度高的系統來說，是絕對需要的。

具有保護能力的系統能夠藉由各部份子系統連接介面，偵測出潛伏錯誤，以提高可信度。如能提早偵測出介面錯誤，就可以防止某故障子系統影響其他正常的系統。然而沒有保護措施的資源就沒有辦法防止無權限和無資格的使用者使用。因此一套具有保護措施的系統就必須具備區分有權限和無權限的能力。

保護功能所扮演的角色是提供強迫執行管理資源使用策略的方法。這些策略可由不同的方式製定。有些是固定在系統的設計中，有些是藉著系統的管理表示出來。更有些是由各個使用者自己去設定自己檔案或程式的保護工作。一個良好的保護系統，必須能針對不同的策略，做某些彈性的運用與管理。

使用資源的策略是依使用狀況而常常在改變的。因此，保護的設計不能只站在作業系統設計者的立場來考慮。相反的，必須使它也能成為一般使用者的工具，這樣，一些由附加系統的應用程式或資源，才不至於被濫用。

電腦系統是一些行程與物件的組合。所謂物件指得是硬體物件(如 CPU、記憶體區段、印表機、磁碟機和磁帶機等)和軟體物件(如檔案、程式和號誌等)二者。每一物件都有唯一的名字，以示區別。而且只能由一些特定的操作來取用。

這些物件各有不同的運作方式，比方說，CPU 只能用來執行，而記憶體則可

以用來讀和寫，然而讀卡機只能讀，磁帶機則可讀、寫和返轉，資料檔可以建立、開啟、讀寫、關閉或刪除，而程式檔則只能讀、寫、執行和刪除。

很明顯地，一個行程只能存取已經獲得存取權的資源，而且任何時候，它都只需取得完成目前任務所需要的最少資源。這個要求也就是我們所謂”必須知道的原則(need-to-know principle)”，它能減低一個錯誤行程可能損害系統之程序。比方說，當一個行程 p 引發一個程序 A 開始執行時，這程序能夠存取的，除了自身的變數之外，就只是多了行程 p 所傳遞給它的參數，行程 p 的變數都是不能 任意存取的。同樣的，當行程 p 要求編譯程式來編譯某一個程式時，編譯程式也只能針對該特定的程式檔案來處理(如原始程式檔、程式報表檔等)。反過來說，編譯程式在編譯的過程當中為某理由或使其完美為目的也有自己的變數，而這些東西，p 是不能取用到的。

為了瞭解不同的保護策略，現在引入一種保護區(protection domain)的觀念。每一個行程都必須在設定的保護區中運作，保護區規定了每一行程可以存取的資源。保護區中設定了“物件"與在這些物體上的運作功能。遣些功能稱之為存取權(access right)。因此保護區也可以說是這些存取權的組合，每一個存取權都是用底下型態的資料序對來表示:〈物件名稱,權利集合〉(即〈object-name, right-set〉)。例如，如果在定義域(domain)D 有存取權---〈file F，{read, wright}〉,則表示行種在定義域 D 中，能對檔案 F 做讀出與寫入的工作，然而它不能對此物件執行其他操作。

定義域與定義域之間不一定要不相交，存取權也可以共用。行程與定義域之間的關係可為靜態(如果在行程的生命期間內，可用的資源集合保持固定)或動態。可以預期的，源自於建立動態保護定義域的問題，比簡單的靜態間題更需要小心的解決。

如果行程與定義域的關係是固定的，並且我們想遵守”必須知道"的原則，則必須有改變定義域之值的方法，一個行程的執行可能分兩個階段。例如它可能在一個階段需要做讀，在另一階段做寫。如果定義域是靜態的，我們必須同時定義此定意域為可讀可寫，然而這種安排在兩階段中都提供比需要還多的權利，因為在我們只需寫的存取權利階段中，我們也有讀的權利，反之亦成立。因此違背了必需知道的原則。我們必須允許定義域的內容可更改，這樣子它總是反應出最少存取權利。

如果關係是動態的,一個方法需適用於允許行程從一個定義域轉換到另一個定義域。我們並且允許定義域的內容改變。如果我們不能改變定義域的內容，當我們希望改變定義域內容時，我們可產生一個新改變過的定義域，並且轉換到新產生改變過的定義域。定義域可以用以下的方式辨識：

- 每一個使用者可能是一個定義域。 在這種情況下，可以存取的物件集合是依據使用者的識別。定義域的轉換在使用者改變時發生—通常在一個使用者簽退而另一個使用者簽到時。
- 每一個行程可能是一個定義域。 在這種情況下，可以存取的物件集合

是依據行程的識別。定義域的轉換相當於一個行程送出訊息給另一行程，然後等待反應。

- 每一個程序可能是一個定義域。 在這種情況下，可以存取的物件集合相當於定義在程序中的區域變數。定義域的轉換在程序呼叫時發生。

6.2 存取矩陣(Access Matrix)

我們的保護模式在觀念上可以把它看成是一個矩陣，稱為存取矩陣。其中列表示定義域，而行則代表處理物件。矩陣內每一個元素，都包含有一些存取權。由於處理物件的名稱已經表示在每一行上，因此在其存取權內，物件的名稱便可以省略不記。對於陣列元素（ i , j ）所定義的，便是一個行程在定義域 Di 內，對處理物件 Oj 所能執行的運作項目。

為了說明這觀念，如下表，有四個定義域及四個物件：三個檔案(F1,F2,F3)和一個印表機。當行程在 Di 中執行時，它可讀 F1 與 F3。若在 D4 中，除了上述具有 D1 內的相同特權外，同時對檔案 F1 和 F3，也能有寫入的權利。

物件 / 定義域	F1	F2	F3	印表機
D1	讀		讀	
D2				列印
D3		讀	執行	
D4	讀 寫		讀 寫	

存取矩陣的技術提供我們說明多樣性策略的方法，此方法包含製作存取矩陣和保證與語意有關的性質確實成立。我們必須保證一個行程在定義城 Di 中執行只能夠存取在 i 列中標明的物件，就像在存取矩陣中許可一樣。保護的策略決定能用存取矩陣之方式來製作，策略之決定牽涉那些權利應該包括在第(i , j)記綠中。我們也必須決定每個行程執行的定義域，最後的策略通常是由作業系統來決定。

通常是由使用者決定存取矩陣記錄之內容。當一個使用者產生一個新物件 Oj，行 Oj 便加入存取矩陣，並由建立者產生適當的記錄。如果需要，使用者可以決定在行 j 中的某些些記錄加入某些權利或在其他記錄中加入其他權利。

存取陣列提供一適當機能以定義和製作行程及定義域之間，靜態與動態相關的嚴格控制。當我們的行程從一定義域轉換到另一個定義域時，我們正在一個物件(定義域)上執行一個動作(轉換)。我們可以包括在存取矩陣物件中之定義域，來控制定義域的轉換。另外我們可以藉著把存取矩陣本身視為一物件，來控制這

些改變。事實上，因為在存取矩陣中的每一記綠可以個別的做修改，我們必須考慮將存取矩陣中每一記錄視為一個物件來保護。

行程應該能夠從一個定義域轉換到另一個定義域。定義域 Di 轉換到 Dj 只有存取權利轉換存取(i , j)才許可。下表表現了一個在定義域 D2 中執行的行程能夠轉換到定義域 D3 或到定義域 D4。一個在定義域 D4 的行程能夠轉換到 D1 及在定義域 D1 之行程能轉換到定義域 D2。

物件 / 定義域	F1	F2	F3	印表機	D1	D2	D3	D4
D1	讀		讀			轉換		
D2				印			轉換	轉換
D3		讀	執行					
D4	讀寫		讀寫		轉換			

允許對存取矩陣記錄內容控制性的改蠻，需要三個外加的動作：拷貝(copy)、所有人(owner)及控制(control)。能夠拷貝一個存取權利從一個定義域(列)到另一個定義域的能力，以附帶一個星號(*)來表示。拷貝的權利只允許同行(也就是對物件)已定義的權利之間做拷貝工作。有兩個此種形式的變體：

1. 當一個權利從 access(i, j)拷貝到 access(k, j)如果接著它從 access(i, j)中移去，這是一個權利的傳送而非拷貝。
2. 拷貝權利的傳播受到限制，也就是說當權利 R 從 access(i, j)拷貝到 access(k, j)時，只有權利 R(而非 R*)產生。在定義域 Dk 中執行的行程不能再拷貝權利 R。

一個系統可以只選擇這三個拷貝權利中的一個，或它可提供所有，標明它們為不同的權利：拷貝、傳送、及限制拷貝。

拷貝權利允許一個行程，在同一行中拷貝某些權利從一個記綠到另一個記錄。我們並且需要一個允許加入新權利及移去權利的方法。所有者的權利控制這些動作。如果 access(i, j)包括所有者權利，則一個在定義域 Di 中執行的行程能加入及移去在行 j 中任何記錄的權利。例如，在下表中，定義域 D2 是 F2 與 F3 的所有者，因此可加入或移去這兩行中任何有效的權利。

物件 / 定義域	F1	F2	F3
D1	所有者 執行		寫

D2		所有者 讀*	讀* 所有者 寫*
D3		寫	寫

拷貝與所有者權利允許一個行程改變一行中的記錄。並且需要一個方法改變一列中的記錄。控制權利只適用於定義域物體。如果 access(i, j)包含了控制權利，則一個在定義域 Di 中的行程將會從第 j 列中移去任何一個存取權利。

雖然拷貝與所有者權利提供我們限制存取權利傳播的機能，但是並不提供我們.防止資訊傳播的適當工具。確保在一個物件內原先擁有的資訊不會移到執行環境外的問題叫做監禁問題(confinement problem)。這個問題通常是無解的。

這些在定義域及存取矩陣上的動作，對他們本身而言並不特別重要，而更重要的是它們顯示了存取矩陣允許履行及控制動態保護需求的能力。新的物件與新的定義域可以機動性的產生，包括於存取矩陣模式中。然而我們只顯示了基礎的方法，關於那些定義域對於那些物件有那些存取的方式的策略決定，必須由系統設計者與使用者來做。

6.3 語言基礎之保護系統(Language-Based Protection)

在現有的電腦系統所提供的保護程式，通常經由作業系統核心來完成，它就像一個安全的管理人員，監督和確認每一個存取保護資源的企圖。因為廣泛的存取確認是一個基本上相當負擔的來源，它必須提供硬體支援來減低每一個確認的花費，否則系統的設計必須傾向於折衷保護的目標。

當作業系統變得複雜，及特別當提供更高階的使用者介面時，而使得保護的目的變得更嚴密。保護系統設計者費了很大的工夫在產生程式語言的構想上，特別在抽象資料類型的觀念上。保護系統現在所考慮的不只是存取資源上的識別，並且在存取的功能本質上。在最新的保護系統中，考慮功能述諸於在一個系統定義功能的集合上擴充，例如標準檔案存取方法，包含了使用者所定義的功能。

資源使用的原則依應用也不同，可能會依時間而改變。所以保護不再單獨的被認為是系統設計者的工作。它應也適用於當做應用設計者的工具，所以應用子系統的資源可以視為避免一個錯誤的影響。

程式語言也用來設計保護系統。規定在系統中對共同資源存取需要的控制，是一個關於資源宣告的敘述。這種陳述經由對它的類型設備擴展，能夠集合成為一個語言。當保護隨伴著資料類型被宣告，每個子系統的設計者可以標示出它的保護需求以及它需使用那些其他系統中的資源。這樣的標示應該直接地像程式一樣組合，並且在程式敘述中用該種語言表示。用這種方法有下述數個好處

1. 只需要做保護簡單的宣稱，而不是以一串對作業系統處理程序的呼叫寫出。

2. 保護需求的敘述可以與特別的作業系統所提供的設備無關。
3. 實行的方法不必由子系統的設計者提供。
4. 一個宣稱的標示很自然，因為存取特權與資料類型的語言觀念相關。

程式語言的製作，提供了不同的技術來實行保護，但任何一種，為了安全都必須依照某階段來自基礎機器的支援及它的作業系統，例如，假設一種語言被用來產生在 CAP 系統上可執行的碼。在這系統中，由基礎硬體所做的任何儲存記錄參考，都間接的經由 CAP 所提供的軟體資格。這限制禁止任何行程在任何時間去存取它的保護環境以外的資源。然而，一個程式隱含了在執行任何行程的特別程式碼分段，一個資源可能如何的被使用。這些限制可以很快的經由 CAP 所提供的軟體資格而製作。

一個語言的製作方式可能提供了標準、保護的處理程序來解釋軟體資格，它會了解語言中所標明的保護策略。這方式將策略標明交由程式者處理，而不必製作實施的細節。

一個系統並不一定提供那麼強而有力的保護核心，但仍然有適用於履行在程式語言中保護標明的方法。最主要的不同是保護的安全性不會和保護核心所支援的一樣大，因為方法必須依靠更多關於系統製作敘述的假設。一個編譯器能分別尋找那些保證沒有違反保護會發生的地方，並不同地對待它們。

那麼，只由核心製作，和主要由編譯器提供的方式，有什麼相對的價值呢？

- 保密性(security)：經由核心製作者比由編譯器產生的保護檢查碼，對保護系統本身，提供了更高度的保密性。在由編譯器支援的形式中，安全度建立在轉換器的正確性，在某些儲存空問管理的基礎方法，它保護分段編輯後的碼被執行，且根於檔案的安全性自己存入程式。一些相同的構想也應用於軟體提供的保護，因為核心可能存在於固定的實體儲存分段，並可能存入一個指定的檔案。在一個附加資格系統中，所有的位址計算都是由硬體或由固定的微程式來做，所以可能更安全。硬體提供的保護對可能是硬體或是系統軟體發生錯誤所引起之保護的違背，有相當的免疫力。
- 彈性(flexibility)：在履行一個使用者定義的策略上，有一些對保護核心彈性的限制，不過它可以供應適當的設施給系統，來強行它自己的策略。利用程式語言，可以依履行的需要來宣稱及強行保護政策。如果一個語言沒有提供足夠的彈性來做擴充或置換，則系統服務的不安定性比起修改作業系統核心而引起的不安定性反而較少。
- 效率性(efficiency)：最好的效率性是當所有的保護強行都是用硬體來支援。在軟體支援所需求範圍內，以語言為基礎的強行固定存取可以在編譯時線外(off-line)被修改。而且，強行方法能夠經由智慧型的編譯器剪裁而達到需求，固定的核心呼叫負擔通常也能夠被避免。

總而言之在程式語言中，必須標明保護允許分配策略的高階層描述與資源的使用。當自動的硬體支援檢視不能用時，一個語言的製作方式能夠提供保護強行的軟體。另外，還能解釋保護標明而產生對硬體及作業系統之保護系統的呼叫。

軟體資格(software capability)可以視為一個物件的計算來使用。此觀念的本質為某個程式可能會產生或檢查這些軟體資格特權構想。一個資格產生 g)程式能夠執行一個指令動作——對資料結構“緘封”(seal)，阻止它的內容被任何沒有擁有緘封或“開啟”(unseal)特權的程式存取。它們可以拷貝它或傳遞它的位址給其他的程式元件，但不能得到對內容的存取權。此觀念中的唯一問題是使用緘封與開啟動作在標明保護上採取一個程序式的方法。一個非程序或宣稱的形式對應用程式設計者而言，是一個較可用的保護描述形式。

一些新語言的結構方式，讓程式人員對特定的管理資源之使用宣告有不同的限制。這些結構提供了三種功能：

1. 將資格安全且有效率地分散到客戶的行程：特別是，這些機能可以確保只有在使用者行程被授權對管理資源的資格後，才可以使用這些資源。
2. 在標明一個可能要求分配資源(例如：一個檔案的讀取行程應該允許讀其檔案，然而一個寫入行程應該能夠讀與寫)的特殊行程裏，標明動作上的限制。它不應該將相同集合的權利給每一個使用者的行程，並且一個行程不能擴大它的權利集合，除非有存取控制方法的認可。
3. 在一個特殊的行程中，要求對資源做不同動作，次序上要有所限制(例如：一個檔案在它被讀之前必須已開啟)。

將保護觀念併入程式語言，當作一個實際系統設計的工具，目前尚在發芽階段。對有著分散架構及日趨迫切的資料安全性的系統設計者而言，保護將可能變成更為重要的考慮事項。語言為基礎的保護可以用 Java 的 JVM 做最佳的範例。因為 JVM 的保護特性是眾所皆知。

6.4 保密與認證

保護問題嚴格說是內在(internal)的問題：在一個電腦系統中我們如何提供對程式與資料的控制存取？另一方面，保密性(security)不僅要求一個妥當的保護系統，並且考慮系統操作的外在環境，如果操作者的操作台被不合法的人員看到，或如果檔案能很輕易的從電腦系統中移去，並裝置到另一個沒有保護的系統，則內在保護就沒有用了。這些保密性的問題主要是管理的問題，而非作業系統本身的問題。儲存在系統的資訊(包括資料和程式碼)，以及電腦系統的硬體資源都需要防止被非授權地使用，惡意的破壞或修改，和意外地加入不一致性。

作業系統能夠提供讓使用者來保護他們的資源的功能。只要系統的使用者不要試著克服刻意的使用和存取這些資源，則這些功能就可以運作的很好。當情況不是如此時，保密性就開始作用。如果一個系統的資源在所有情況下都是有意識

的使用和存取，則我們稱此系統為安全。但是達到完全地安全通常是不太可能的。無論如何，必須有某些機能以使保密性偶爾有些小缺口，而非經常狀況。

違反系統保密性可以分成敵意(惡意)或意外。保護防止意外的誤用比防止惡意的誤用容易。以下是一些惡意存取的型式：

- 非授權的讀取贅料(竊取資訊)
- 非授權的修改資料
- 非授權的破壞資料

完全地保護系統避免惡意的濫用是不可能，但可以讓侵犯者付出相當高的代價以妨礙大多數沒有正當授權者試圖存放在系統內的資訊。

為了保護系統，我們必須以兩種層次來做保密性的衡量：

- 硬體：包含電腦系統的一個站或數個站必須在硬體上堅固以防止武裝的或不正當的侵入。
- 人：為了降低使用者將存取權給入侵者(例如為了賄賂)的機會，使用者必須小心地審查。

如果必須確保作業系統的安全，道兩個層次的保密都必須維持。高層保密(硬體或人)的弱點會侵犯到低層(作業系統)的保密測量。

在許多應用上，對電腦系統的保密性花費可觀的努力是值得的。包含薪資或其他財政資料的大型簡用電腦會成為竊賊的目標。包含有關於公司操作資料的系統可能會引起無恥競爭者的興趣。進一步來說，無論是意外或賄賂而損失遺些資料都可能嚴重地傷害公司運作的能力。

作業系統的主要保密問題是認證(authentication)問題。這個能力事實上落在我們正確地驗證每一個系統使用者的力量上。一個使用者必須正常地證明自己。我們如何決定使用者的身份是否被認可？一般而言，認證是基於三項集合的組成：使用者擁有物(如鑰匙或卡片)、使用者知識(如使用識別字或密碼)及使用者屬性(如指紋、感光方式或簽名)。

6.4.1 密碼概述(passwords)

最通常的方法去認證使用者身份是使用者的密碼。當使用者要證明自己，他必須給一密碼。如果使用者提供的密碼與存在系統內的密碼相符，則假設他是個合法的使用者。

在沒有更完備的保護方法下，密碼通常用來保護電腦系統中的物件，它們可被視為一種特別的鎖匙或資格。例如，每一資源都與一個密碼有關。每當使用者要求使用資源，必須提出密碼，如果密碼相符則存取被認可。不同的密碼可能與

不同的存取權利有關。例如:不同的密碼可能用作讀取、增加和檔案更新。

雖然使用密碼有些問題存在，但因易於瞭解與使用，故仍極為普遍。密碼的問題是相當難保持密碼的保密性。密碼會由於被猜出、意外的暴露或非法地從一個授權者轉移給一個非授權者而被破壞。

有兩種常見的猜測密碼方法。一種是入侵者(人或程式)知道使用者或是擁有關於使用者的資料。遺種情況經常發生，一般人常使用明顯的資料(例如，他們的生日或兒女的名字)做為密碼。另一種方法是嘗試由文字、數字、和標點符號組成的所有組合，直到找到密碼為止。簡短的密碼無法留有足夠的選擇以防止密碼被重複嘗試的猜出。例如：一個四位的密碼，只提供了 10,000 種不同的變化。平均只需 5,000 次嘗試即可猜出。如果寫一個程式來做嘗試密碼的工作，每一次嚐試需百萬分之一秒，則它將只需花費大約 5 秒即可猜出密碼。較長的密碼則較不易由計算猜中，而將所有標點符號字元用在密碼中也會讓猜密碼的工作變得比較困難。當然使用者必須利用較大的密碼空間和不能只用小寫字母。

因為暴露造成密碼保密的失效可能是由於視覺或電子的監測造成。侵入者可能在使用者簽到時從使用者背後偷看，然後輕易地從鍵盤上看到密碼。另外，任何可以使用連在網路上之電腦的人都可以不知不覺的加入一台網路監視器，這可以讓他發覺到所有在網路上傳遞的實料，包括了使用者識別碼和密碼。如果密碼寫在能夠讀出或遺失的地方，則暴露就是一個特別嚴重的問題。有些系統強迫使用者選擇難記或較長的密碼，但這可能造成使用者記錄密碼，造成比系統允許簡易密碼時更沒有保密。

最後一種密碼被傷害的方法是由於人性所造成。大部份電腦裝置都有不允許使用者共享帳號的規則。這個規則有時候是為了記帳的原因，但通常是為了幫助保密性。例如，如果一個使用者帳號被數個使用者共用，而且從該使用者帳號產生保密的漏洞，那就不可能知道誰在當時使用該帳號。若是每一個使用者一個使用帳號，則可以直接詢問該使用者關於該帳號的使用。有時候，使用者破壞帳號分享的規則以幫助朋友或迴避記帳，而這種行為可能造成系統被非授權的使用者使用。

密碼可由系統庫生或由使用者選擇。然而系統產生密碼可能難以記憶，也可能寫下相同的密碼。然而使用者選擇的密碼通常容易被猜出(例如：使用者的帳號或名字)。有些裝置有管理員，他有時檢查使用者的密碼，如果太短而易被猜出，則會提醒使用者。一些系統也有老舊密碼，強迫使用者在固定期間內要去改變，此方法不是很簡單，因為使用者可能會把前後兩個密碼串在一起。在一些已經完成的系統之中，解決這個問題的方式是對使用者的密碼經歷作記錄。例如，該系統可以記錄最後使用過的幾個密碼，並且不准他們再使用。

幾種不同的簡單的密碼方法可被使用。系統可以限制密碼不能是字典內的單字。在最極端的情況，在每一活動期間都要改變密碼。在活動的最後期間，必須選擇新的密碼(由系統或由使用者來選擇)，給下次活動期間使用。

6.4.2 編碼密碼(Encrypted Passwords)

一個有關所有這些方法的共同問題是很難對密碼保密。UNIX系統使用編碼密碼以避免保持它的密碼機密。每個使用者都會有一個密碼。系統包含有一個非常困難逆轉的函數，但卻易於計算。也就是說，給一個 x 值，則很容易去算出 f(x)的值。然而給一函數值 f(x)，則不可能算出 x。這函數用來對所有的密碼做編碼。只有編碼後的密碼被存起來。當使用者顯示一個密碼，它將被編碼並且與先前存起的編碼密碼做比較。縱然先前存起的密碼被看到，也無法去解碼而決定出密碼。如此密碼檔就不必保密了。這些函數 f(x)是典型的編碼演算法，它們已經經過嚴謹地設計和測試。

這種方法的缺點是系統不再對密碼有控制權。雖然密碼被編碼，但具有密碼檔拷貝的任何一個人都可以執行快速的編碼程式以找出密碼，例如把字典中的每一個字編碼，然後比較密碼檔是否符合。如果使用者所選的密碼也是字典中的一個字，則密碼就會被破解。

由於 UNIX系統使用一種大家都知道的編碼演算法，駭客可能保有一個密碼索引對，以便快速地找到密碼，而這個密碼就是先前所破解的。為了這個原因，新的UNIX版本將密碼元素隱藏起來。

在 UNIX系統中密碼方法的另一個弱點，是許多 UNIX系統只處理最前面的前八個字元。因此，對使用者而言，如何利用密碼空間是非常的重要。為了避免字典式的編碼方法，有些系統不允許將字典之中的字當作密碼。有一種好的技術是藉由一個容易記住的慣用語之每一個字的第一個字元來產生你的密碼。或將較大和較小的兩個字元與一個數字或標點加入，也可以增添好的處理。

6.4.3 單次密碼(One-time passwords)

為了避免密碼有被發覺和背後窺視的困擾，系統可以使用一個配對密碼(paired passwords)的集合。在一個對話層開始的時候，系統隨機地選擇並且顯示密碼對的一部份，使用者必須供給另一部份。在這個系統之中，使用者接受盤問(challenged)，並且須回應(respond)出盤問的正確答案。

這種方法可以使用像密碼一樣普及的演算法。如整數函數。系統選擇一個隨機的整數並且顯示給使用者，而使用者應用遣個函數並且回應正確的結果。系統也使用這個函數，假如兩個結果吻合，則允許存取。

另外有一個不容易暴露密碼的方法。例如，使用者可能鍵入一個密碼，任何一個實體攔截這個密碼，並且嚐試再次使用它時，將會失敗。在這個變化之中，系統和使用者共用一個機密。這個機密不會在媒體之中傳送，因此不會暴露。說得更恰當一點，這個機密和種子(seed)一起輸入到一個函數。種子是一個隨機的數字或字元串列，使用在函數 f(機密，種子)的輸入，這個函數的結果再如同密

碼一般地傳途到電腦。下一次，使用者需要被認證時，另一個種子會產生，接著發生相同的步驟。但這一次的密碼是不相同的。因為每一次的密碼都不相同，任何人從一次對話層攔截的密碼後，下一次使用時都將失敗。

單次密碼的其他變化如使用編碼簿(codebook)或是單次襯墊(one-time pad)，這是一個單獨使用密碼的串列。在這個方法之中，串列之中的每一個密碼只使用一次，然後將它註消或刪除。

6.5 程式威脅(Program Threats)

在一個程式由某一個人撰寫，而由另一個使用者使用的環境，就有機會產生誤用，這種情況可能產生不可預期的行為。

6.5.1 木馬程式(Trojan Horse)

許多系統有允許使用者所寫程式被其他使用者使用的方法。如果這些程式在提供正在執行的使用者存取權利的定義域中，它們可能誤用這些權利。例如在一個內在的文字編輯程式中，可能要去被編輯的程式中尋找某個關鍵字，如果有任何發現，則整個檔案可能拷貝到一個可被文字編輯器的產生器存取的特別區域。一個程式碼分段誤用它的環境即稱為木馬(Trojan horse)。冗長搜尋路徑造成木馬問題惡化。搜尋路徑列出當給予混淆檔名時要搜尋目錄的集合。此路徑將被搜尋以找出該名稱的檔案，並且執行該檔案。在搜尋路徑中的所有目錄必須保密，否則木馬可能潛伏到使用者路徑中，並且意外地執行。

木馬的一個變化是一個模擬的登入程式。一位不注意的使用者開始在終端機登入時，他注意到他似乎輸入錯誤。於是他再試一次並且成功。到底發生什麼事呢?他的密碼和認證鍵已經被登入模擬程式所竊取，這個模擬程式是小偷留下來在終端機上執行。模擬程式將密碼儲存起來，列印出登入錯誤訊息並且離開，然後使用者才碰到一個真正的登入。這種型式的侵入可以藉由作業系統在交談對話層結束時，列印出一個有用的訊息。

6.5.2 陷阱(Trap)

程式或系統設計者可能會在軟體內留下一個僅供自己使用之陷阱。例如，該程式碼可能檢查某一特定使用者或密碼以迴避正常的安全檢查程序。曾經有程式人員藉著在他們的程式碼中加入錯誤的四捨五入，讓偶爾出現的半分錢進入到他們的帳號。這個帳號的金額可能累加到一筆很大的金額。

聰明的陷阱會將它放在編譯器之內。該編譯器可產生標準的目的碼和陷阱，而不論其原程式為何，就算檢查原始程式亦無法發現任何安全性問題。只有在編譯器中才包含該項資訊。陷阱暴露了一項很棘手的問題，因為我們很可能需要分析一個系統所有的原始程式碼。若該軟體系統由數百萬行程式所組成，很少會去

做這分析。

6.6 系統威脅(System Threats)

大部份作業系統均提供一項方法供行程去產生其他行程。在這種環境下，有可能產生作業系統資源和使用者檔案被誤用的情況。造成濫用最常見的兩種方式是蟲和病毒(worm and virus)。

6.6.1 蟲(Worm)

蟲是使用複製功能來癱瘓系統性能的行程。蟲產生本身的複製品，佔據系統資源並可能不讓其他所有行程使用系統。在電腦網路上，蟲的臧脅更大，因為它們可在系統間不斷複製而癱瘓整個網路。1

蟲是由兩個程式所組成，一個是掛鉤(grappling hook)程式和主程式。掛鉤程式叫做 /1.c，是由 99 行 c 語言程式碼編譯而成，並在它能觸及的機器上執行。一旦建立在被侵略的系統上，掛鉤程式就連到它原先的機器，並且載入主程式到鉤上的系統)。主程式繼續搜尋其他可以輕易連上這台剛被感染電腦的機器。在這些動作中，可以利用 UNIX 網路中的公用程式 rsh 以方便遠程任務的執行。藉著建立列有主機姓名組合的特殊檔案，使用者可以在每一次利用這些組合使用遠端帳號時省略掉敲入密碼。蟲搜尋這些特殊檔案中不需要密碼就可以做遠端執行的站名，只要遠端的殼建立起來，蟲的程式就被載入並開始執行新的。

經由遠端存取所達成的侵略是蟲的三種感染方法之一。另外兩種方法牽涉到作業系統中在 UNIX finger 和 sendmail 程式裡的錯誤。公用常式 finger 是做為電子電話的目錄指令，可以傳回一個人的真實和簽到名字，以及使用者可能提供的其他資料，例如辦公室和家裡地址與電話號碼、研究計畫、或生日。finger 是以背景行程(daemon)的方式在每一個 BSD 站執行，並且對整個 Internet 做反應。

使用在 sendmail 中的錯誤也是使用一背景行程以做為惡意侵入。sendmail 安排網路環境中的電子郵件。偵錯功能對系統管理者很有用，而且通常被放在背景行程。一旦進入位置，主要的蟲有系統的試圖找出使用者的密碼。它由最簡單的沒有密碼或由使用者帳號名稱組合建成密碼的情形開始試，接下來和內部字典的 432 個常用密碼選擇做比較，到最後階段的用標準 UNIX 線上字典的每一個字做為密碼嘗試。這種巧妙和有效的三階段破解密碼法則使蟲可以進一步取得被感染系統的其他使用者帳號。然後蟲搜尋破解帳號的 rsh 資料檔案。任何 rsh 都會被試，然後就和前面敘述過的一樣，蟲可獲得在遠端系統的使用者帳號。

6.6.2 病毒(Virus)

另一種電腦攻擊的型式是病毒。病毒和蟲一樣，都是設計成散佈到其他程

式，並且可能在系統中造成大破壞，包含了修改或破壞檔案、造成系統毀損和程式不正常動作。然而蟲是建造成一個完整、獨立的程式，而病毒則是加在合法程式中的片段程式碼。病毒是電腦使用者的一項主要問題，尤其是微電腦系統。通常使用者電腦不容易受到病毒困擾，因為可執行程式受到作業系統保護而無法寫入。即使病毒感染到一個程式，它的效能依然受限，因為系統的其他方面受到保護。單一使用者系統沒有這些保護，因此病毒可以自由橫行。

通常病毒藉由從公用佈告欄或交換包含有感染病毒的軟碟片來散佈。例如米開朗基羅(Michelangelo)的病毒就是這樣的例子，該病毒被安排在這位文藝復興大藝術家 517 歲生日，也就是 1992 年 3 月 6 日消除掉被感染的硬碟檔案。因為廣泛的宣傳圍繞在此病毒，使大部份美國的電腦站在病毒發作前先找出並摧毀此病毒，所以它造成很小或者說幾乎沒有傷害。這些情況使大眾注意，並且警告他們關於病毒的問題。掃毒程式現在是非常好的販賣產品。大部份的商業套裝軟體只針對特殊知名的病毒有效。它們藉著搜尋系統上所有程式的某一特殊型式指令組(組成該病毒的特徵)來工作。當它們找到一個已知型態時，它們移去這些指令，以解除此程式的病毒。這些商用套裝軟體有它們所搜尋之數百種病毒的目錄。當每一種病毒皆被條列出來後，使用不同技巧的其他病毒就出現。譬如，在 1991 年的‘梅麗莎’(Melissa)病毒廣泛地傳佈，並且妨礙許多電子郵件伺服器。它使用 Microsoft Outlook 的巨集功能，並且偽裝成一個重要的訊息。當使用者開啟此訊息時，巨集指令就被觸發，並且送出 Email 訊息給此使用者通訊錄的前 50 位。

保護電腦免於電腦病毒的最佳保護方法是預防的習慣。從購買未開封軟體、以及避免從公用資源或交換磁碟片得到免費或盜版軟體，是預防感染的最佳途徑。然而，即使新拷貝的合法應用軟體也不能免於病毒感染。

另一項防衛雖然不能避免感染，但卻能早期偵測出病毒。使用者必須先完全重新對硬碟做格式化，尤其是啟動磁區，啟動磁區通常是病毒攻擊的目標。只有安全的軟體才載入，並且對每一個檔案皆計算出一個核對值(checksum)。此核對值的表列必須避免被非授權地存取。在任何一次系統重新開機時，一個程式可以重新計算核對值，並且和原先表列的核對值做此較，只要有任何不同都可做為被感染的警告。

6.7 威脅監督(Threat Monitoring)

兩種管理技術可以用在增進系統的保密性上。一種是威脅監督。在企圖偵測保密違背時系統可以檢查可疑的活動模型。這種方法的一個普通例子是分時系統，當一個使用者登錄時，它記錄了使用者給不正確密碼的次數。超過了一些不正確的次數後，則表示使用者企圖去猜出密碼。

另一個普通的技術稱為審核登記(audit log)。一個審核登記簡單地記錄時間、 使用者及對一個物件所有存取的類型。當保密性遭到破壞，審核登記能用來決定問題是如何及何時發生，並且可能危害的程度。這些訊息在從損害中復原

及在可能發展更好的保密方法來防止未來的問題上將很有用。很不幸地，登記可能變大，登記使用掉系統資源，因此這些資源就無法給使用者取得。

取代登錄系統活動的是，我們可以週期性的巡視系統以防止安全的漏洞。這些巡視可以在電腦使用較少的時候來做，因此它們比登記造成的反效果小。這種巡視可能檢視系統的一些現象：

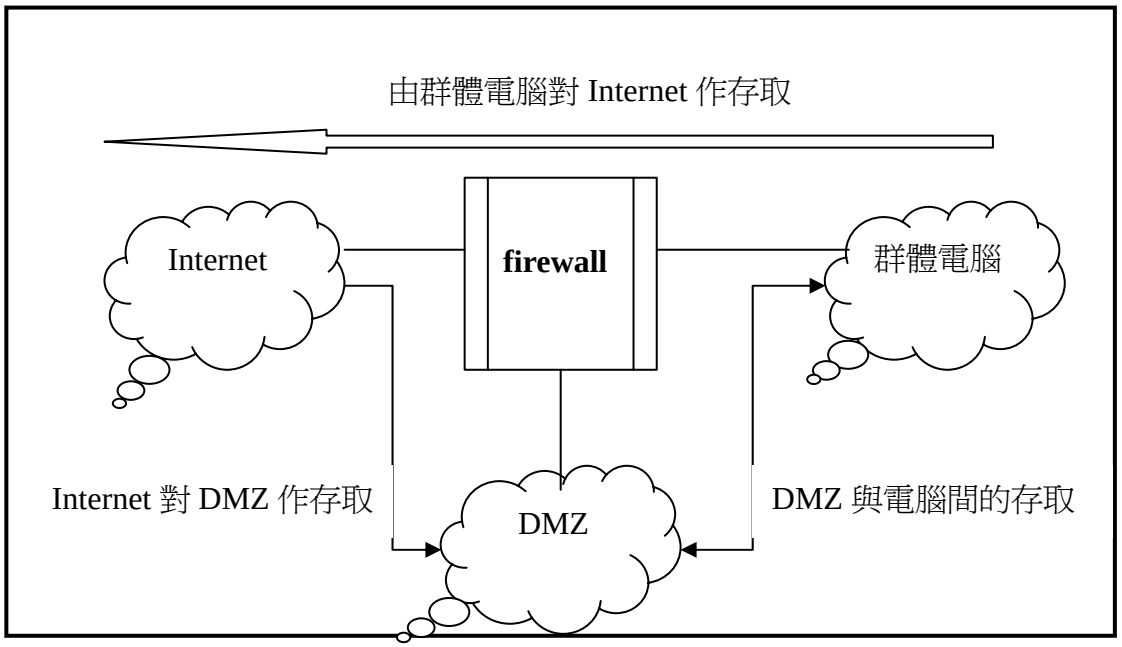
- 短或容易猜中的密碼
- 非授權的設定使用者辨別碼程式
- 在系統目綠中的非授權程式
- 非預期的長期執行行程
- 對使用者和系統目錄的不正確目錄保護。
- 對系統資料檔的不正確保證，例如密碼檔、裝置驅動程式、或甚至於作業系統核心本身
- 在系統搜尋路徑的危險侵入(例如木馬)
- 變更到系統程式來檢查核對值

任何由安全巡視所發現的問題都可以自動地修補，或報告給系統管理者。網路式的電腦系統較孤立式之系統更容易遭保密性之破壞。除了冒著來自一組已知的存取點之破壞危險外，還面對著來自相當多未知存取點之破壞危險—這是一個潛在嚴重之保密性問題。更進一步來說，以數據機經由電話連接之系統亦暴露於危險中。

有些政府設備採取最極端的保密預防。插入到和機密電腦溝通的終端機插頭，在該終端機不使用時就鎖在辦公室中的保險箱。一個人必須同時知道實際上鎖的組合，和電腦本身的辨識，才可以取得電腦使用權。

很不幸地，對於系統管理者和專業的電腦保密而言，將機器鎖在房間之中或者拒絕所有的遠方存取常常是不可能的。例如，Internet 網路經常連接數百萬部的電腦。對許多的群體和個人而言，它變成不可缺乏而且很重要的資源。在一些結合數百萬成員的群體之中，有許多好的成員和一些不好的成員。這些不好的成員有許多可以使用的工具，以便越過 Internet 企圖取得內部連結電腦的存取。

問題是如何把可信任的電腦在一個不值得信任的網路之中安全地連結？一個解決方法是使用防火牆(firewall)來分隔可信任或不可信任的系統(見下圖)。



防火牆是一部電腦或是一個繞路器，它設立在可信和不可信任的系統之間。它限制兩個安全範圍(security domains)之間的網路存取、監督和記錄所有的連結。例如，資訊網站伺服器(web servers)使用 http 協定來連接資訊網路瀏覽器。因此，防火牆可能　需要允許 http 通過。事實上，防火牆可以將網路區分成許多個範圍。一般的製作有如同 Internet 的不可信任範圍，稱為廢棄領域(DMZ：demilitarized zone)的半可信任和半保密網路的另一個範圍，以及一整個群體的電腦之第三種範圍(參考下圖)。允許從 Internet 到 DMZ 電腦的連結，以及從群體的電腦到 Internet，但是不允許從 Internet 或 DMZ 電腦到群體的電腦。在這個作法之中包含所有的存取以及任何一個 DMZ 系統，它可以允許通過防火牆的協定潛入，但是仍然不可對群體電腦作存取。

6.8 電腦保密等級(Computer-Security Classification)

美國國防部的電腦系統信任評估標準在系統中指定四個保密的等級:A、B、C 和 D。最低位準(或最少保護)等級是 D 等級。等級 D 只由一個層次組成，並且　用在被評估的系統，但是它無法滿足任何一個其他保密等級的需求。例如，MS-　DOS 和 windows3.1 就是這個等級。

等級 C 提供無條件的保護和使用者的責任，以及他們透過使用審核能力的行為。等級 C 分為兩個層次：C1 和 C2。C1 層次的系統混入一些控制的形式，以便允許使用者保護私人的資訊，以及防衛其他使用者的意外讀取或破壞他們的資料。在相同的靈敏度層次之中，C1 的環境是一個合作的使用者存取資料環境。大部份的 UNIX版本是 C1 層次。

電腦系統之中的所有保護系統之總合(硬體、軟體、韌體)正確地實施保密政策，如大家知道的電腦信任基礎(TCB：Trusted Computer Base)。C1 系統的 TCB

藉由個人的命名或群的定義允許使用者指定和控制物件的共用，以便控制使用者和檔案之間的存取。此外，在他開始任何期望 TCB 傳達的動作之前，TCB 需要使用者認同他自己。這個確認是透過一個保護的功能或密碼完成。TCB 保護證明的資料。

C2 層次系統透過個人層次的存取控制增加 C1 系統的需求。例如，存取檔案的權利可以被指定為單一個人的層次。此外，系統管理人能夠基於個人的認同選擇地審核一個或更多使用者的行為，TCB 也從碼或資料結構的修正保護它自己。此外，先前的使用者沒有產生任何訊息，可以由其他存取已經釋放回系統的儲存物件之使用者得到。有些特別的 UNIX 保密版本已經達到 C2 層次的標準。

等級 B 是強制性的保護系統，它有 C2 層次系統的所有特性，加上它們在每一個物件加入敏感度分類表。B1 層次的 TCB 在系統之中保有每一個物件的保密分類表，這個分類表是用來判斷關於強制性的存取控制。例如，在某一個機密等級的使用者，不可以在具有比較高敏感度的保密層次之中存取檔案，TCB 也在任何人類可讀取的輸出之中的每一頁之上方和底部指示出敏感的層次。除了一般的使用者名稱密碼的授權資訊之外，TCB 也保有個別的使用者之授權和解除，並且最少提供兩個以上的保密層次。這些層次是階級狀的，例如，使用者可能存取任何一個物件，這個物件攜帶的敏感度標籤要等於或小於它的保密解除。

B2 層次系統擴大敏感度標籤到每一個系統資料，例如儲存物件。指定實體裝置的最小與最大的保密層次，它讓系統藉由實體環境的強制用來實施約束，此時的裝置已經被配置好了。此外，B2 系統提供隱藏的通道和可能導致隱藏通道被利用之事件的審查。

B3 層次系統允許存取控制列表的產生，這個列表代表使用者或群，對一個已知名稱的物件作存取是不被允許的。TCB 也包含監督事件的功能，可以指出保密政策的違背。這個功能通知保密管理者，並且在必要時用最少分裂的方法終止這個事件。

最高的層次是等級 A。A1 層次系統在功能上是相似於 B3 的系統架構，但是 A1 使用正規設計敘述和修正技術，這個方式允許高等級的保證，因為已經正確地製作它的 TCB。一個系統要越過 A1 層次可能需要藉由信任的人使用信任的機器來設計和發展才行。

要注意的是 TCB 的使用，只不過是用來確保系統可以實施保密政策的方向，TCB 並沒有指定應該用那一種政策。典型的已知計算環境藉由保密代理人對保密政策的保證和設計的認可來發展。無疑的，計算環境可能需要其他的保證，例如 TEMPEST 可以監視電子式的竊聽。TEMPEST 保證系統有一些接頭用來遮蔽，以便防止由溢漏產生的電磁場。這個遮蔽物保證房間或建築物以外的設備之接頭，是用來掩蔽以便不能藉由在末端檢測出要呈現的是那些訊息。

第7章 網路結構

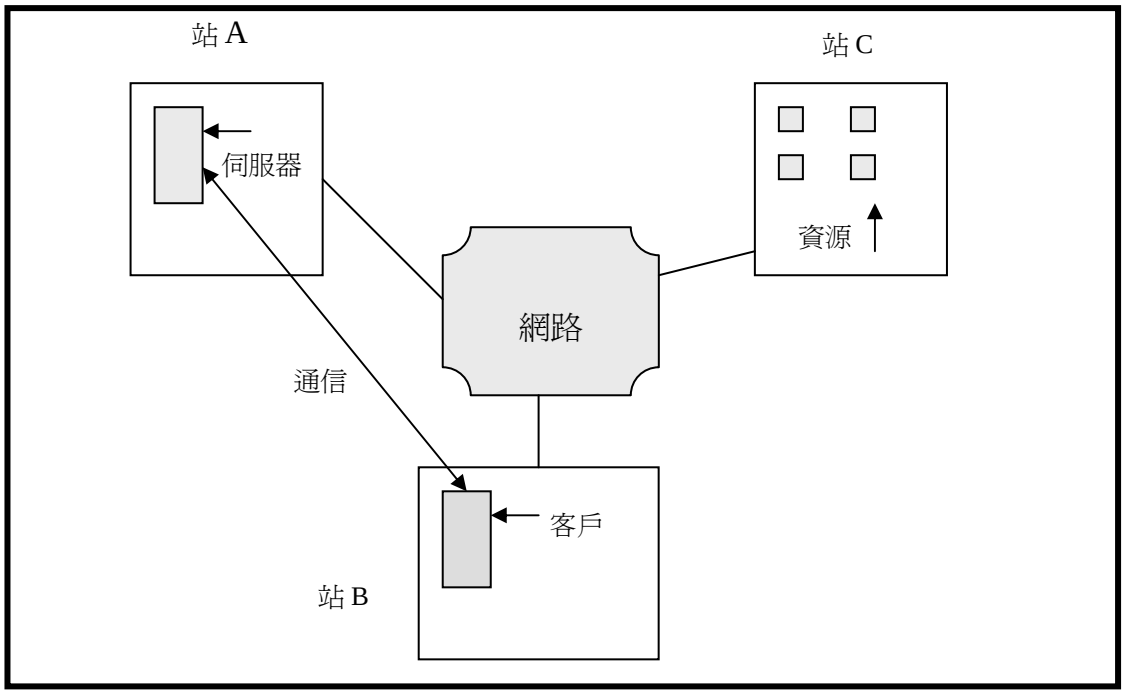
近來電腦系統中的一個傾向就是將計算工作分佈在幾個實際的處理機之間執行。基本上有兩種建立這類系統的方法。在多處理機(multiprocessor)的系統中，這些處理機彼此共用記憶體和同一時脈(clock)，這些處理機之間經由共同記憶體來互相聯繫。在分散式系統中，處理機並不共用記憶體或是同一時脈，而是每個處理機擁有它自己的區域記憶體。處理機之間必須藉著不同的通訊網路互相溝通，例如高速度匯流排或電話線。

7.1 分散式作業系統

分散式系統是藉由通信網路連接的一群鬆散耦合處理機。從系統中一個特定處理機之觀點來看，其他處理機及其資源是屬於遠程(remote)的，而其本身之資源則為局部(local)的。

在分散式系統中各個處理機可能在大小和功能上各有不同。其中可能包括小

型微電腦、工作站(workstation)、迷你電腦以及大型一般用途的電腦系統，這些處理機都被稱之為一些不同的名稱，譬如站(sites)、節點(nodes)、電腦、機器(machines)、主機(hosts)，以及其他等等。這主要是取決於描述它們的前後文而定。一般而言，一個站中的一台主機就叫做伺服器(server)，它擁有其他用戶(client，另一站的其他主機)所想要使用的資源。分散系統的目的就是為這種資源共用提供一個有效率且便利的環境。下圖顯示出一個分散式系統。



7.1.1 分散式系統的優點

建立分散的系統主要的理由有四項：資源共用、加速計算、提高可信度以及通信。

- 資源共用：如果有一些不同的站相互連接在一起，那麼在某個站上的使用者就可以使用其他站上所擁有的資源。舉例來說，在 A 站上的使用者就可以使用 B 站上可用的雷射印表機。同樣地，B 站上的使用者也可以存取 A 站的檔案。一般來說，在一個分散式系統中的資源共用都會提供方法來共用遠程站上的檔案，在一個分散式資料庫中處理資訊、在遠程站上印出檔案、使用遠程的特殊硬體裝置以及其他的作業。
- 加速計算：如果某個運算過程可以被分解為許多的小運算過程來同時執行，那麼分散式系統的能力就可以允許我們將其運算過程分散在不同的站上進行，以達到同時運算的效果。此外，如果某個站上目前的工作量超過

負載了，那就可以將其中一些工作移到其他工作量較輕的站上。這項移動任務的方式稱之為負擔共攤(load sharing)。自動負載分擔就是分散式作業系統自動地搬移工作，但是在商業化系統尚未普及。

- 可信度：如果在分散式系統中的某個站當掉了，其餘的站還可以繼續作業。如果系統是由一些自主性的設備所組成(也就是指一般用途電腦)，則其中之一的損壞將不會影響到其他的站。反過來說，如果該系統是由一些小機器所組成，而每一個站上都負責了某些重要的系統功能(譬如終端機字元的 I/O 或檔案系統)，那麼某一個單獨站上的失誤會嚴重地將系統的整個作業停止掉。一般來說，如果有足夠的重複作業存在此系統中(在硬體和資料兩方面)，系統可以在即使有某些站當掉了的情形之下，繼續它的作業。某個站若是產生錯誤就必須讓系統偵測到，並且需要採取一些適當的行動來修復產生錯誤的地方。系統必須不再使用該站上所提供的服務。此外，如果產生錯誤的那個站可以用另一個站來取代的話，系統必須保證能夠正確轉換其功能。最後，當產生錯誤的站被修好了之後，必須有辦法將它加入原來的系統中而不會產生與原先有的任何差異。
- 通信：當一些站藉著一個通信網路而連接在一起的時候，在不同站上的使用者就有機會交換資訊了。在低階中，訊息在兩系統之間傳遞就跟單一電腦訊息系統十分相似。在獨立系統中所有高階訊息傳輸功能都可以擴充到分散式系統。這些包括檔案傳遞、登入、郵件、全球資訊網和遠程程序呼叫(RPC)。分散式系統的優點是這些功能都能到達很遠的距離。一個計畫可以由兩個人在地理上分開的站上來實行。藉由傳遞計畫的檔案，登入相互的遠方系統來跑程式和交換郵件來合作工作，使用者能夠減少長距離工作的限制。

分散式系統的優點已造成工業界廣泛地趨向於縮小化。許多公司正把它們大型主機用工作站或個人電腦網路來取代。對於公司所造成的優點包括了省錢，儲存資源和擴充設施的彈性，更好的使用者介面，以及更容易維護。

7.1.2 分散式作業系統型態

在分散式作業系統中，使用者以存取自己資源之方式來存取遠程資源。從一站至另一站之資料與行程的轉移均在分散式作業系統控制之下。

資料轉移(Data Migration)

當 A 站的使用者想要取用 B 站所擁有的資料時，通常系統有兩種方法可用來傳輸資料。其一是將整個檔案傳輸到 A 站之後，所有存取檔案的動作就都被視為是區域性的了。直到使用者不再需要檔案時，再把檔案拷貝回 B 站(如果內容有過修改的話)。此功能可視為自動化的 FTP。，但不是很有效率。

另一種方法是只將目前工作上所需要的檔案部份傳送到 A 站。如果處理一陣子之後，需要另一部份的資料，再把另一部份資料傳輸進來。當使用者不再需要這個檔案時，就要把修改過的部份拷貝回去即可

很明顯地，如果所存取的，只是大檔案的一小部份，則第二種作法顯然較為理想。相反地，如果檔案的絕大部份都需要被使用到，則第一種作法效率應該比較高。

要注意的是，就是兩站之間並不只是具有傳輸資料的能力便夠了。如果兩處理站並不相容，系統就必須提供某些資料轉換的功能〈例如，如果它們使用不同的字碼或是用不同數目或順序的位元表示整數)。

運算轉移(Computation Migration)

在某些情況，系統如果能夠傳送運算內容，可能比傳送資料要有效率；這種做法就叫做運算轉移(computation migration)。比方說有一個工作程式，希望存取多個位於不同站的檔案，而其存取的目的，是想從中獲取某些資料項的統計數字。這種情況下，如果我們能夠讓檔案存取的動作，各自在其所在工作站內處理，然後將運算結果傳給需要這項資料的工作站，則執行效率將會大大地提高。一般而言，如果傳送資料的時間比執行遠程指令的時間久，就應該使用遠程指令。

像這種運算我們可以有很多種方法來處理。假設行程 P 希望存取 A 站內的某個檔案，存取的動作在 A 站本身執行，但是卻可以經由一個遠程程序呼叫(remote procedure call)產生。RPC 使用資料圖規約(internet 上的 UDP)在遠程系統上執行常式。行程 P 引發 A 站事先定義的一個程序，這程序正確地執行之後，將結果以參數方式傳回給行程 P。

另一種作法是，行程 P 傳送訊息給工作站 A。作業系統便在 A 站建立一個新的行程 Q，以完成所指派的工作。當行程 Q 完成執行動作之後，利用訊息系統將結果傳回行程 P。在這種結構中，行程 P 和 Q 的動作可以同時進行，事實上，都可能有好幾個行程在不同的工作站同時運作著。

這兩種作法皆可以用來存取分散在各站的多個檔案。一個遠程程序呼叫可能會激發另一個遠程程序呼叫，或是造成訊息傳輸到另一站的動作。同樣地，行程 Q 也可以在其執行過程中，傳送一個訊息到另一站，這訊息接著又產生另一個行程，這行程可能會傳回一個訊息給行程 Q，或是繼續重複上述的動作。

行程轉移(Process Migration)

運算轉移的邏輯延伸就是行程轉移。當一個行程被呈交執行時，它不一定是在初始站上執行。這對在不同的站執行整個或大部份行程而言，可能是個優點，下列為此方法可能被使用的幾個理由：

- 負荷平衡(load balancing)：行程(或子行程)可以分散到網路的各個角落，以平衡整個系統的負荷。
- 加速運算：(computation speedup)：如果一個行程能夠分成好幾個子行程，

分散在多個處理站同時執行，則整個工作的回復(turnaround)時間，勢必會大大地減低。

- 硬體偏好：(hardware preference)：此行程可能會有某種特性，較適合在某種特殊處理機上處理的情況)。
- 軟體偏好(software preference)：行程可能需要某一站上所擁有的某種軟體，若是軟體不能移動，還不如將行程移過去處理以節省花費。
- 資料存取(data access)：就如同運算轉移一樣，如果用在計算的資料很多，則　　把行程轉移到遠端執行可能會比把所有資料傳送到自己電腦上有效率。

基本上我們可以利用兩種互補的作法完成網路中行程轉移的任務，系統可以企圖去隱藏已被使用者移開行程的事實。這個方法的好處是使用者並不需要為了完成該遷移而明顯地將程式編碼。這種作法通常是在同質系統中為求平衡系統負荷，以及加速運算結果而採用的。因為這種作法不需要使用者輸入來幫助系統在遠端執行程式。

另一種作法是允許使用者自己設定轉移的方式，通常是用在某項工作必須利用某項特定軟體或硬體設備才能完成的情況下。

7.2 網路型態(Network Types)

基本上有兩種網路型態：區域網路(local-area network)以及廣域網路(wide area network)。這兩種型態的差異，主要在其分散的程度。區域網路是指多個處理機散佈在小範圍之區域內，比方說一棟大樓或一些鄰接之建築物之內。而廣域網路則是由一群自主性的處理機，散佈於一大片地理區域之上，例如整個美國。

這些差異影響到通僧網路上速度和可靠性，在設計分散式作業系統時，必須加以審慎考慮。

7.2.1 區域網路(Local-Area Networks)

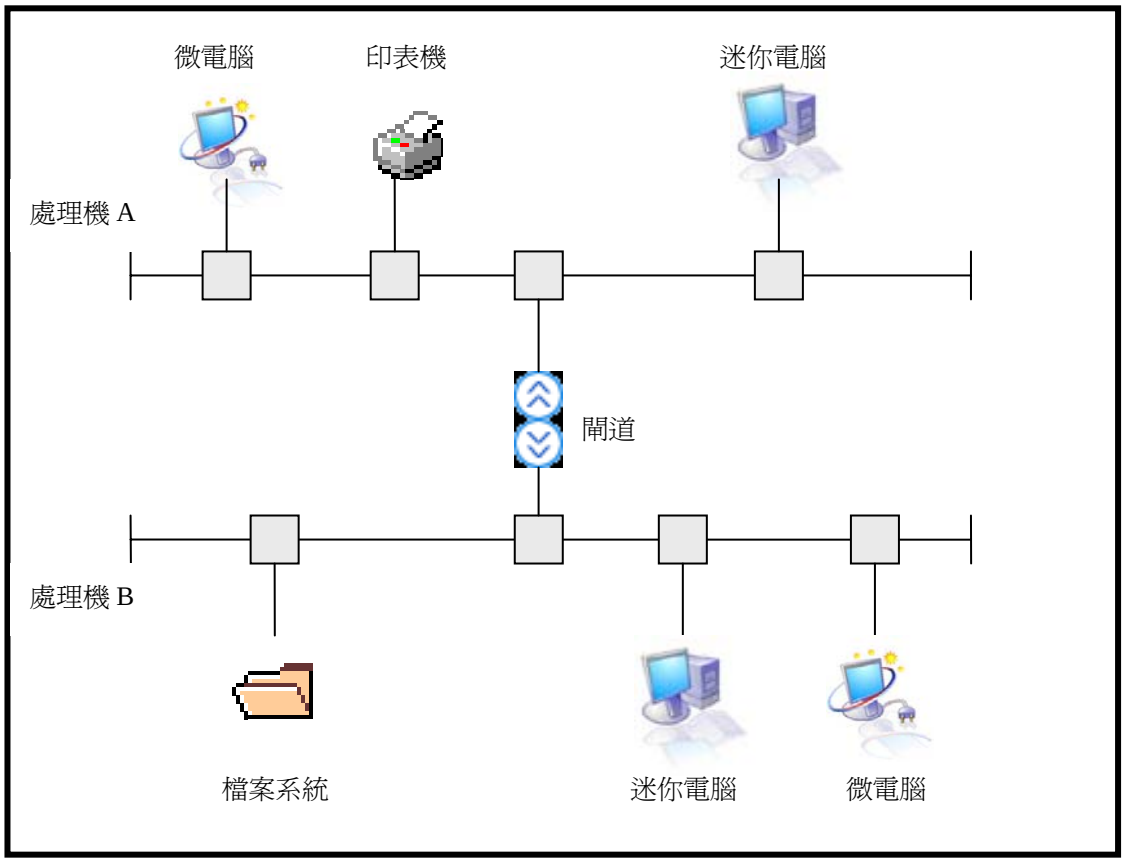
區域網路(LAN)起源於 1970 年代的早期，起初是為替代大型電腦系統用的，但是後來發覺對某些企業而言，擁有多部小電腦處理個別業務，要比單獨一部大電腦實用多了。因為每一部小電腦可能都需要使用全數的週邊設備，諸如磁碟機和印表機等，而且在同一個企業經營下，資料之間多少都有聯繫的必要，因此，將多套小系統連接成網路乃是必然的趨勢。

LAN 通常設計成涵蓋一個小型地理區域(例如一棟單獨建築物或一些鄰近的建築物)，而且通常用在辦公室的環境。在這種系統中的所有工作站都彼此很接近，所以通訊的連接傾向於比廣域網路有較高的速度和較低的錯誤率。為了可以

達到這種較高的速度和可靠度，就需要高品質的電纜。當然也有可能只使用電纜做為資料網路的交通。在較長的距離時，使用高品質電纜的代價十分高，因此只使用電纜就變得不太可能。

在區域網路中最常用的連接媒體是雙絞線和光纜。最常見的結構是多重存取的匯流排、環狀結構和星形網路。通訊速度從每秒 1M(百萬)位元到每秒 1G(十億)位元的光纖網路都有。每秒 10M 位元是最常見到的速度，也是乙太網路的速度。

典型的 LAN 常包含許多從大型電腦到手提型電腦或 PDA 等等的各種電腦、各種共用週邊設備〈如雷射印表機和磁帶機等)以及一個或多個閘道(特殊的處理機)，以便和其他網路相通(如下圖)。



其中 Ethernet 結構常被用來建立 LAN。因為使用廣播網路所以其結構沒有中心控制器，因此任何新增的通信站，都能夠很輕易地加入網路系統中。

7.2.2 廣域網路(Wide-Area Networks)

廣域網路(WAN)首先出現在 1960 年代，起初只是被當作是一種學術研究，探討如何利用有效通信，使得各站的軟硬體能夠經濟而方便地運用在團體中的每一個使用者身上。其中第一套發展成功的網路系統為 Arpanet，於 1968 年開始啟用，然後逐漸從原先只有四站連接的實驗性網路，發展成為世界性的網路系統 Internet，包含有數百萬個電腦的網路系統。

由於 WAN 中每一站在實際地理位置上都相當地分散，因此通信連結線的傳送相對地也就較慢較不可靠。典型的連接線有電話線、微波以及衛星通道等等。這些通信連接線都由其特定的通信處理機(communication processor)來加以控制。這些通信處理機必須負責定義各站透過那些界面來聯繫，以及傳送各站的資訊。

Internet WAN 此系統提供遠距離的主機(host)互相聯繫。主電腦之間不論在機型、速度、字元長度，甚至作業系統都可能有很大的差別。主機通常位於 LAN 之內，LAN 再經由區域性的網路連結至 Internet 上。在網路間之連接通常是使用稱為 T1 之電話系統服務來達成，T1 在整條租線上提供 1.544 M bit/sec 的傳輸速率。對於都需要較快速的 Internet 存取的站，可以使用由平行工作的多重 T1 單元所集台而成的 T1 組，用以提供更多的產量。例如，T3 是由 28 個 T1 連結所組成的，而且它的傳輸速率為每秒 4 仟 5 百萬位元組(45M bytes/sec)。Router 控制每個封包經由網路傳遞之路徑。該路徑可以是動態的以增加通訊效率，或是靜態的以降低安全的風險，或用來計算通信費用。

其他運作中之 WAN，使用標準電話線為其主要之通訊方法。數據機(modem)是從電腦端接收數位資料，然後把它轉換成電話系統使用之類比信號的裝置。在目的端的另一台數據機把類比信號轉回成數位，以便讓目的端的電腦接收。WAN 通常較 LAN 為慢，其傳輸速率介於 1200 bytes/sec 至 1M bytes/sec 之間。

7.3 網路通信

通信網路的設計者必須決定四個基本的論點：

- 命名和名稱決定：兩個處理機之間如何互相指名，以互相通信?
- 路徑策略 (routing strategies)：訊息如何在網路內傳遞。
- 封包策略 (packet strategies)：單獨地或如同串列般地傳送封包?
- 連接策略 (connection strategies)：兩台處理機之間如何傳遞一連串訊息?
- 爭執 (contention)：因為網路是一種共同使用的資源，我們要如何去解決使用上相互抵觸的需求呢?

7.3.1 命名

網路通訊的第一要件是網路系統的命名。對於一個在 A 站的行程要和 B 站的另一行程交換資訊時，他們必須能夠彼此指出對方。在一個計算機系統中，每一個行程都有一個行程識別碼。因為用網路連接的系統沒有共用記憶體，所以一開始時，他們並不知道另一台電腦上的目的行程，或甚至於是否有其他行程存在。

為了解決這個問題，遠端系統的行程通常是用一組〈主機名，識別符號〉來

辨別，其主機名(host name)是網路中獨一無二的名稱，而”識別符號”則可能是一個行程或是其他在那台電腦獨一無二的名稱。一個主機名通常是一個文數字的識別符號，而不是只使用一個數目字，如此才方便使用者去設定。

名字是方便人去使用，但電腦卻比較偏好數目字，以達到快速和方便。因為這個原因，必須要有一個機構去把主機名對映到主機號碼(host-id)，而主機號碼才是表明網路硬體上的目的地。主機名稱的情況，有兩種可行方法。第一種方法是，每一電腦主機都有一份資料檔案，其中包含了網路上可到達電腦的名稱和地址。問題是這種方法要從網路上增加或刪除一台主機時，需要對網路所有主機的資料檔做更新。另一種方法是把這項資訊分散到網路上的不同系統。網路必須使用某種規則來分散和取回這項資訊。這種方法就像是執行時的連結。第一種方法是原先使用在 Internet 的方法，但是當 Internet 變大時，這種方法變得無法維護，而第二種方法叫做區域命名服務(domain name service，簡稱 DNS)，則是目前使用的方法。

DNS 規定了主機的名稱結構，以及名稱對映到地址的結果。Internet 上的電腦主機在邏輯上都是用多重部份主機去命名。名稱是由最特定的部份到最普通的部份，並以逗點隔開每一項。例如”mail.im.tku.edu.tw”就表示了在淡江大學資管系的 ”mail”主機。通常，系統是以顛倒的順序去解析站名。每一部份都有一個名稱伺服器(name server)去接受名稱，並傳回負責該名稱之名稱伺服器的地址。最後，想要連接上之主機的名稱伺服器會被找到，然後就傳回它的主機地址。

這種規則似乎不是很有效率，但是每一台名稱伺服器都以快取記憶體來存取地址，以加速整個過程。當然如果名稱伺服器改變了或地址改變了，則快取記憶體的內容也必須隨時更新。事實上，這項服務十分重要，所以在存取規則中有許多最佳化方法和保護方法。試想如果主要的名稱伺服器壞掉了，則會如何呢?很可能就找不到主機可以用了。解決的方法是用第二台備份的名稱伺服器，其中拷貝了主要伺服器的內容。

在區域名稱服務(DNS)使用之前，Internet 上的所有主機必須拷貝一份包含網路上每一台主機站名和地址的檔案。這個檔案有任何改變都必須對主機 SRI-NIC 註冊，所有主機都必須每隔一段時間向 SRI-NIC 拷貝更新過的檔案，以便能夠和新加入的主機連接，或是和名稱已改變過的主機連接。在 DNS 下，每一台名稱伺服器只負責它領域中的主機資訊更新。DNS 在查詢過程中會自動地取出更新的資料，因為查詢時會和遠端主機直接接觸。在某一領域中，可以有獨立自主的副領域，以負責其範圍內的主機名稱和站名之改變。

一般而言，接受行程所指定〈主機名，識別符號〉訊息是作業系統的責任，作業系統還需要把此訊息傳遞給適當的主機。指定主機的核心則根據指定的識別符號，把訊息傳給所指令的行程。

7.3.2 路徑(Routing)

當在 A 站的行程想要和在 B 站上的行程通信時，訊息該如何傳遞呢?如果 A 到 B 之問只有一條實際路線(譬如在一個星狀或階層網路中)，通信就必須經由這條路線。但是，如果有許多條路徑可以由 A 通往 B，那麼就存在有許多條路徑可供選擇。每個站都有一個路徑表(routing table)，其中標示著可以用來將訊息傳至其他站的不同路徑。這個表裏面可能包括了關於速度和不同通信路徑的代價等資料，在必要的時候也需要更新其內容，可用手動或藉由程式來改變路徑的訊息。最常見的三種路徑是固定路徑、虛擬路徑和動態路徑：

- 固定路徑(fixed routing)：由 A 到 B 的路徑事先規劃好了，除非硬體出了毛病使這條路徑不通，否則將不會變更。一般來說，都是選擇最短的路徑，用以降低通信代價。
- 虛擬路徑(virtual circuit)：在某一段時間之內由 A 到 B 的路徑是固定的。但是在不同的區段內之訊息由 A 傳至 B 的路徑就可能不同。一個區段可以短到一筆檔案的傳遞，長到一次遠端登入。
- 動態路徑(dynamic routing)：當要傳遞訊息時才在 A 站與 B 站之間選出一條路徑來用。因為其決定是機動下達的，前後的訊息可能會交給不同的路徑傳遞。A 站將會做一個決定將訊息傳至 C 站；接下來輪到 C 決定一條路徑可將訊息傳至 D，如此下去。最後，某個站將會把訊息傳給 B。一般來說，一個站會將訊息傳送至在那一時間最少被使用到的站上。

在這三種方法之間有些魚與熊掌不可兼得的現象。固定路徑無法適應承載改變。換句話說，如果 A 與 B 之間已經建立了一條路徑，訊息就必須經由這條路徑傳遞，即使這一條路線非常忙碌而其他的路線卻很少被使用到。這個問題可以使用虛擬線路來得到部份解決，並且可在動態路徑中得到完全避免。固定路徑與虛擬路徑可以保證訊息能夠按照其被傳送的順序由 A 送抵 B。在動態路徑中，訊息抵達時可能不按照順序來。這個問題可以藉著附加一個循序號碼到每段訊息上來得到解決。

但是，動態路徑是最難建立和執行的，但它卻是在複雜環境下管理路徑的最佳方法。UNIX 同時提供了固定路徑和動態路徑，也有可能兩種混合。在一個站中，主機可能只需要知道如何達到連接區域網路到其他網路的系統即可。這個系統就叫做閘道(gateway)。個別的主機必須有一條固定路徑到達閘道。閘道本身則需要有動態路徑，以便能夠到達網路上其他的主機。

路由器(router)是電腦系統內負責路徑訊息的單位。Router 可能是一台擁有安排路徑軟體的主機或是一台特殊目的裝置。無論是何者，Router 必須至少有兩個網路連接，否則它就無法安排路徑訊息。Router 還決定是否任何一個它

從網路所
接收到的訊息，必須傳送到另一個它所連接的網路上。它是藉由檢查訊息目的地的 Internet 地址來做決定。Router 檢查它的表格以決定目的地主機的位址，或至少決定出送往目的地主機的網路。在固定路徑下，這個表格只能由人做改變。在動態路徑下，Router 之間會使用某種路徑規約(routing Protocol)以告知網路的改變，以及讓它們可以自動地更新路徑表。

閘道和路由器通常都是專門的硬體裝置。它們執行軟體的程式，而非執行一般用途的網路程式。

7.3.3 封包(Packet)

一般而言，訊息長度是不固定的。為了簡化系統設計，我們一般是用稱為封包(Packet)，欄(frame)或資料單元(detagram)的固定長度訊息來製作通訊。用封包方式製作的通訊，可以用非連結訊息的方式傳送到它的目的地。一個非連結訊息可能是不可靠的(unreliable)，在這種情形下傳送器不保證且不能告知，這個封包是否已經到達它的目的地。另一種方式是，封包是可信任的(reliable)，通常從目的地回送另一個封包來指示這個封包已經到達。(當然，這個回送的封包也可能在回程之中漏失)。假如訊息太長，以至於無法放入一個封包之中，或者假如這個封包在兩個通訊器之間需要向前或向後流動，則必須設立一個連結，以便允許多個封包可靠的交換。

7.3.4 連接(Connection)

一旦訊息到達它們的目的地，行程就可以開始通訊以交換資訊。有許多不同的方法可以用來將任一對想要在網路中聯繫的行程連接起來。最常見的三種方法就是線路切換、訊息切換和封包切換。

- 線路切換 (circuit switching)：如果有兩個行程彼此想要聯繫。一條永久性的連線就建立在他們之間。這條連結線在通信期間就配給他們，在這段期間之內其他的行程則不可以使用。這個方法和電話系統很相似。一且某條通信線已經開放給了兩個用戶，別人就不可以再使用這條線路了，一直要到通話終止之後才可以。
- 訊息切換 (message switching)：如果有兩個行程想要通信，一條暫時性的連結線就會在傳送一段訊息的時候暫時出現。實際的連結是當需要的時候動態地分配給使用者，並且只有很短一段時間。每段訊息乃是一個區段的資料，並且加入了一些系統資訊來促使通信網路能正確地傳遞訊息。這種方法與郵局發信系統很類似。每一封信都被視為一段含有目的地地址與來源地址的訊息。

- 封包切換 (packet switching)：一個邏輯上的訊息可能必須分為好幾個封包。每個封包則分別地送往其目的地，並且可能分別採取不同的路徑通過網路。這些封包在被收到之後還必須重新組合成原來的訊息。

各種方法之間存在著很明顯的不可兼得的問題。線路切換需要準備時間，但是傳送訊息比較快。訊息切換與封包切換所需的準備時間不多，可是其每段訊息的傳送訊息卻反而較長。同時，在封包切換中，每段訊息必須要分成封包並且稍後再重組回來。封包切換是資料網路上最常用的方法，因為它使網路得到最佳使用，而且對於資料分成封包沒有任何傷害，這些封包有可能個別地安排路徑，而最後在目的地再重新組合起來。

7.3.5 衝突

因為網路的架構不同，一條連結線上可以連接許多站，所以可能會產生許多站想要同時在一條線路上傳送資訊的情況。這種狀況主要發生在環狀或多重存取匯流排網路中。在這種情形之下，被傳送的資訊可能會擠成一堆，而必須將之廢棄。站獲知這個問題，如此方能重新發送所需要的資料。如果沒有特殊的功能來處理它，這種狀況就會反覆出現，結果造成性能降低了。目前有許多方法都是設計出來避免反覆衝突，其中包括碰撞偵測(collision detection ,CD)、許可證傳送(token passing)和訊息槽(message slots)。

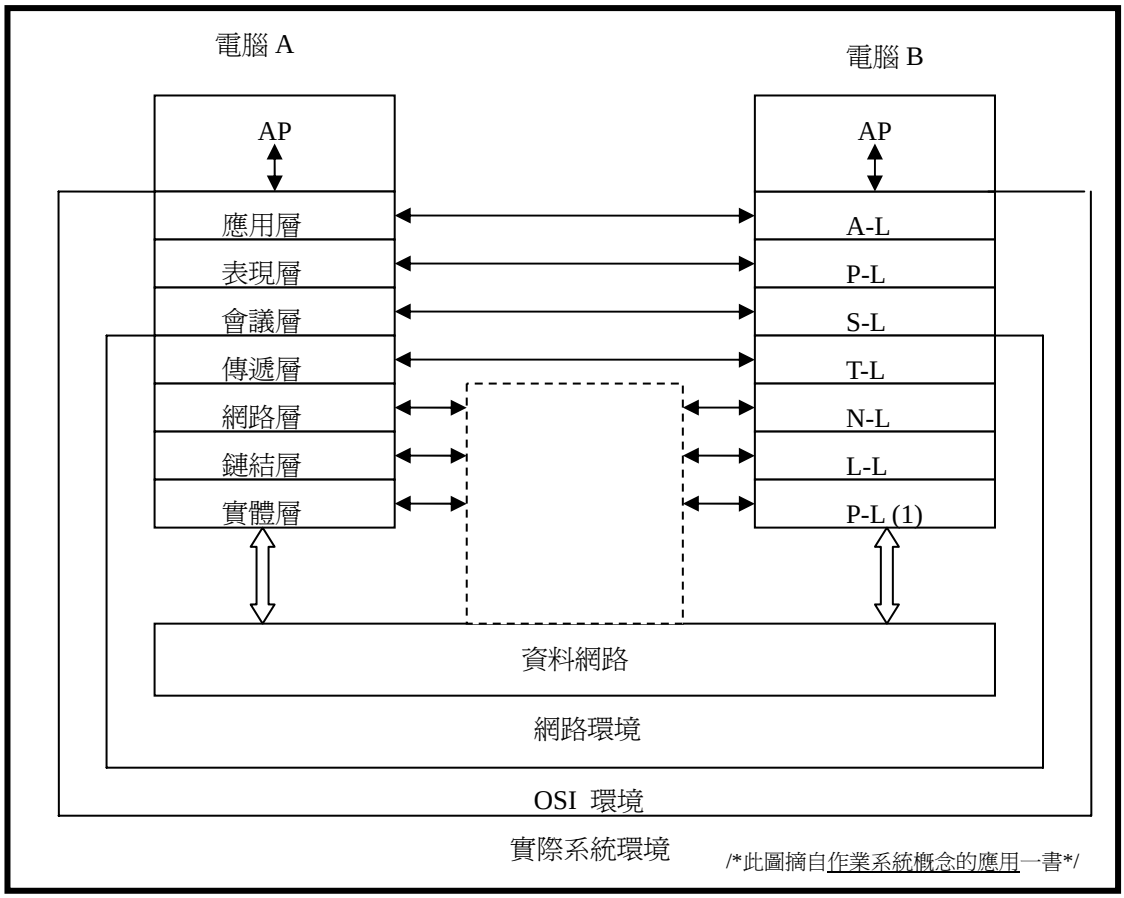
- 碰撞偵測 (CSMA/CD)：在某條線上傳送一段訊息之前，某站必須先檢查看看是否已有其他的訊息目前正在該條線上傳送，此技術稱為 CSMA(carrier sense with multiple access)。如果該線有空，這站就可以開始傳送了。否則，它就必須等到該線有空的時候。如果正好在某一時刻有兩個以上的站開始傳送(每一個都認為當時沒有別人在使用)，那麼它們就必須做CD(collision detection)並且停止傳送。然後每個站在任意一段時間之後再試試看。當某個站開始在一條線上傳送時，它必須不斷地傾聽，以偵測來自其他站的訊息衝突。這種方法的主要問題是當系統非常忙碌，可能產生許多衝突的時候，系統的性能就會降低了。這個 CSMA/CD 的方法曾經成功地使用在最普遍的網路 Ethernet 系統上。當系統變快時，他們在每一時段可以送出更多封包。因此，為了保持網路性能合理，每一乙太區段的系統個數就減少。
- 許可證傳送 (token passing)：一種獨特的訊息類型，稱為許可證(token)，不斷地在系統中循環走動(通常是在環狀結構中)。一個等待著要傳送資訊的站必須等待許可證到達。它就把該許可證從環中拿走並且開始傳送它的訊息。當該站要發送的訊息傳送完了之後，它就把許可證重新傳出去。這項行為接下來將會允許另一個站持有許可證並且開始傳遞它的訊息。如果許可證遺失了，系統必須要能發現它的遺失並且再發出一個新的。其解決方法一般

是去選出一個新的站出來，被選到的站於是就令許可證重新開始走動。

- 訊息槽 (message slots)：一些固定長度的訊息槽在系統中不停地來回走動。每個槽內可以持有一段固定長度的訊息以及一些控制資料(譬如來源、目的地和該槽是滿的或空的)。一段訊息必須等到一個空槽到達才能傳送。然後把其訊息插入槽中，設定適當的控制資料。這個槽就繼續在網路內周遊。當抵達某個站的時候，該站就必須檢查其控制資料，以判斷槽內所載之訊息是否是要送到這個站上的。如果不是，該站就將槽與訊息再傳送下去。否則，它將把槽內的訊息取走，並且重新設定控制資料來標示這個槽此刻是空的了。接下來該站可以使用該槽來傳送它本身的訊息或者釋放這個槽。因為一個槽裡面只能存放固定長度的訊息，一段邏輯上的訊息可能必須要被打散為一些較小的封包，每個封包則由不同的槽來傳送。

7.4 通信協定(Communication Protocol)

當設計一個通信網路的時候，我們必須要在一個潛在性緩慢並且有錯誤傾向的環境下，處理不同步作業間溝通時的協調。網路間的系統共同以某一協定，或一組協定以決定站名，找出某一站在網路上的位置，建立連結等事項也是很重要的。我們可以問題分為許多層來簡化設計的問題。一個系統中的每一層和其他系統的同一層溝通。每一層可以有自己的協定。此規則可以用硬體或軟體來實現。下圖顯示兩台電腦的邏輯通信，其中最低的三層是以硬體來製作。



按照國際標準組織(International Standard Organization ,ISO)的規定，我們用以下的敘述來討論這些分層：

1. 實體層 (physical layer)：實體層負責處理實際傳遞一串位元串的機械與電子二進位的 0 和 1 信號部份的細節
2. 資料連結層 (data link layer)：資料連結層負賣處理固定長度的封包，包括在實體層中任何錯誤時偵測與修復。
3. 網路層 (network layer)：網路層負實提供通信網路的連接以及封包的傳送，其中包括要送出的封包位址、被送來封包的位址解碼、並且為了正確反應承載變動而保存的路徑資料。路由器在此層工作。
4. 傳遞層 (transport layer)：傳遞層負賣網賂的低階存取和使用者之間的訊息傳送，其中包括將訊息分割為封包、保存封包的次序、流程控制與實際位址的產生。
5. 會議層 (session layer)：會議層負責實現會議、或行程對行程的通信協定。通常這些協定是遠端登入、檔案和郵件傳遞的真正通信。
6. 表現層 (presentation layer)：表現層負責解決網路中的不同站上所使用的格式，包括字元轉換，以及半雙向和全雙向的轉換等。
7. 應用層 (application layer)：應用層負責直接和使用者接觸的工作，這一層負責檔案傳遞、遠端登入協定和電子郵件，以及訂定各分散資料庫的型態。

大部份 Internet 的站依然使用 IP(Internet Protocol)通信。服務是經由無連線的 UDP(User Datagram Protocol)和有連結方針的 TCP/IP(Transmission Control Protocol/ Internet Protocol)在 IP 的最上層完成。TCP/IP 協定比 ISO 模式少好幾層。因為 TCP/IP 將幾組功能結合放在同一層裡，所以比較難做到，但卻比 ISO 的連結方式有效率。IP 規則負責傳遞 IP 資料段(datagram)，IP 資料段是在 TCP/IP 網路上最基本的資料單位。TCP 使用 IP 在兩個行程之間傳遞一串可靠的資料。UDP/IP 是一種不可靠、沒有連接的傳送規則。它使用 IP 來傳遞封包，但加入錯誤更正碼以及一個協定端地址，以訂定此封包將前往的遠端系統之行程。

7.5 故障排除

分散式處理系統常會遭遇各種型態的硬體故障。其中最常發生故障的部份大概就是連結線部份、站本身，有時候訊息傳丟了也是個問題，因此，如果要維持系統正常地運轉，必須要能夠偵測出這些故障，以便重組系統使其恢復運算，以及在連結線或站修復之後能夠重新加入系統內。

7.5.1 故障偵測

在沒有共用記憶體的系統中，通常並不容易區別，到底是站故障、連結線故障還是訊息傳送錯誤。我們只能知道，其中某一種錯誤情況已經發生，但究竟是那一種，我們並不知道。一旦錯誤情況被偵測出來，應該採取何種應對措施，那就要依各種應用情況而定了。

要偵測連結線和站是否發生故障，我們可以採用握手的方式(hand shaking)來進行。假設站 A 和 B 之間存在一條實際連結線(physical link)，則在固定時段 內，兩個站都必須傳送一個訊息—*I am up* 給對方，告訴對方自己還好好地在那裡工作著。如果站 A 在時間內沒有收到站 B 傳送的聯繫訊息，則 A 假設 B 已經發生故障，或 A 到 B 之間的連結線發生故障，要不然就是聯繫訊息傳丟了。在這個時候，站 A 可以有兩種選擇，一種是等著下一個時段看 B 會不會再傳送訊息來；另一種是主動以訊息 *Are you up?*去詢問 B。

如果站 A 還是沒有收到回音，那麼它可以再重複這些動作，但是這樣站 A 也只能得到一個結論，就是其中有某些部份發生故障。

這時候站 A 可以試著從其他連結來跟 B 連結，傳送一個 *Are you up?*的訊息給 B。如果 B 接到這個訊息，並透過另一連結線立即回答 A，則表示原先的錯誤是發生在 A 和 B 的直接連結線上。但是因為我們並沒有辦法事先預估訊息從 A 傳到 B 再傳個來所需要的時間，因此我們必須使用一種時間終止(time-out)的方法。當 A 送出訊息 *Are you up?*時，它必須同時指定一個時間值；如果在這時間

值內 B 回答了,那麼 A 便可以斷定 B 還是好好的。但是如果超過這時間值 B 還　沒有回答，則 A 還是只能斷定底下四種情況中，必有一種或多種已經發生：

1. 站 B 已經故障
2. 從 A 到 B 之間的直接連結線已經故障
3. A 到 B 的另一條連結線已經故障
4. 訊息被傳送丟了

但是站 A 還是沒有辦法斷定，究竟是發生了哪一種情況。

7.5.2 重組架構

假使站 A 透過上述偵測方法，發現其中確實已有故障情況存在，則它必須啟動一個程序，使系統能夠重組其架構，以恢復正常的運作。

- 如果是 A 到 B 之間的直接連結線故障，則系統必須將此狀況傳播給每一工作站，使各站能及時更新所保有的各種路徑表。
- 如果其中某一個站確實已經壞了，則系統必須告知其他每一個站，以免再有站試圖要去使用故障站所擁有的資源。如果發生故障的，是擔負某種協調工作的站(如死結偵測等)，則必須選擇另一個新的站來充當。同樣地，如果故障的是邏輯環狀結構中的某一站，則必須另建一新的邏輯環狀結構。

但是萬一站根本沒有故障，而只是沒有辦法連結到，則可能會造成兩個協調程式同時並存的現象。當整個網路被分成兩部份之後，兩個協調程式〈每一部份一個〉之間，可能會發生衝突的現象，比方說，假使協調程式是負責實施互斥，我們可能會發現造成兩個行程同時進入臨界區間執行的現象。

7.5.3 故障修復

當故障的連結線或站修復之後，我們必須很順利地將它加入系統內。

- 如果是 A 和 B 之間的連結線故障，則在修復之後，必須同時通知 A 和 B。這通知的動作可經由“握手"方式來完成。
- 如果是站 B 故障，則在修復之後，它必須通知每一個站，說它已經恢復運作了。然後接著站 B 便可以陸陸續續地再接到其他站傳給它的資訊，好讓它可以更新其區域控制表，諸如路徑表、故障站的名稱等。

單元二 Linux 作業系統個案研究

第1章 Linux 的發展歷史

1.1 Linux 的創造

〝你好，所有使用 minix 的人 -我正在為 386 (486) AT 做一個免費的操作系統 (只是為了愛好)，不會像 GNU 那樣很大很專業。〞

1991 年 8 月，芬蘭的一個學生在 comp.os.minix 新聞組貼上這段話，這個人就是 Linux 的創造者 Linus Torvalds，他在 803861 的微處理器上，寫了一個麻雀雖小五臟俱全的核心，並命名為 Linux。

在一個被一群軟體巨頭統治的(系統)軟體王國裡，Linux 似乎是個謎。一個由一些電腦 hackers 編寫的作業系統如何能夠參與競爭？一個由不同的國家許多不同的人編寫的軟體如何能夠保持其穩定性和高效性？這裡的答案是 Linux 具有非常好的可靠性，高效性和競爭能力。許多大學和研究機構都在用 Linux 來作計算。許多人們已在其 PC 上安裝了 Linux。絕大多數公司都或多或少地在使用 Linux。Linux 被廣泛地用來瀏覽 web 站點，檔案處理，發送 email ，玩電腦游戲。Linux 絕不是一個電腦界的玩具，而是一個由全世界的愛好者開發的非常完善的，專業化的作業系統。

Linux 的源頭可以追溯到 Unix 家族。1969 年，貝爾實驗室的研究人員 Ken Thompson 其開始在一台空閒的 PDP-7 機器上實驗其多用戶，多任務的作業系統 (multi-user, multi-tasking operating system) 。不久 Dennis Richie 和其他兩位同事加入了他的行列。他們與實驗室中的其他同事一道開發出了最早期的 Unix 版本。Richie 在早期的項目 MULTICS 中發揮了很大的作用。Unix 其實是 MULTICS 的雙關語。早期的 Unix 是用組合語言寫的。第 3 版時採用了 C 語言。C 語言是 Richie 設計並編寫的，以用來作為編寫作業系統設計的語言。用 C 改寫過的 Unix 使得 Unix 可以被移植 PDP-11/45 和 DIGITAL 11/70 電腦上。Unix 移植到 DIGITAL 11/70 是一個歷史性的轉折，使得 Unix 正式從實驗室走向大型主機計算環境。很快，絕大多數的電腦制造商都發布了其相應的 Unix 版本。

Linux 誕生的原因極其簡單。Linus Torvalds，Linux 的作者和主要管理者，當時窮的只能夠付的起 Minix。他對 MINIX 提供的功能不滿意，自行發展 Linux，因為在學校裡每天用的都是 Unix， 所以他選擇將 Unix 作為他的軟體的

模型.最開始的工作是在一台 Intel 386 的PC機上。他的進展很迅速。Linus對他所做的事情充滿了興趣，並通過剛剛出現的，還侷限在學術領域的電腦網路，將已有的程式碼共享給其他的學生，他的原始碼被免費公開在 Internet 上，允許人們自由使用 (under GNU Public License)。其他的人看見了 Linus 的軟體並開始加入開發的行列。不同的人由於在使用 Linux 時碰到不同的問題，所以這個軟體也就不斷地被更新和完善。因此他的開發過程，其實是來自世界各地的許多使用者，所共同建造的。Linux 是第一個完全免費的 Unix，很快地，許多人開始修改及加強 Linux，如今，Linux 除了可以在原先設計的 Intel x86(x>=3) 上執行外，它也被移植到 Alpha, Sun Sparc, Motorola 68K, MIPS, PowerPC 等等的平台上。

Linux 在開發的早期炷要的重心放在作業系統核心，它是一個具有特權的管理程式，負責管理所有的系統資源，並能直接控制硬體。不久之後，Linux 就成為一個完整的作業系統了。值得注意的是 Linux 中沒有任何 Unix 程式碼，而是根據 POSIX 標準重新編寫的 Linux 中使用了許多在 Cam bridge , Massachusetts 的 Free Software Foundation 的提供 GNU 軟體。

Linux 的核心和 Linux 系統是兩個名詞。Linux 的核心是真正完全由 Linux Community，由零開始開發完成的一塊軟體零件；但被稱為 Linux 系統的整套軟則包含了很多不同的元件，有從頭開始開發的，也有些是從其他開發計劃借用來的，還有一部分是和其他的研究小組共同開發的。

多數人僅把 Linux 當做一個簡單的工具來使用。也有許多用戶在 Linux 進行應用程式的開發。只有很少數的人敢於為 Linux 寫設備驅動程式和核心的程式。從事 Linux 核心開發的人員畢竟是只有一少部份人。大多數 Linux 用戶似乎不關心這個作業系統是如何構造和運行的。但對於學習 Linux 要能更理解一個作業系統功能的有利途徑，去關心這一部分是必要的。

Linux 不僅僅是設計的很好，更重要的是其原始碼是公開的。這是因為雖然作者擁有這個軟體的版權，但在 Free Software Foundation's GNU Public License 的基礎上，原始碼是可以免費獲取的。

除了核心程式以外，一個作業系統還需要其他的系統程式跟應用程式才有實用性，Linux 系統中常用的系統程式大部份是美國自由軟體基金會 (Free Software Foundation) 開發出來的軟體，而且也有不少機構或個人利用自己閒暇時間，不計報酬的為 Linux 開發應用程式，這些程式一樣大多都是自由軟體，任何人都可以免費的在網路上取得。　　不過自行去取得這些程式再一一安裝非常不便，於是便有某些有系統整合能力的公司會去搜集、整合 Linux 上的程

式，把〝核心-系統程式-應用程式〞總合起來構成一個完整的作業系統，讓一般使用者可以簡便的安裝完整個系統，這就是所謂的〝安裝套件〞
(distribution) 。

我們一般講的 Linux 系統便是針對這些安裝套件而言，Linux 安裝套件的種類繁多，著名的有早期在台灣非常流行的 Slackware 套件，還有近來越來越多人使用 Debian 跟在國外已經是佔有率超過一半的 RedHat Linux，這些不同的安裝套件都算是 Linux 系統，同樣都用 Linux 核心，收錄的程式大同小異，相互之間的程式都可以共用，不同的地方只在於一些系統設定跟程式套件的管理方式而已。同樣是 Linux 系統，卻分成不同公司、機構整合出來的不同安裝套件，這就是大家常常在網路上看到 Linux 有那麼多〝種〞的原因。

Linux 具有 UNIX 系統的程式介面跟操作方式，也繼承了 UNIX 穩定有效率的特點，網路上安裝 Linux 的主機連續運做一年以上而不曾當機、不必關機是稀鬆平常的事，不過 Linux 但是卻不像一般 UNIX 須負擔龐大的版權費用、而且要在專屬的昂貴硬體上才可以使用。

Linux 可以在一般的 i386 PC 上執行，自然而然的接收了過去幾十年來在 UNIX 上累積的程式資源跟使用者，加上 GPL 的版權允許大家免費傳播散佈 Linux 的原始碼，並針對自己的需求修改程式，使得 Linux 在目前已經成為非常受人歡迎的一個多人多工、免費、穩定、高效率、可以在包括 i386、 Sparc、Alpha、Mips、PPC……等眾多不同電腦系統平台上執行的作業系統。 提供中文化簡易〝控制台〞，MIS 可透過 web 中文畫面輕易設定 email 帳號、 DNS、proxy，……等等參數。

基本上 Linux 是〝免費〞的，你可以自由從 Internet 上傳下載。不過由於 Linux 連其附屬軟體最少也有幾百 MB，為免浪費上網的費用及時間，大部份人都會購買其 CD-ROM。其 CD-ROM 一般由一百元到千元不等，(其實你可以由網上下傳合法燒錄於 CD-ROM 上販售) 加上你無需為每個用戶另購 License，所以架設伺服器，Linux 遠比其他 OS 便宜，亦成為很多學生及小型企業首選。

1.2 Linux 的特色

免費是 Linux 的一個特點，但自由卻是 Linux 能發展開來的主要原因。作為自由軟體，任何人都可以自由修改或複製 Linux 的 Source Code 給其他人。對於有經驗的工程師來說，他可以合法任意修改 Linux 以符合自己需要。對學生來說，閱讀 Linux 的 Source Code 可以了解操作系統的內部運作及學習高手的編修程式技巧，提高個人能力。而其他人則可以免費或以低成本獲得高手自發

性對系統作出改良的成果。加上 Linux 採用『（Bazaar）』（ 市集 ）式的開發模式，歡迎任何人參與其開發工作及修正的工作，吸引了大量 Hacker 及電腦發燒友使用及寄回自己對系統的改良或研發程式，這使得 Linux 的除錯（Debug）及改版速度奇快，穩定性和效率奇高，並且資源充沛。這也是 Linux 比其他同為自由的 OS（ 如 FreeBSD ）發展得更快更有活力更多人使用的主要原因。

Linux 是一種類似 Unix 的作業系統，與 POSIX 1003.1 的標準相容。它具備了現代作業系統所應有的功能，包括了：

- 真正優先權式多工(preemptive multitasking)，各行程在執行時彼此獨立，不會相互干擾。
- 可供多位使用者同時使用(multi-user access)。
- 支援多重處理器(multi-processor)。
- 可在 x86、Alpha、Sparc、Mips、PPC 等多種不同的平台上執行。
- 虛擬記憶體使用分頁機制寫入磁碟，而非置換整個行程(process)的所有記憶體(後者的效率較差)。
- 依需求載入執行檔，系統只會把執行時要用到的部份載入記憶體。
- 支援多種執行檔格式。
- 支援多種檔案系統，包括 Linux 的 ext2、Windows 系統的 vfat、ntfs、CDROM 的 iso9660、網路檔案系統 NFS 等。
- 支援包括 TCP/IP、PPP、IPX、Appletalk 在內的各種網路協定。
- 多重虛擬主控台(virtual console

第2章 行程和排班

2.1 何謂行程

行程是程式執行時的基本環境，所有程式要進行的活動，會在作業系統幫助下運作。程式是死的，存放在儲存設備中，一般而言就是硬碟。而行程則是核心由硬碟中，將程式載入記憶體中，予以排入執行周期來處理。因此行程是動態的執行狀態，程式則是靜態的儲存狀態。

許多 CPU 除了執行指令外，還會訂定數種優先等級，當 CPU 在不同的優先等級下執行相同的指令會得到不同的結果。 Intel x86 的保護模式定了四種等級，但是 Linux 只用到其中兩種，核心模式(kernel mode)及使用者模式(user mode)。核心模式有較大的權限，可以任意存取週邊設備，通常作業系統核心都是以核心模式在執行，而使用者模式權限小，限制多，應用程式均以使用者模式執行。

當應用程式正常執行時，有下列幾個狀況會切換到核心模式：

1. 程式主動呼叫系統服務函式，例如程式想讀取硬碟裡的檔案，或是程式向系統要求多一點記憶體。

2. 硬體要求 CPU 反應，例如你在鍵盤敲入一個字母，負責鍵盤的晶片要求處理這一筆資料，另一例是電腦內的時鐘每一固定時間會通知 CPU,告訴 CPU 說時間又過了一個 tick(在 x86 上,約 0.01 秒), CPU 就會切換到核心模式，執行與此事件相對應的核心碼。這時有許多要做，其中一項是計算目前這個程式已經執多久了，是否該換手執行讓下一個程式了，如果時間未到，核心事情處理完後會讓原先的程式繼續進行。幾乎每一種可能發生的事件，核心都早已備好對應的程式碼。

3. 程式作了不該作的事，例如讀取不屬於它的記憶體，或直接存取週邊設備，這時 CPU 也會轉到核心模式，讓作業系統核心決定如何處置這個有問題的程式，最普遍的處置就是停止這個程式的執行，把錯誤回報到終端機。

行程在執行時會根據不同的情形改變狀態。 Linux 行程有下列狀態：

1. Running 執行態

行程正在運行(它是系統的目前行程)， 或者是準備運行的(它正在等待被分到系統的 CPU 資源) 。

2. Waiting 等待態

行程正在等一個事件或一個資源。 Linux 中的等待態有性質不同的兩種類型：interruptible (可中斷的)和 uninterruptible (不可中斷的)。可中斷的等待行程能被信號打斷，而不可中斷的等待行程直接等待某種硬體條件， 在任何情形下都不能被中斷。

3. Stoped 停止態

行程被停止了，通常是給一個信號來通知行程的停止。

4. Zombie 僵死態

某個已經終止的行程，由於一些原因，仍然在任務排程中占有資料，就稱做處於僵死態。

Linux 是一個多重處理型的作業系統(multiprocessing, 或叫做多工)。 行程各司其職， 如果某個行程崩潰， 不會導致系統中別的行程崩潰。 每個行程在獨立的虛擬位址空間中運行， 除非通過核心提供的安全的機制之外， 不能和別的行程相互作用。

2.2 行程模型

基本的行程管理，可以分成兩個部分：建立一個新的行程，和執行一個新的程式。要注意到行程和程式是兩種不一樣的東西，因此我們建立了一個新的行程不等於就執行了一個新程式，你可以在父行程下建立一個子行程，子行程仍然繼續執行著原來父行程的程式。相同的，執行一個新程式也不代表一定要建立另一個新的行程，我們執行一個新程式的同時，舊的執式會先被結束掉，然後新的程式會在現有的這個行程的執行背景下（context）繼續執行下去。

要建立一個行程，我們會使用 fork 去呼叫核心來完成。要執行一隻新的程式，則使用 exexve 讓新的程式執行。在這樣的設計下，執行新程式時我們不需要一一的對新行程的環境做設定，直接在現有的行程下執行新程式即可。如果想修改行程的執行環境，也可以先建立一個子行程，然後在子行程內更改好環境，再呼叫 execev 來執行新的程式。

2.2.1 創建行程

當系統啟動時，它在核心模式運行並且有，從某種意義上說，僅僅只存在一個行程，initial process（初始行程）。像所有的行程一樣，初始行程的機器狀態由堆疊，暫存器等等表示。當系統的另外的行程被創建並運行時，這些將會被保存在初始行程的資料結構裡。系統初始化結束時，初始行程啟動一個核心執行緒(叫 init) 然後進入一個等待新行程的空閒循環。當沒有別的事情做時，排程器將運行這個空閒行程。 空閒行程的在工作結構是唯一一個不被動態地被分配的， 當創建核心的同時， 它事實上是被定義在初始的工作結構中。

Init 核心執行緒或行程的行程辨識器為 1， 是系統的第一個真正的行程。它做一些系統初始化設置工作（例如打開系統控制台，安裝根檔案系統）然後運行系統初始化程式。 這個程式是放在 /etc/init、/bin/init 或者 /sbin/init，與你的系統有關。init 程式使用 /etc/inittab 作為腳本檔案（腳本檔案是需要一個直譯器來運行的可執行檔案）來創建系統中的新行程。這些新進的行程可能還要再創建新行程。 例如，當用戶試圖登錄時， getty 行程可能會創建 login 行程。 所有這些行程都是 init 核心執行緒的後代。

新行程通過複製舊行程，或複製目前行程來創建。 一個新任務通過系統呼叫(fork 創建一個新行程或 clone 創建一個新執行緒）來創建。由核心在核心模式下來完成複製行程的工作。在系統呼叫結束時如果排程器安排給新行程執行權， 新行程就可以運行了。 新的工作資料結構在系統實體記憶體中分配出來，而且有一頁或多頁實體記憶體頁被用來作為複製行程的堆疊(用戶堆疊和核心堆疊)。 新的行程辨識器同時也被創建，行程辨識器是唯一而不重被複。但有時後是有必要讓複製出來的行程記住它的父行程。 新的工作資料結構中被加入 task vector（任務向量)， 老行程的工作資料結構的內容被複製到複製的進程的工作資料結構

當複製行程時， Linux 允許兩個行程共享資源而不是各自複製一份。此時我們稱為執行緒，不同的緒可以共用共享同樣的資源。這包括行程的文件， 信號處理程式， 以及虛擬記憶體。 當資源被共享時， 各自的計數域將被增加， 這樣當執行緒全部釋放資源的時候 Linux 才會回收它。

複製行程的虛擬記憶體比較困難。 新的虛擬記憶體區塊資料結構集合要被創建， 還有它們所擁有的 mm_struct 資料結構，以及被複製的行程的頁表。最難的部分是虛擬記憶體的複製工作，因為有的虛擬記憶體在實體記憶體裡，有的

在可執行 CPU 中， 有的在交換檔裡。 為此，Linux 使用稱為 "copy on write" (寫時複製）的技術，它的做法是當其中一個行程試圖寫共享虛擬記憶體時才進行複製。實現的方法是把可寫的記憶體區域在頁表中標為 "read only"（只讀），在 虛擬記憶體區塊資料結構中標為 "copy on write"。 當某個行程試圖寫時，就會發生 page fault， 此時 Linux 就進行記憶體的複製， 並修改頁表和虛擬記憶體的資料結構。

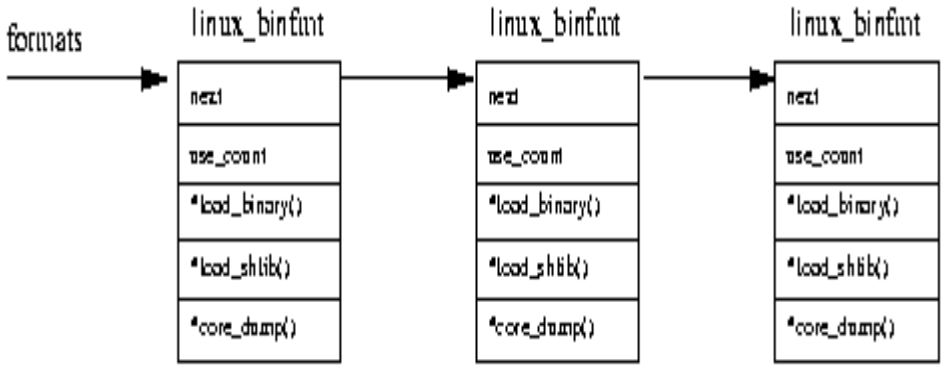
2.2.2 執行程式

像 Unix 系統一樣， Linux 系統中的程式和命令通常是由一個命令直譯器來執行的。 一個命令直譯器是一個用戶行程， 一般被稱為 shell ， 因為它就像是系統的外殼， 被用戶直接感受到。

Linux 系統中有許多命令直譯器， 最流行的一些是 sh， bash 和 tcsh 。除了一些內部命令之外， 例如 cd 和 pwd ， 一個命令就是一個可執行的二進制的檔案。 對每個輸入的命令，命令直譯器會在行程的搜索路徑中所指定的目錄中尋找能夠對應的可執行的呼叫檔案。 搜索路徑由 PATH 環境變數定義。 如果找到了對應的檔案， 它就被裝載執行。

命令直譯器使用上面說的 fork 機制複製自己。 新的子行程用所找到的可執行的二進制可執行檔案的內容替換自己原先的內容，也就是命令直譯器自身。通常命令直譯器等待命令完成， 也就是等待子行程退出。 你能讓命令處理器不要等待，只要把子行程放到背景運行就可以做到。使用 control-Z 組合鍵，它會導致一個 SIGSTOP 信號被送給子行程， 讓它暫停。 然後你可以用 shell 命令 bg 把它放到背景。 命令直譯器向它發送一個 SIGCONT 信號讓它恢復運行，它將一直在哪兒， 直到運行結束或者它需要做終端輸入或輸出。

一個可執行的檔案能有許多格式或甚至是一個腳本檔案。 腳本檔案必須被識別出來並且用適當的直譯器來處理。 例如 /bin/sh 解釋 shell 腳本。 可執行的目標檔案中包含可執行的程式碼和資料， 以及足夠的信息以便作業系統能夠裝載並運行。但是理論上， Linux 靈活到幾乎能處理任何格式的目標檔案。



就像檔案系統，格式由 Linux 支援了的二進制程式碼是在核心創建的時候就被放進入核心內，或者是作為模組的方式被裝載。核心保有支援二進制格式的一張表，在當被嘗試執行一個檔案時，每二進制的格式接著被試用直到一個人工作。

通常被支援了的 Linux 二進制程式碼格式是 a.out 和 ELF 。可執行的檔案並不須要完全被讀進記憶體，而是用我們們所知的需求載入的方式。當載入記憶體的可執行檔案的每部份被行程使用了，檔案的未使用到的部份就可以從記憶體被丟棄。

在行程的環境之中，有三組重要的資訊：行程的辨識資料（process identity）、行程環境區塊（process environment）以及行程執行背景（process context）：

2.2.3 行程的辨識資訊

每個行程有一個程辨識器。 行程辨識器不是任務向量的一個索引， 它就是一個數字而已。 每個行程也有使用者和使用群組辨識器， 它們是用來控制這個行程對系統中的檔案和設備的存取的。ID 的作用在當應用程式使用系統呼叫傳送信號（signal）、修改或等待其他行程時，需要另一個行程的行程 ID 給系統知道。要注意的是行程 ID 是不能更改的，直到行程結束前，此 ID 都用來表示這個行程。

Linux 像所有的 Unix 一樣，有使用者(user)和使用群組(group)辨識器在來檢查行程對系統檔案和設備的存取權限。Linux 系統中的檔案都有所有權和許

可權， 這些許可權描述了系統中的用戶對那個檔案有什麼存取權限。基本的許可權有讀(read)， 寫(write)和執行(execute)，它們被分派到 3 類用戶：檔案的所有人(owner)， 屬於某個特定群組的所有行程，還有系統中的所有行程。每一類用戶可以有不同的許可權，例如：一個檔案可以允許它的所有人讀寫， 允許檔案所在的群組讀並且不允許系統中的其它行程存取。

Linux 系統中，使用者群組就能夠把檔案的權限分配到一組用戶，而不是簡單地到一個使用者或到所有的行程。例如，你可以為一個軟體項目中的所有使用者創建一個群組， 並且只允許這個群組中的使用者能夠讀寫該項目的原始程式。每個行程能屬於若干群組(預設最多能夠屬於 32 個群組)。只要行程所屬的群組中有一個具有存取權限， 這個行程就有權存取那個檔案。

每個行程總共記錄了 4 對使用者和使用群組的辨識器：

1. uid， gid
   行程所代表的使用者（也就是啟動這個行程的使用者）的使用者和使用群組的辨識器。

2. effective uid and gid
   有一些程式在執行的時候會把 uid 和 gid 改變為它們的自己的特定的某個 uid 和 gid，稱為"setuid"程式。它是限制系統服務(service)的權限一個方法， 尤其在實現為別的使用者服務的網路精靈程式等類似的服務時很有用。effective uid 和 gid 來自程式的檔案本身， 和啟動它的使用者無關。核心在檢查權限的時候會使用 effective uid 和 gid。

3. file system uid and gid
   這兩個辨識器通常和 effective uid 和 gid 一樣， 當檢查檔案系統存取權限時會用上。這兩個辨識器是為了建立 NFS(Network File System， 網路檔案系統)而使用的，因為使用者模式的 NFS 伺服器需要像一個特別的行程一樣來存取檔案。在這種情況下，只有 file system uid 和 gid 被改變（有效的 uid 和 gid 不變）。這樣可以防止惡意的用戶向 NFS 伺服器發送 kill 信號。Kill 信號會被以一個特別的有效 uid 和 gid 發送到行程。

4. saved uid and gid
   這是 POSIX 標準中要求的兩個辨識器。 當行程透過系統呼叫來改變 uid 和 gid 的時候必須要用它們來保存真實的 uid 和 gid 。

2.2.4 行程環境區塊

一個行程的環境區塊包括兩個向量組成，一個是參數向量，一個是環境向量。參數向量只是簡單的一列命令列參數，用在啟動程式時。環境向量則是一個”NAME=VALUE”的串列，用來指定環境變數相對應的值。

當產生一個新的行程時，參數向量和環境向量完全不會被更改，新的子行程是繼承自它的父行程而來。如果是執行一個新程式的話，就必須重新設定一個環境區塊。在 execve 時，由行程指定給新程式使用的新環境，新的環境區塊將取代現在的環境變數。

環境變數的機制讓使用者模式下之系統軟體的不同元之曲,可以使用相當有彈性的方法來傳遞資訊。也可以為每一個行程制定特定的作業系統屬性,例如使用的語言或編輯器等。

2.2.5 行程的執行背景（Process Context）

行程執行背景是正在執行中的程式在某一個時間點時的狀態，他會不停的變化的。行程辨識資訊和行程環境區塊是比較不具變動的，從行程開始到結束可以說是完全一樣的（除了我們用選擇性的方式去修改部分的辨識資訊）。

排班執行背景（Scheduling context）

行程的執行順序是由排班器（scheduler）來管理的，在排班執行背景裡提供了讓排班器決定該將行程暫停（suspend）或讓它重新開始執行（restart）的資訊。除止之外，包括行程執行會使用到的暫存器，也被記錄下來。另外還有排班的優先權，和等待被送入行程處理的信號等。

- 記帳（accounting）

用來記錄每一個行程使用到多少的資源，以及每一個行程在整個生命期會使用到多少資源。

- 檔案表（file table）

當使用和檔案有關系統呼叫時，行程就會用到這個。包含行程目前正在使用的所有檔案的訊息。每打開一個檔案，在檔案結構中的一個空閒的檔案指標被用來指向新檔案結構。

■ 檔案系統執行背景（file-system context）

當要開啟一個新的檔案時，目前的根目錄和用來搜尋新檔案所使用的預設目錄都會被記錄在此。

■ 信號處理表（signal-handling table）

信號表裡定義了系統接受外界事件時，所送出的訊息給行程，行程接受到此訊息應處理之常式位址。

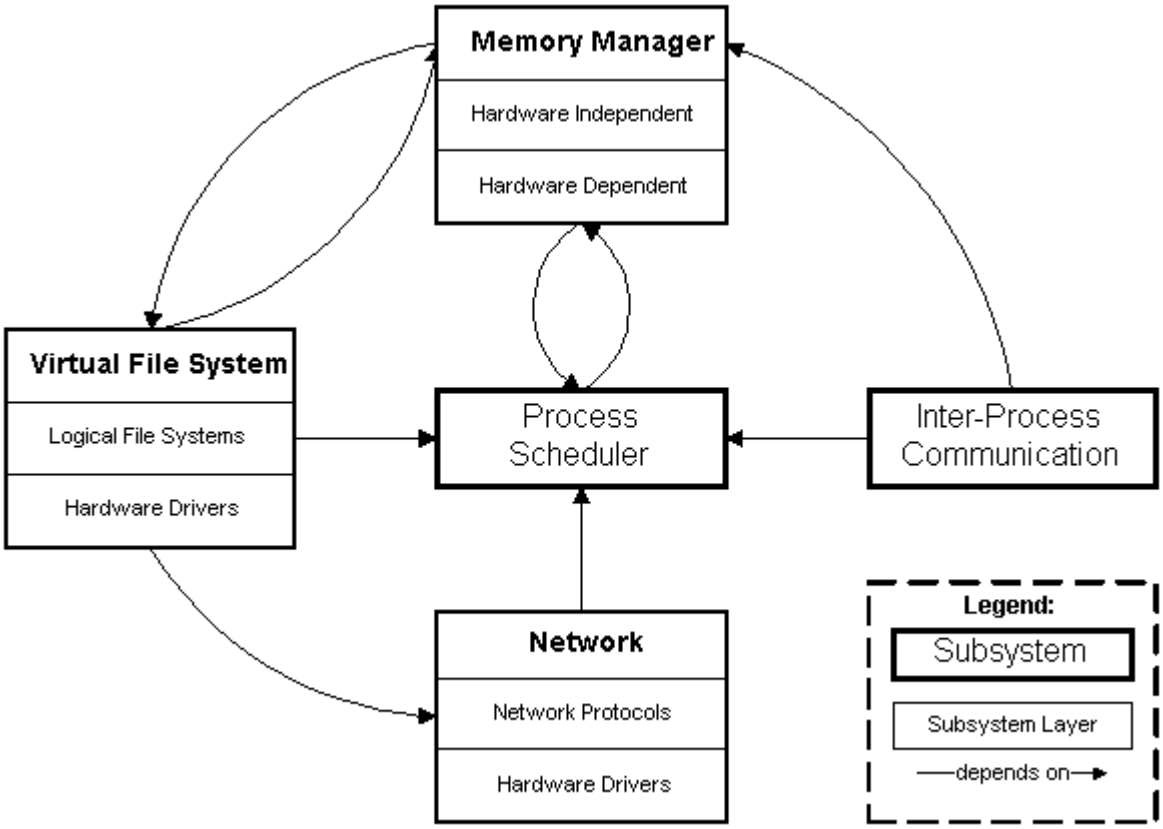
■ 虛擬記憶體執行背景（virtual-memory context）

在虛擬記憶執行背景下，行程所使用的位址空間。

2.3 排班（Scheduling）

行程執行時總是一會兒在使用者模式下， 一會兒在系統模式下。 雖然不同硬體在完成這樣的機制下都有不同的方法,但是都有一種安全機制保証從使用者模式進入系統模式然後再回到使用者模式。使用者模式時行程的權限比較系統模式要小。每當行程進行系統呼叫的時候就會從使用者模式切換到系統模式，然後繼續運行。 進入系統模式之後， 核心程式碼開始執行有關這個行程的服務。在 Linux 系統中， 行程不能從目前正在運行的行程那裡強占執行的權利。 當執行的行程需要等待某個系統事件的時候， 它才會讓出 CPU 。 例如，行程可能等待從一個檔案中讀出一個字元。 這個等待在系統呼叫內部， 處於系統模式；這時，等待事件的的行程將被核心暫停， 其它更著急的行程會被選中來運行。

行程總是要經常做系統呼叫所以就經常會這樣等待。 盡管如此， 如果行程願意， 它還是可以長時間地不做系統呼叫從而不合理地占用 CPU 的處理時間。 因此， Linux 系統要使用先佔式的排程。 在這種情況下， 每個行程被允許運行一小段時間， 比如 200ms， 如果時間到了， 核心就會暫停目前的行程，(不管它是不是願意)， 選擇別的行程來運行。 這一小段時間就是所謂的 time slice（分時)。



核心子系統的總覽

排程在核心部分是負責在系統中所有可以運行的行程中，選擇最該運行的行程。可以運行的行程(runnable process)是指這個行程就在等待 CPU 來執行。

Linux 使用基於優先級的相當簡單的排程演算法在系統在目前的行程之間選擇。 當選擇了新行程來運行時，它保存目前的行程的狀態，包括有特定的處理器暫存器以及其它的環境背景，到這個行程的工作資料結構中。 然後它恢復新行程的狀態(這仍就是處理器相關的)， 把系統的控制交給這個行程， 開始運行它。 排程器為了能夠公平地分配 CPU 時間， 它在每個行程的工作資料結構中保存了下列信息：

■ policy（策略）

在這個行程上使用的排程策略。Linux 行程有兩種類型， 普通和實時。 實時行程比其它所有的行程的優先級都要高。 如果有實時行程可以運行， 它將總是首先運行。 實時進程有 2 種排程策略， Round Robin (循序式)和 First in First out (先入先出式)。在循序式排程下， 每個 runnable 的實時行程循序運行； 在先入先出式排程下， 每個 runnable 的行程依次運行， 次序就是它們進入排程時的順序， 而且不會變化。

■ priority（優先級）
行程的優先級。它也是這個行程被允許運行的時間的總量(以 jiffy 為單位)。 通過系統呼叫和 nice 命令能夠改變行程的優先級。

■ rt_priority（實時優先級）
實時行程的優先級高於其他類型的行程。 這個域允許排程器給每個實時行程以相對的優先級。 實時的行程的優先級可以通過系統呼叫來改變。

■ counter（計數器）
這是該行程被允許運行的時間的總量(以 jiffy 為單位)。 行程第一次開始運行時， 這個值就被設定為優先級的大小， 每次時鐘中斷一次， 這個數值就被減小。

核心內若干地方會運行排程器。 把目前的行程加人到等待隊列後會運行排程器; 在系統呼叫結束，即將返回到使用者模式的時候，也可能會運行。如果系統定時器把目前的行程的 counter 減小到了零， 它也需要運行。排程器運行時， 需要做的事情是：

■ kernel work（核心工作）
排程器運行 bottom half handler（下半部中斷處理)並處理排程器的任務隊列。關於 bottom half handler 以及這些輕量的核心執行緒在 第 11 章 核心機制 中有詳細講解。

■ process current process（處理目前行程）
在選擇其它行程運行之前，必須處理目前行程。如果目前行程的排程策略是 Round Robin（循序式)， 它就被放到運行隊列的末尾，

如果工作是可以被中斷的(interruptible)， 並且自從最後一次排程之後，它會收到了一個信號，那麼就設置它的狀態為 RUNNING(運行)。如果是目前的行程執行超時了，那麼它的狀態也會變為 RUNNING。如果目前的行程就是 RUNNING，它將保持這個狀態。不處於 RUNNING 態並且也不是 interruptible 的行程就被移出運行隊列。 這意味著當排程器要尋找最需要運行的行程時， 不再會考慮它們。

■ Process selection（行程選擇）
排程器尋找整個運行隊列來選擇最需要運行的行程。 如果有實時行程(排程策略是實時的那些)， 它們就會獲得比普通行程更高的權重量。 正常的行程的權重是它的 counter（或者優先級)， 而實時行程是 counter 加

1000。 這說明如果系統中有處於 runnable 狀態的實時行程， 它們就會比普通的 runnable 的行程先執行。 目前的行程， 因為已經執行了一段時間， 經過了若干分時後， 它的 counter 就被減去了一些， 所以如果有同樣優先級的進入的話， 它就要讓位了。 這正是需要的。 如果若干行程有同樣的優先級， 在隊列前面的被先選中。 目前行程會被放到隊列的最後。 在有許多優先級相同的行程的平衡的系統中， 它們會被循序運行。這就是稱為 Round Robin 的排程方案。然而， 行程會等待資源， 它們的運行順序就會發生變化。

■ Swap processes（交換行程）

如果最需要運行的行程不是目前行程， 目前行程就必須被暫停， 新的行程將取而代之。 行程在運行時， 它在使用 CPU 的暫存器和系統的實體記憶體。 呼叫過程時， 它用暫存器傳遞參數， 並且可能需要把返回位址放在堆疊中。 因此， 當排程器運行時它是在目前進程的背景(Context)中。這時， CPU 處於特權態下， 也即核心模式， 但是正在運行的仍然是目前行程。 如果要暫停它， 就必須把它的上下文保存進它的 task_struct 資料結構中。 然後， 新行程的機器狀態的必須被裝載。 這是和具體的系統相關的操作， 各種 CPU 的做法很不相同， 但是通常有一些硬體輔助來做這件事。

行程背景的切換在排程器運行結束時進行。 所切換的背景是與被排程行程有關的硬體環境在此時的一個狀態。

如果剛才的行程或新的目前行程使用虛擬記憶體， 系統的頁表的項目可以需要更新。 同樣地，這是和特定的機器體系結構相關的。 像 Alpha AXP 這樣的處理器， 使用 Look-aside Tables（轉換對照表)或者 cached Page Table Entries（緩衝頁表項)， 必須刷新那些屬於先前行程的表項。

2.4 對稱式多處理器（Symmetric Multiprocessing）

在 Linux 2.0 版的核心就開始做到支援對稱式多處理器（SMP）硬體平台，那就是說， 有能力在系統的 CPU 之間平衡工作。 在排程器中做這種工作是最合適的。然而對核心工作而言，仍要限制其執行時只能有一個處理器，因此在核心中用到了一個空轉鎖（spinlock)。

多處理器系統中， 理想的情況是所有處理器均忙於運行程序。 每當一個 CPU 的目前的程序用盡它的時間片或必須等一個系統資源， 就會單獨運行排程

程式。 關於一個 SMP 統要注意的第一事情是系統中不止存在一個空閒的行程。在單處理器系統中空閒的行程是在任務向量的第一任務，在一個 SMP 系統中每個 CPU 都有一空閒的行程，並且你可能有不止一個空閒的 CPU 。另外每個 CPU 有一個目前行程， 因此 SMP 系統必須追蹤每個處理器上的目前行程和空閒行程。

SMP 系統中每個行程的工作資料結構包含它目前正運行在上面的處理器的編號(processor)以及上次運行在上面的處理器的編號(last_processor)。 雖然行程可以每次在不同的 CPU 上面運行，Linux 可以使用 processor_mask 來限制行程可以使用的 CPU 。 如果 processor_mask 的第 N 位被設置， 這個程序就能在處理器 N 上運行。 當排程器選擇新程序， 它不會選擇 processor_mask 中和目前的處理器對應的位址被清除的行程。排程器會略微照顧上次在這個處理器上面運行的行程， 因為把行程在不同的處理器之間移動通常會帶來一定的性能損失。

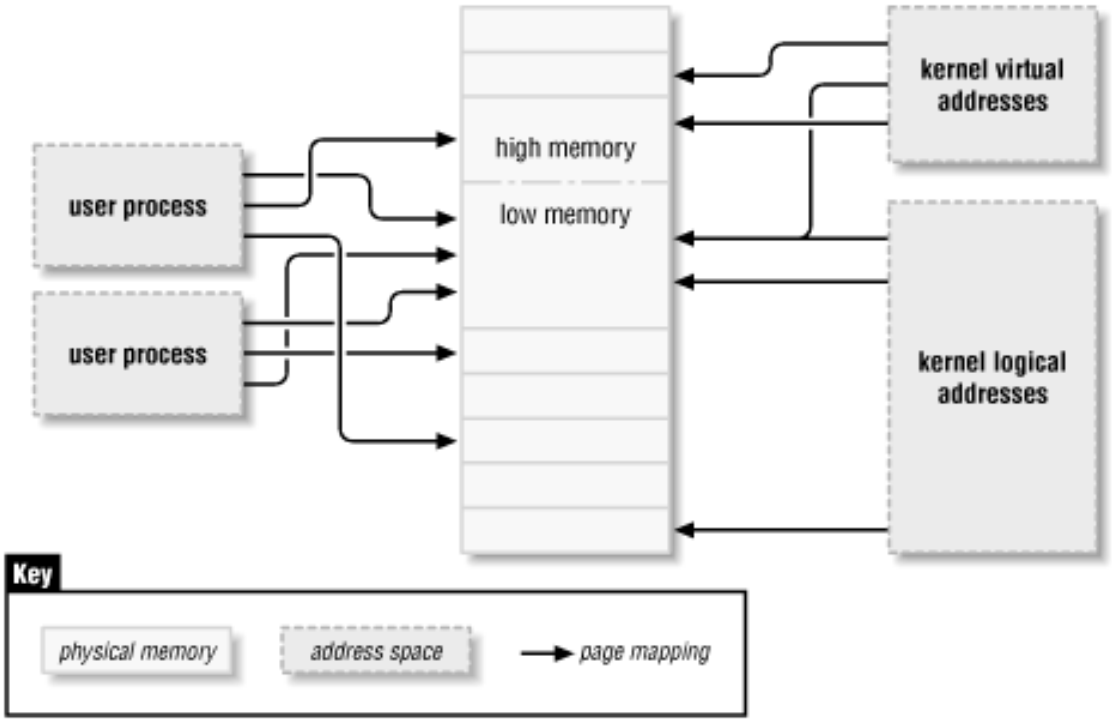
第3章 記憶體管理

3.1 簡介

在討論到記憶體管理時，我們要先認識負責記憶體管理的兩個元件。一個是實體記體管理系統（physical memory-management system），它負責配置和釋放實體的分頁、分頁群組、以及小型的記憶區塊。另一個元件是虛擬記憶體(virtual memory)，用來管理行程中的位址空間與記憶體對映的工作。

3.2 實體記體管理系統（physical memory-management system）

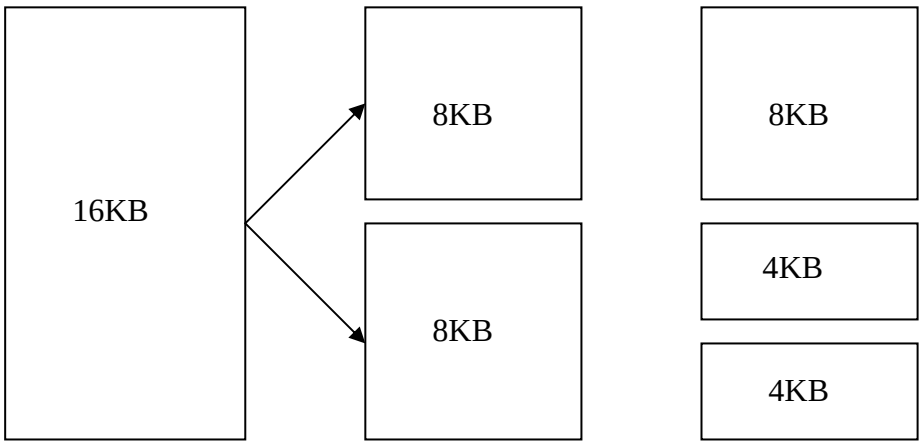
在 Linux 核心，主記憶體的管理是由分頁分配者來完成的。它負責管理實體分頁的配置與釋放，也可以一次配置實體一連續的數個分頁。它採用的是對偶式的堆積演算法來追踪管理可便用的實體分頁。



Linux 的位址形態

所謂的對偶式堆積配置演算法是將記憶體相鄰的兩個配置單元編成一對，每

一個可配置的記憶體區域，都有一個相鄰的夥伴。當我們需要一塊記憶區塊時，例如我們要的是 4K，但最小的區塊大小是 16K，這個 16 K 就會先被切成對等的兩個 8K 的區塊，這兩塊也就是我們說的成對的區塊，8K 又會繼續被切割下去，成為更小的兩個 4K 的區塊，此時才符合我們要使用的大小。如果成對的兩個區塊發現剛好都未被使用中，那麼這兩個區塊就會被合併成較大的區塊，此較大的區塊的夥伴如果仍然是未使用中的，那麼他們又會再結合成更大的區塊，未使用的區塊會一直向上結合，直到夥伴有被使用為止。



3.3 虛擬記憶體（virtual memory）

Linux 支援虛擬記憶體，虛擬記憶體是一種利用硬碟來解決實體記憶體不足的問題，核心會將現在不使用的記憶體區塊內的資料寫入到硬碟，以至將空出來的記憶體可以挪為它用，當原本不使用的資料又再度需要被讀取時，它們就從硬碟被讀取到主記憶體裏。對使用者而言不需要去關心主記憶體的容量不足的問題，所有的記憶體管理都由系統去負責。

Linux 需要在硬碟預先配置一塊分割區，用來做為虛擬記憶所使用的。這個與我們在 windows 的做法有不同，在 window 不用特別獨立去在硬碟上分割虛擬空間的大小，而 Linux 則要在硬碟分割一塊置換空間給虛擬記憶體去使用。

swap 分割區是比較快的，但是 swap 檔是比較容易改變大小的(沒有必要重新分割整個硬碟，或者可能一切重頭開始)，當您清楚知道您需要多少置換空間，

您應該分割一個 swap 分割區,但是如果您仍然不確定的話,您可以先使用 swap 檔,使用系統一陣子看看,好讓您有個短暫時間瞭解到底您需要多少置換空間,直到您完全肯定使用多少置換空間時,然後您在分割一個 swap 分割區。
虛擬記憶體不僅使電腦的記憶體看起來更多,還提供以下的功能:

擴大位址空間作業系統擴大了系統的記憶體空間。

- 記憶體保護系統中每個程序都有它自己的虛擬位址空間。
- 記憶體映射記憶體映射將檔案內容連接到虛擬位址中。
- 記憶體管理子系統公平地分配記憶體給正在運行的各程序。

在虛擬記憶體系統中,所有位址都是虛擬位址而非實體位址。處理器根據作業系統中的一組表格而把這些虛擬位址翻譯成相應的實體位址。為了方便做虛擬記憶體與實體記憶體的翻譯,虛擬記憶體和實體記憶體被劃分成許多適當大小的塊,叫做“頁”(page)。記憶區塊是以頁為單位提供給程式使用。在 Linux 上的所有分頁都是一樣大小,以方便記憶體管理。

3.4 需求分頁(demand paging)

因為虛擬的記憶體比實體記憶體大很多,所以我們不可能把所有的資料全部讀入實體記憶裡,因此為了節省記憶體,我們只能把有使用的程式碼和資料讀入,這就是按需求分頁。但是因為在實體記憶體裡的資料只是一部分的,所以當我們要求的分頁不在實體記憶中時,就發生了 page faulting 的狀況,此時我們就必須要回到磁碟中去把我們要的分頁載入。

此時若沒有其他的行程在執行(一般來說會有其他的行程同時在運作),系統就會等待磁碟中的分頁被載入到實體記憶體中之後,再從 page faulting 的地方繼續做下去。然而磁碟讀取的速度遠比記憶體讀取的速度慢很多,如果不停的發生 page faulting 的狀況,導致系統一直要做磁碟讀取的動作,那麼執行速度就會被拖慢下來。

3.5 頁交換

當行程要載入一分頁進實體記憶體時,如果在實體記憶體裡得不到空位置,

作業系統必須從記憶體中丟棄別的頁，為這頁提供空間。如果從記憶體中被丟棄的那頁是從記憶體映對或資料檔案中來的，並且記憶體映對和資料檔案沒被修改過，那這頁不需再被保存，可以直接丟掉。如果程序再需要那頁，它可以重新被從記憶體映對或資料檔案中讀入記憶體。

但如果該頁已被修改了，作業系統必須保存這頁的內容以便它以後能再被存取。當它們被從記憶體中移出時，它們被作為特殊的交換檔（swap file）保存。但相對於實體記憶的讀寫速度而言，讀取硬碟的速度顯然慢了很多，因此作業系統必須衡量那些頁要保留在記憶體以備使用，那些要放回磁碟去。

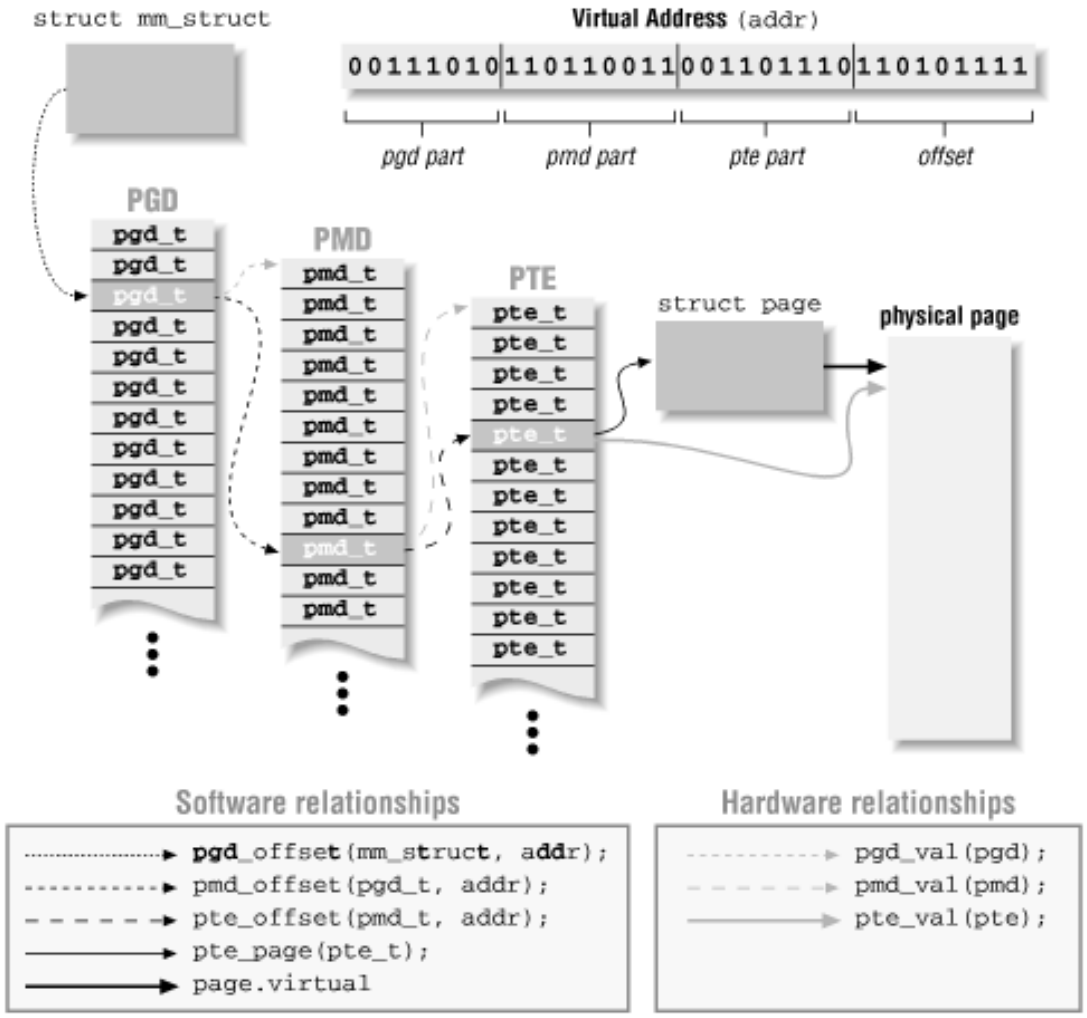
如果安排的不好，就會發生某一頁剛被放回硬碟，一下子又被取回記憶體裡，處理器一直在做這交換的動作，自然效率不會高，因此一靠好的交換法則是很重要的。Linux 使用最近不常使用演算法(Least Recently Used) 來選擇從記憶體中被丟棄的頁。每一個分頁都有一個年齡（age），在每次使用過後，他的年齡值就會被降為零，當要選擇踢出分頁時，年齡老的會先被踢出。

3.6 虛擬記憶體的共享

虛擬記憶體允許個行程有自己虛擬位址空間，行程的記憶體存取是透過頁表，每一個行程也都有自己的頁表，當多個行程共享記憶體中的一頁時，實體的頁號就會同時在多個行程的頁表中出現，因此透過虛擬記憶體，行程之間更容易做到記憶體共享了。

3.7 頁表的存取的控制

頁表中也同時記錄了存取控制的訊息，如果一個分頁是可執行的程式碼時，你會希望對這個區塊做存取的限制，限制他只能讀取，以避免在他的可執行程式碼上面寫資料。相反的，如果這個分頁是屬於資料形態的，我們就會設定他為可讀寫。這樣的控制可以避免我們因為不小心而寫入了可執行碼。另外對可執行碼，我們也可以用核心程式和使用者程式來做區分，這樣子便用者程式不能隨便去執行核心的程式，核心程式的資料結構也不會被不是核心程式的程式碼所存取。



Linux 頁表的三階層

Linux 頁表有 3 層。每一層負責保存下一層頁表所在的頁號。在虛擬位址上記錄了很多值；每個值都是記錄在某一層頁表中的偏移量。要把一個虛擬位址翻譯成實體位址時，處理器取得每一層的值把它變成頁表中的偏移量，而讀出下層頁表的所在頁號。這樣重復 3 次直到找到包含虛擬位址的實體頁號。虛擬位址的最後一個值，叫做位元偏移量，被用來在實體頁內找到所需資料。

每個運行 Linux 的平台必須提供翻譯巨集(Translation macros) 以便核心可以檢索頁表，完成某種操作。這樣，核心不需要知道各平台上頁表記錄的具體格式和它們是怎麼被安排的。

這就是為什麼 Alpha 有 3 層頁表，Intel x86 處理器只有 2 層頁表，但是 Linux 仍可以用 Alpha 處理器和 Intel x 86 處理器一樣的頁表操作代碼，。

3.8 頁的分配與回收

系統對頁的操作，最常用的到的就是當一段分頁被使用到，而需要裝載入記憶時，必須對頁做配製的動做，另一個就是，當記憶體中的分頁不再使用後，要將其卸載,並釋放掉。頁的分配和回收機制是維持虛擬記憶體分系統效率的關鍵。

系統中所有實體記憶體頁的資料結構中記載著下列一些重要的資訊：

■計數器

記錄這個分頁被多少使用者所使用，當此數值大於 1，表示這個分頁被多個使用者所共用。

■年齡

描述頁的年齡，被用來決定頁是否是被丟棄或交換的候選條件。當分頁被使用到時，頁的年齡就被回歸到零，也就是最年輕的。長期不使用的頁年齡就變老，在決定要丟棄或交換時，老的頁（不常使用到的）就會被優先丟棄。

■map_nr

描述這個 mem_map_t 對應的頁的實體頁號。

如我們之前所介紹的，在 Linux 是以對偶式的堆積演算法來分配和回收頁，因此我們記錄的每個單元頁塊的訊息，第一單元是從 1 個頁塊，第二個單元 2 個頁塊，第三個單元則要描述到 4 個頁塊，往上以 2 的倍數遞增。

系統在找尋空頁時，會從所需頁的大小開始找起，往上找比他大的頁，如果正好沒有空頁可以配置，則系統會再往上尋找更大的（兩倍大的），直到找到全部找過或有找到可以配置的分頁時停止，有找到適當的分頁，就會將程式碼或資料載入此分頁，如果找到的空頁塊比所需的大，它必須被分割成正確的大小。；如果全部找完仍找不到可用的分頁,那麼系統就需要把已存在的分頁移出就給新的分頁用。

頁分配時容易將大塊連續的記憶體分成很多小塊。頁的回收程式碼須盡可能將小塊的空記憶體重新組合成大塊的。事實上，頁塊的大小對記憶體的重新組合很重要。

當一頁塊被釋放時，系統會檢查和他相鄰同大小的夥伴頁，看它們是否是空的。如果是，它們將被拼成一個大的整塊。每次當兩塊記憶體被拼成了更大的空塊時，回收程式還會再次檢查合併後的新頁，看他的夥伴頁是否也正好是空的；嘗試將它們與夥伴頁組合，以得到更大的空間，回收程式會一直往上結合到夥伴

頁不在是空的為止，這樣得到的空頁塊可以滿足任何對記憶體的需求。

3.9 頁的快取（Cache）

Linux 就使用了很多與記憶體管理有關的快取技術，為了是提高處理器和記憶的效能，將重要的資訊留在快取裡，這樣就能更快取得所需要的資料了：

■緩衝區（Buffer Cache）

緩衝區包含區塊設備驅動程式（block device driver）使用的資料緩衝區。

■頁快取（Page Cache）

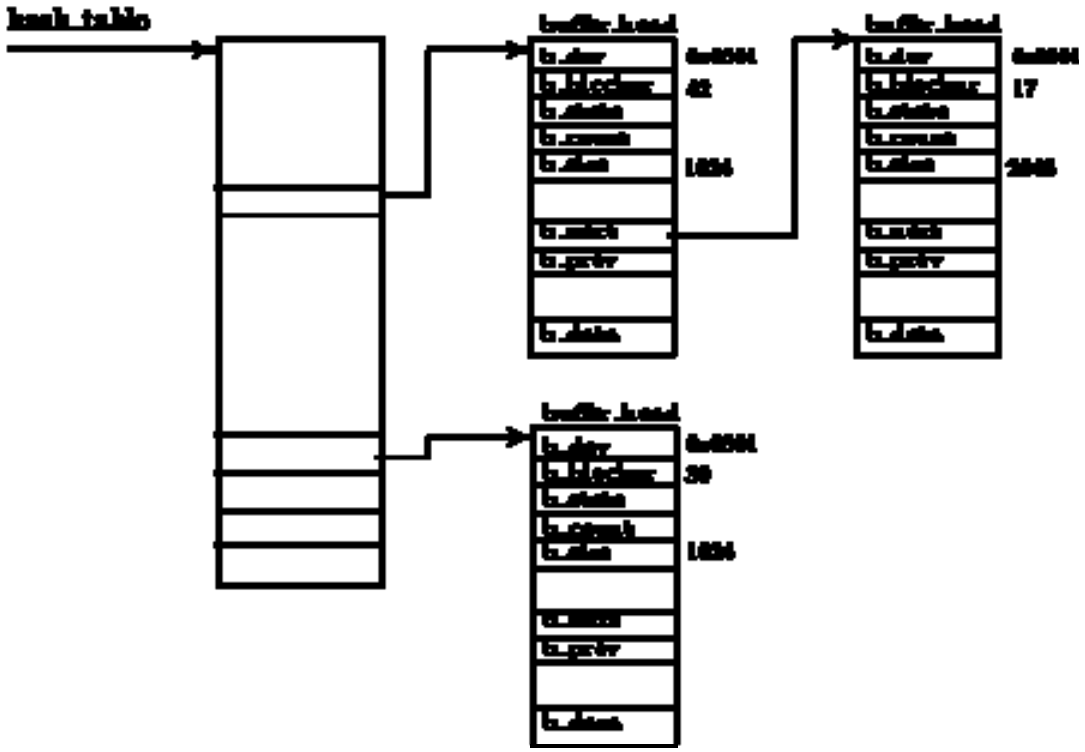
它被用來加快磁碟上資料的存取。

■交換快取（Swap Cache）

只有修改了的頁，被保存在交換檔中。

■硬體快取（Hardware Caches）

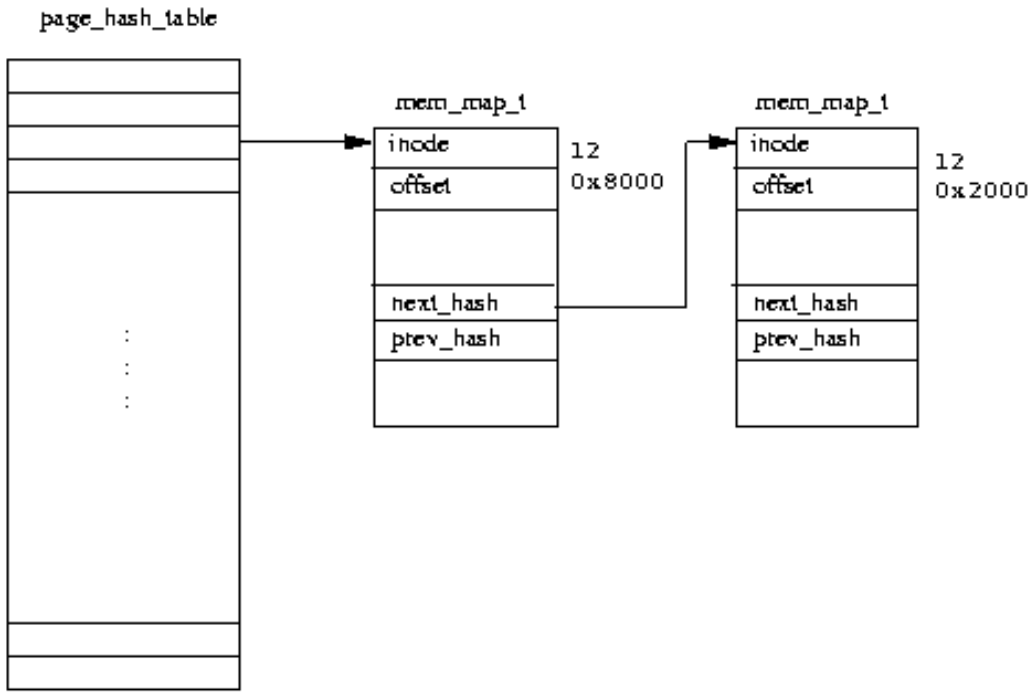
其實對中央處理器而言，並不會直接去讀取頁表的資料，通常是讀取保留在快取上所記錄的頁表資料，這些稱作 Translation Look-aside Buffer，保存了系統中多個程序頁表的拷貝。



緩衝快取

Linux 的頁快取可以每次自磁碟中讀取一頁放在頁快取裡，來加快從磁碟機中讀取到主記憶體的速度。在從磁碟中讀入檔案前，會先去頁快取裡看是不是已經存在此分頁，如果頁快取裡有，那就直接讀到主記憶體裡，如果在頁快取找不到，才到磁碟去讀入。

一般來說檔案讀取都是一頁接著一頁的，因此Linux 會預先把下一頁讀到快取裡，當須要下一頁時，該頁就已經在記憶體裡等待了。隨著檔案讀取的愈來愈多，快取的分頁也會變多，此時不再被使用的分頁就會被移出。Linux 也會視記憶體的不足，減少頁快取的大小。



當將頁移入交換檔中時，並非所有情況，Linux 都需進行寫操作。有時一頁既在交換檔中，又在記憶體中。這種情況是由於這頁本來被移到了交換檔中，後又因為被調用，重又被讀入記憶體。只要在記憶體中的頁沒被修改過, 在交換檔中的拷貝仍然是有效。

Linux 使用交換快取來記錄這些頁。交換快取是一張頁表記錄的表，每條記錄對應一頁。每條頁表記錄描述被換出的頁在哪個交換檔中及其在檔案中的位置。如果一交換快取記錄非零，表示在交換檔中的那頁沒被修改過，如果頁被修改了(被寫)，它的記錄將被從交換快取中移出。

當 Linux 需要移出一頁記憶體到交換檔中時,它先查詢交換快取, 如果這頁有一個有效的記錄，它就不需要把頁寫到交換檔中了。因為自從它上次被從交換檔中讀出後，在記憶體中沒被修改過。

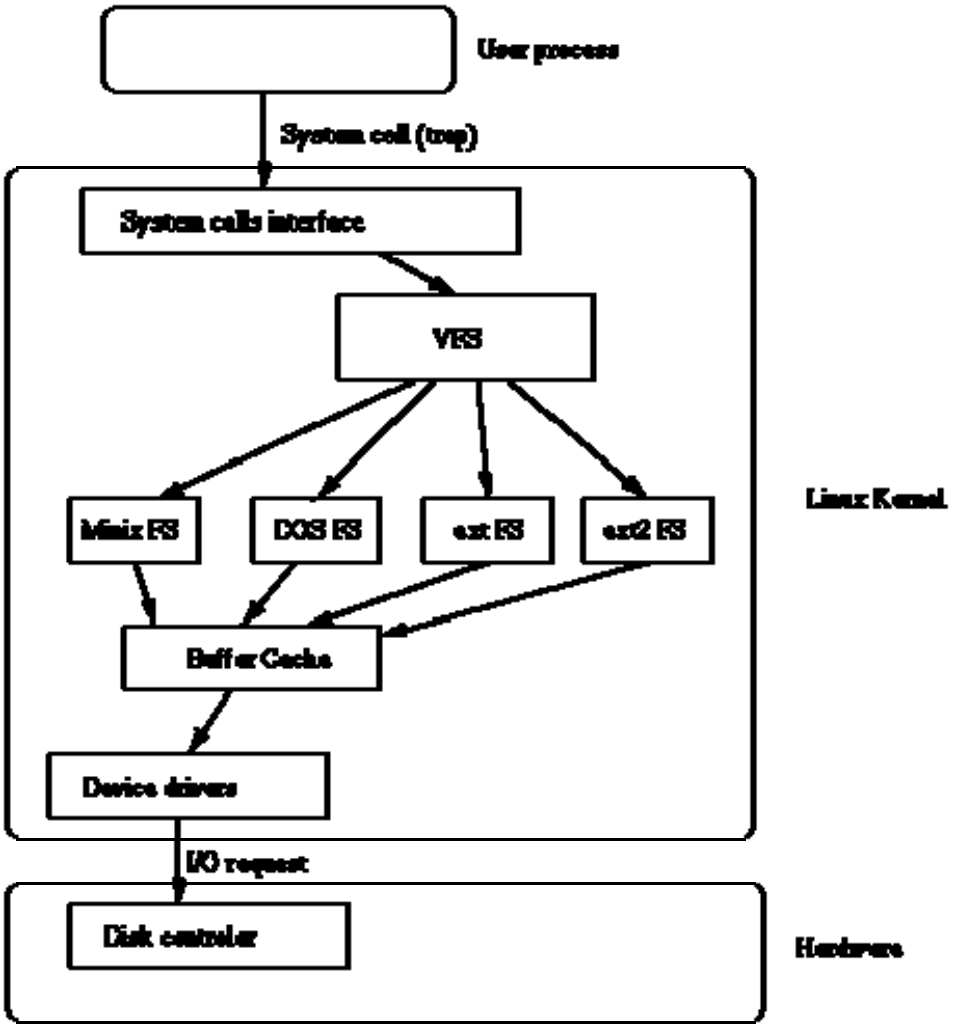
交換快取中的記錄描述已被移到交換檔中的頁。它們被標為無效，但是告許 Linux 頁在哪個交換檔以及在交換檔的哪一頁。

第4章 檔案系統

4.1 Linux 的虛擬檔案系統(virtual file system)

Linux 可以支援數十種檔案系統，如 ext、ext2d、minix、vfat、msdos…等。Linux 之所以能夠做到同時的對不同檔案系統的支援，靠的就是 Linux 系統裡使用虛擬檔案系統(virtual file system)的介面層來將應用程式和實際的檔案系統分隔開來，不同的檔案系統提供給虛擬檔案系統層相同的軟體介面，那麼對 Linux 核心而言，看起來就都是一樣的。

Linux 檔案系統其實可以分為三個部分， 第一部分叫 Virtual File System Switch， 簡稱 VFS。 這是 Linux 檔案系統對外的介面。 任何要使用檔案系統的程式都必須經由這層介面來使用它。 另外二部分是屬于檔案系統的內部。 其中一個是 cache，另一個就是真正最底層的檔案系統， 像 Ext2，VFAT 之類的東西。



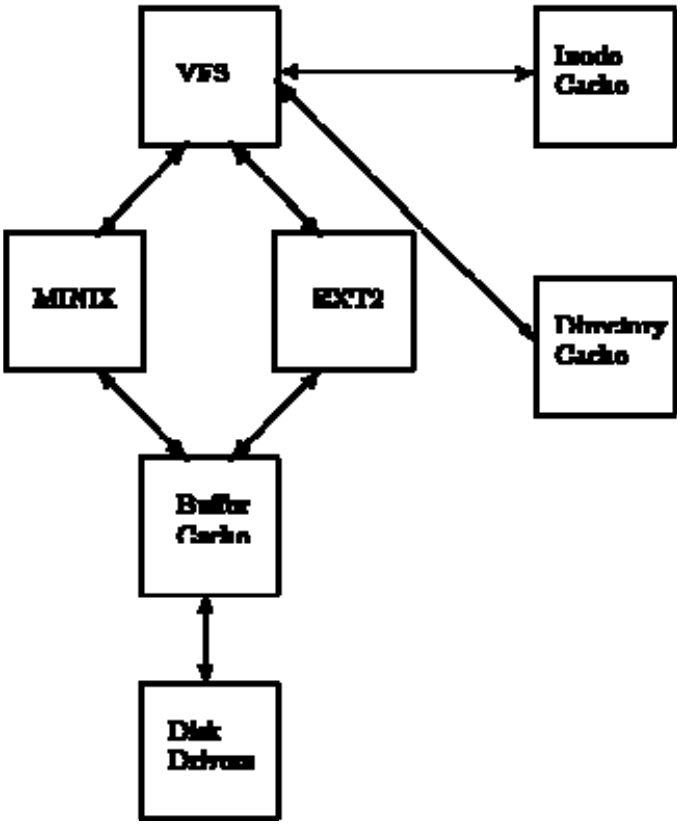
使用者執行呼叫檔案的程序

很重要的觀念是，對 Linux 而言，什麼東西都可以是檔案系統；一個硬碟、一個硬碟分割區、一個平行埠，甚至是網路上的一顆硬碟，只要能夠提供給虛擬層相同的介面，應用程式就不需要去知道現在用的檔案系統是什麼格式的，完全的交由中間的 VFS 層的趨動程式去負責溝通。

當 Kernel 要使用檔案系統時，都是經由 VFS 這層介面來使用。不論是使用者或程式設計師去讀取一個檔案的內容時，它不會因為這個檔案位于不同的檔案系統就需要使用不同的方式來讀取。因為中間的 VFS 內部的函式幫我們處理了。當我們要讀寫的是 FAT 檔案系統時，VFS 會去呼叫 FAT 的函式來處理，如果我們要讀取的是 CD-ROM，那麼 VFS 會去呼叫 iso9660 來處理。

另外還有一層像 PC 的 cache 一樣用來加快速度，在 Linux 檔案系統其實也是

有一個 Cache 的機制以加快速度，稱為 Buffer Cache。底層的檔案都要先經過 Buffer Cache 的讀取。如果資料在 Buffer Cache 裡有的話，就直接讀取，如果沒有的話，才透過 Buffer Cache 要求 driver 去讀寫。除了 Buffer Cache 之外，其實， Linux 檔案系統裡還有一個 Cache，叫 Directory Cache。這是針對使用者下 ls 命令用的，因為每次的 ls 或讀寫檔案都要對目錄的內容做 search。因此在目錄這方面做個 Cache 的話，可以將整個系統的速度往上提升。

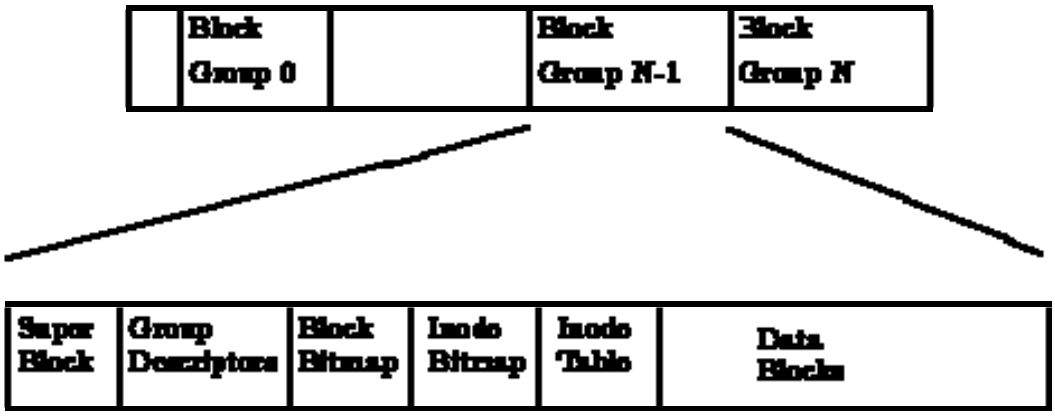


VFS 的邏輯圖

因為 Linux 的檔案系統可以是不同的型式，因此不能像 Windows 或 DOS 一樣給予每個磁碟機一個數字或命名來存取，它們是以單一的樹狀結構目錄來連結到檔案系統的實體。任何一個檔案系統可以被連結在檔案系統樹的任何位置，被安裝的目錄上原本的存在的內容則會被覆蓋掉，直到這檔案系統被卸除後，才可以看得見此目錄原本的檔案。這個動作被稱為掛載（mount），就像一個新的檔案系統被加入到已存在的檔案系的子目錄裡

4.2 第二擴充檔案系統 (Second Extended File System)

ext2 是目前廣泛的在 Linux 所使用的系統，由 Remy Card 所設計，比起 ext 檔案系統來說，ext2 具有向上相容的特性，因此新的檔案系統版本程式碼無須重新製作現存的檔案系統。



ext2 實體層的呼叫

1. 區塊（Block）

對 ext2 檔案系統而言，和其他檔案系統一樣，硬碟是被分割為若干的相同大小的區塊（Block），區塊的大小一般為 1024 bytes 或 4096 bytes。每一個檔案的大小也被調成區塊的整數倍，也就是說對一個檔案系統的區塊如果是 1024 bytes，那麼 1025 bytes 的檔案大小，勢必占用掉兩個區塊。我們可以在創建 ext2 檔系統的時候被決定區塊的大小，也可以由系統管理員指定，或由檔系統的創建程式根據硬碟分區的大小，自動選擇一個較適當的值。檔案系統中，有一部分的區塊是用來存放一些檔案系統的結構資料。這些區塊被聚在一起分成幾個大的 區塊組（Block group）。每個區塊組中有多少個區塊是固定的。另外有個 blocks bitmap 用來記錄每個 Block 是否被使用了。

2. Super block

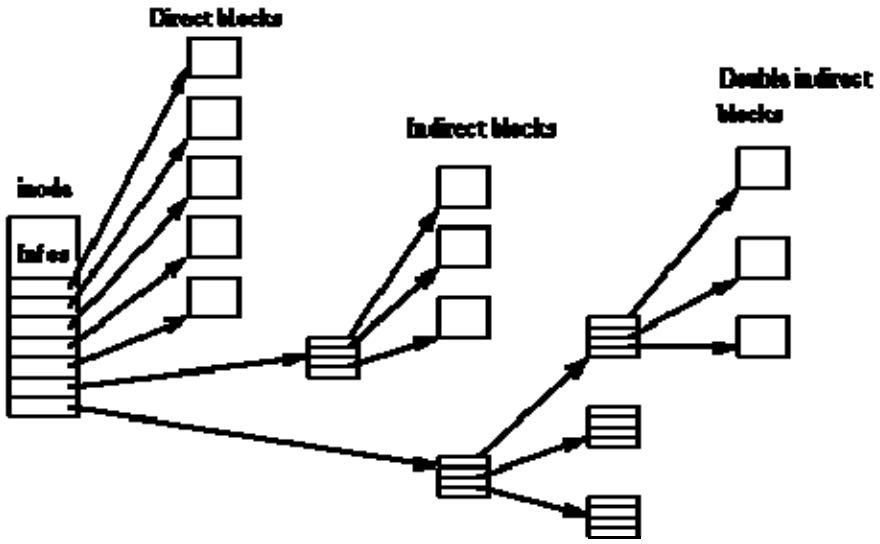
是指硬碟分區的第 1024byte 算起的一部分資料，在此記錄了檔案系統的基本大小和其形狀的描述。當檔案系統被安裝時，通常僅僅 Super block 被讀進記憶體。

3. 區塊組描述器（group descriptor）

每一個區塊組描述器都相對應一個區塊組，區塊組描述器被放在 Super Block 的後面，其內包含了三種檔案的資訊：Blocks Bitmap、Inode Bitmap、Inode Table。

4. Inode

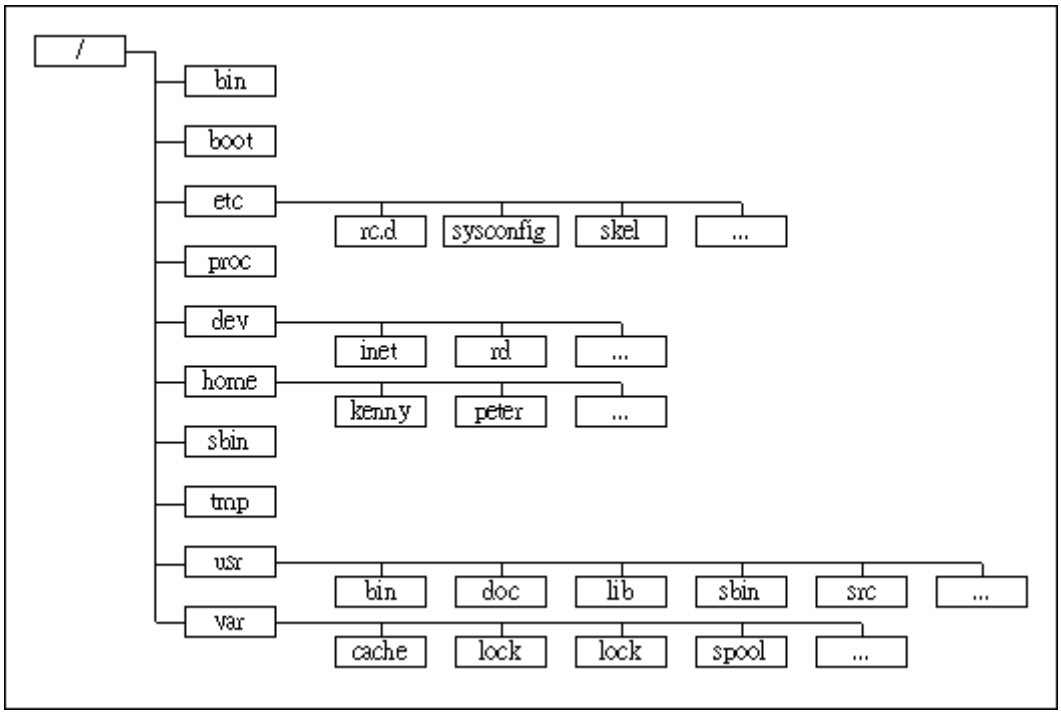
一個 inode 描述一個檔案中的資料占據哪些塊，檔案的修正時間，存取權利和文件類型等等。Ext2 檔案系統中，每一個 inode 對應到一個檔案，每個 inode 有一個唯一的數字標識。所有的 inodes 被集中放到一個 inode table 中。另外還有一個 inode bitmap 對映到每一個 inode 有沒有被使用了。



Inode 的結構

4.3 檔案系統結構

在 Linux 中所有的檔案或目錄都掛在根目錄（Root）下，根目錄寫作/。根目錄下是各檔案系統的子目錄，而子目錄下面可以再接各檔案系統的子目錄和檔案。因此 Linux 的檔案系統結構看起來就像是一個樹狀的分枝，稱為階層式（hierarchy）的架構。



儘管子目錄的名稱可以自行訂定，但是對於一些常用的目錄仍有一些習慣的命名，我們最好也能遵循相同的命名規則。以下是常用的目錄名稱：

目錄名稱	檔案類形
/root	系統開機核心所需檔案
/etc	系統設定檔案
/lib	函式庫檔案
/dev	設備檔案
/var	系統資訊/設定檔；具變動性質的相關程式目錄
/bin	使用者一般執行檔
/sbin	使用者系統執行檔
/usr	普通用戶的程式目錄
/home	使用者家目錄所在
/tmp	暫存檔存放目錄
/mnt	檔案系統掛入點

目前有一套規範檔案目錄的命名及存放標準 File System Standard（ FSSTND），大部分的 Linux 系統也都遵守此一標準。

4.4 檔案系統掛載 （Mount）和卸載（Umount）

因為所有的檔案系統是連接在根目錄之下，形成一個單一的樹狀結構。因此對於新加入的檔案系統，我們要進行掛載的動作。
掛載的指令範例如下：

```
$ mount /dev/hda2 /home
$ mount /dev/hda3 /usr
```

mount 使用到了兩個參數，第一個是對應該磁碟或分割區的設備檔包含檔案系統，第二個是掛載在那一個目錄。在上例中我們把/dev/hda2 掛載在/home 的子目錄下，把/dev/hda3 掛載在/usr 的子目錄下。對於我們掛戴點上如果不是空目錄，則在掛戴後，原本目錄的資料會被隱藏起來，直到我們把檔案給卸戴後，才能再次看到原本的資料。
當某個檔案系統不再需要被掛載時，它可以使用 umount 來卸戴。

```
$ umount /dev/hda2
$ umount /usr
```

umount 只有一個參數，不是設備檔就是掛戴點。

4.5 常用的基本指令

4.5.1 ls 顯示檔案資訊

ls 是用來列印所有檔案及目錄資訊的指令，我們只要在命令列上直接輸入 ls 就可以查詢目前目錄下的所有檔案和目錄，例如：

```
tau@PC44 /
$ ls
bin  cygwin.bat  cygwin.ico  etc  lib  sbin  tmp  usr  var
```

除了基本的查詢之外，如果我們希望得到更多資訊，可以加用-l 的參數：

```
tau@PC44 /
```

```
$ ls -l
total 250
drwxrwxrwx+   3 65535    unknown    192512 Dec  4 18:48 bin
-rwxrwxrwx    1 65535    unknown        57 Dec  3 15:26 cygwin.bat
-rwxrwxrwx    1 65535    unknown       766 Dec  4 18:43 cygwin.ico
drwxrwxrwx+  17 65535    unknown     12288 Dec  4 18:48 etc
drwxrwxrwx+  22 65535    unknown     45056 Dec  4 18:48 lib
drwxrwxrwx+   2 65535    unknown         0 Dec  4 16:12 sbin
drwxrwxrwx+   2 65535    unknown         0 Dec  4 18:48 tmp
drwxrwxrwx+  21 65535    unknown      4096 Dec  4 16:12 usr
drwxrwxrwx+   8 65535    unknown         0 Dec  4 18:43 var
```

在完整列表下可以得到的資訊有：

-	rw-	r--	r--	1	root	root	103	Oct 27 22:09	test.ls
檔案性質	User 權限	Group 權限	Others 權限	Link 數目	User 名稱	Group 權限	檔案大小	最後一次修改日期	檔案名稱

- 檔案屬性：

第一個如果是‘-’的話，表示這是一個常規檔案，如果是‘d’的話表示這是一個目錄，如果是‘l’的話，表示這是一個 link。
User、group、others 權限：
接下來的九個字母分別是代表 user、group、others 的使用權限，有關於使用權限的內容和設定，我們留到安全管理的部分再做詳細的介紹。

- Link 數目：

指目前 link 的數目，目前只有一個 link，表示他是只有一個 hard link。
User、group 名稱：
這裡記載著他是屬於那個使用者，以及他是那一個群組的。只有系統管理員才能建立和刪除使用者和群組。

- 檔案大小、修改日期和檔名：

檔案的大小是以 1byte 為單位，如果是目錄，則固定是 1024，這與目錄能放多少東西無關，1024 只是存放一些系統的資料。修改日期對文字檔而言，就是你上一次存檔的時間；對執行檔而言，如果不是你自己寫的程式，那記錄的就是你複製過來的時間。在 Linux 系統中，檔案名稱是具有彈性的，長度不限，也可以用特殊符號。

如果想要看到隱藏檔則可使用-a 的參數，在 Linux 中如果以句號當開頭的檔名，就會被當作是隱藏檔。當然你也可以用組合命令 ls -la 把所有檔案的完整資訊列印出來。

4.5.2 df 檢視檔案系統

因為在 Linux 系統中，可以使用不同的檔案系統，因此有時如果你想知道你在什麼檔案系統下，以及這些檔案系統的詳細資料，你可以使用 df 指令。

```
$ df
tau@PC44 /
$ df
Filesystem           1k-blocks      Used Available Use% Mounted on
C:\cygwin\usr\X11R6\lib\X11\fonts
                      10241404   9532460    708944  94%
/usr/X11R6/lib/X11/fonts

C:\cygwin\bin         10241404   9532460    708944  94% /usr/bin
C:\cygwin\lib         10241404   9532460    708944  94% /usr/lib
C:\cygwin             10241404   9532460    708944  94% /
c:                    10241404   9532460    708944  94% /cygdrive/c
d:                    10361924   8827256   1534668  86% /cygdrive/d
```

你可以看到磁碟機所掛載的位置，以及他的使用率。通常裝置的容量是以 1k bytes 為單位。

/dev/hda2	497699	71252	400743	15%	/
裝置名稱	裝置容量	使用空間	可用空間	使用率	掛載點

du 查看目錄使用狀況

你可以用 du 的指令來查看現在目錄下和他子目錄的的使用狀況。

```
tau@PC44 /usr/man/sv
$ du
11      ./man1
19      ./man5
61      ./man8
91      .
```

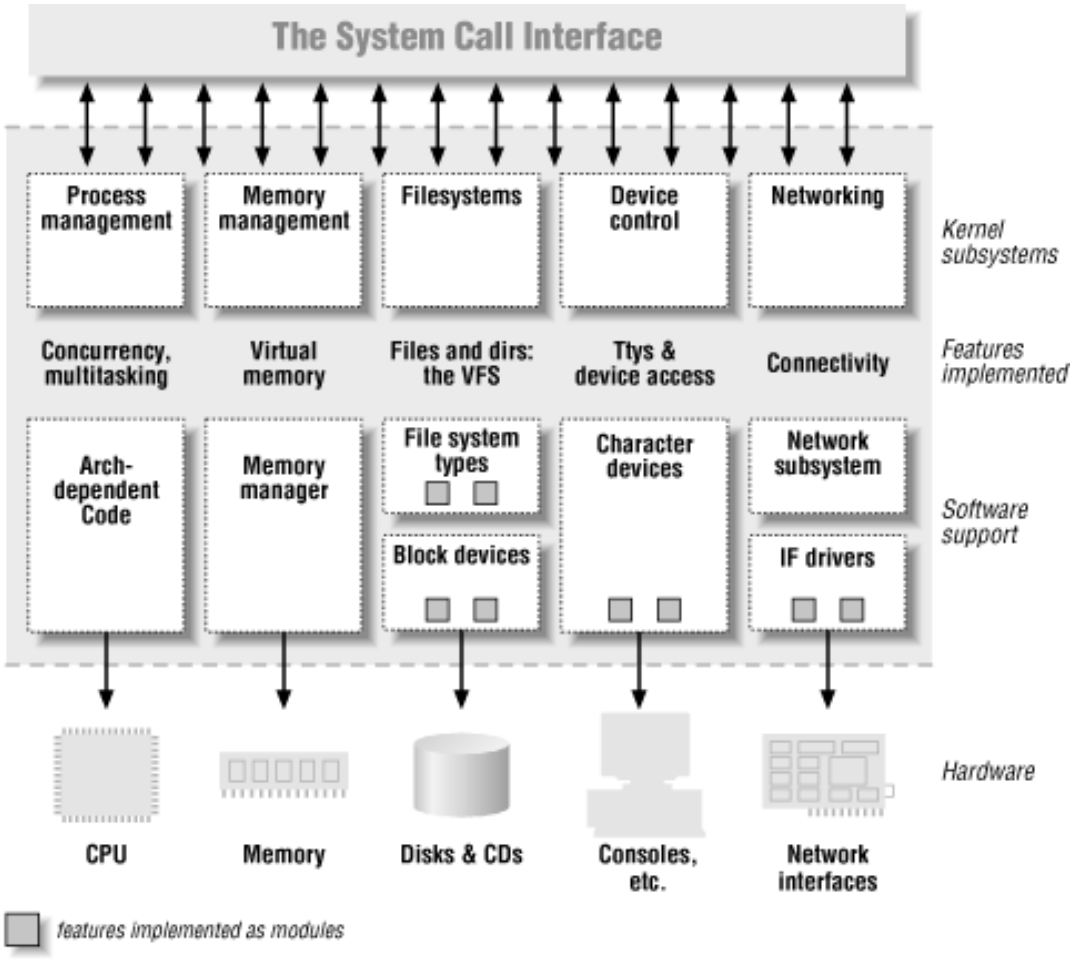
前面的數字表示此目錄用了多少的容量，以 1k bytes 為單位。最後一行會顯示此目錄究竟使用了多少單位。

第5章 週邊設備

5.1 簡介

對使用者而言，在 Linux 下看所有的硬體驅動程式，就像看是一般的檔案一般，使用者可以像一般檔案一樣的開啟驅動程式的之間的存取通道。系統管理者也可以在檔案系統裡做一個特別檔案，來映對到某個裝置的驅動程式上。所有的實體設備被當做正規的檔案來處理，可以被“開啟”，“關閉”，“讀”和”寫“，甚至給予每個裝置設定存取權限，來控制誰可以存取此裝置，就像我們用系統呼叫處理檔案一樣。

每一個實體設備都需要透過設備控制和狀態暫存器(CSR)，不同的設備間所使用的 CSR 也不相同，鍵盤，滑鼠和序列介面由多功能卡(SuperIO)控制，IDE 磁碟由 IDE 控制器掌握，SCSI 磁碟有 SCSI 控制器控制。用來管理系統中硬體控制器的程式碼位於 Linux 核心中，而不是在每個應用程式中。用來管理硬體控制器的軟體通常叫做設備驅動程式。Linux 核心的設備驅動程式基本上是一些共享函式庫(Shared Library)，在函式庫中含有一些特權的，常駐記憶體的，一些用來處理底層硬體的程式。Linux 的設備驅動程序用來處理各種硬體的多樣性。



核心的外觀

系統中每一個設備都對應一個設備特殊檔案（例如，系統中的第一個 IDE 磁碟的設備檔案名是/dev/had），對於區塊設備(如，磁碟)和字元設備，它們的的設備特殊檔案通常是通過 mknod 命令用主設備號和次設備號來描述和創建。主設備號是對應到共用的設備驅動程式的，次設備號則用來區分不同的設備和設備控制器。因此可能相同的設備可以使用主設備號共用同一個驅動程式，但在次設備號則可有各別的控制器。

字元裝置不須要支援一個檔案所有的功能，例如一個揚聲器只要具備寫入資料的功能，而不須要從此設備中讀出資料。而檔案內部的搜尋功能，在磁帶上會被支援，但對滑鼠之類的裝置而言，就沒有其必要了。

網路裝置的處理不同於區塊和字元裝置，使用者不能直接將次料傳輸到網路裝置上，必須去使用核心裡的網路子系統（network subsystem）建立一個連結，接著才透過網路子系統和其他機器通訊。

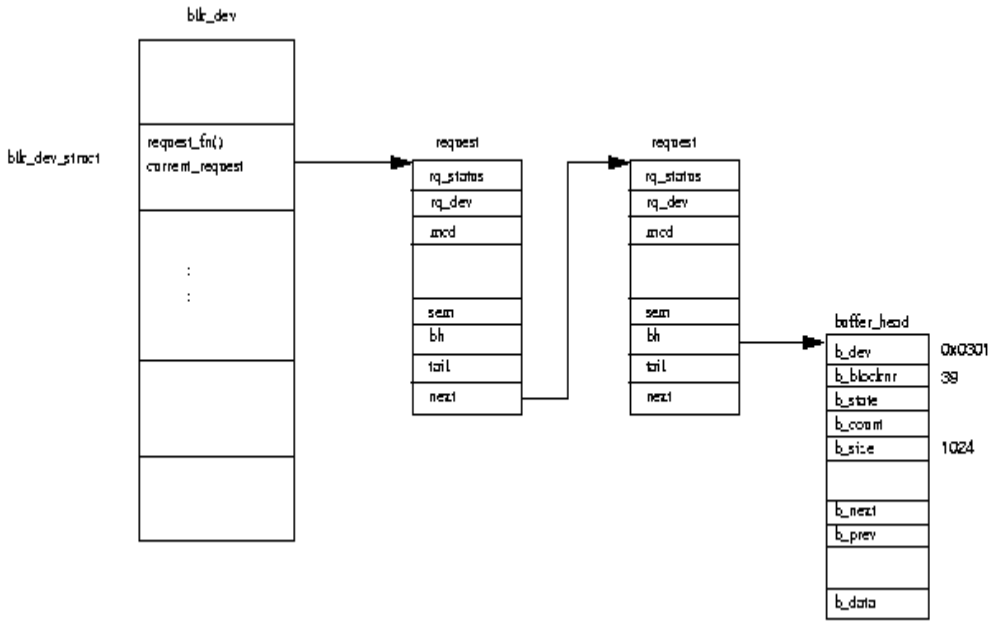
Linux 核心必須能夠通過一些標準的方法來和設備驅動程式介面。每一類設備驅動程序，(字元，區塊和網路)都提供一個一致的，共同的介面給核心以用來核心向它們申請服務。這些共同介面(common interfaces)意味著核心可以將這些不同的設備和其驅動程式一樣來對待。例如，SCSI 和 IDE 磁碟的行為是不同的。但 Linux 核心對它們使用一個同樣的介面進行操作。

Linux 是採動態的方式，可以重新建構的。每次 Linux 核心啟動時，可能偵測到不同的實體設備，因此需要不同的相應的設備驅動程式。在核心重新建構的時候，Linux 允許透過配置檔案方式將設備驅動程式帶進核心。當這些驅動程式在機器啟動初始化的時候，有可能系統中並不存在相對應的實體設備。有些驅動程式可以在需要時被裝載進入核心。為了處理設備驅動程式的這種動態特性，系統要求設備驅動程式在初始化時向系統進行登記。Linux 核心負責維護一些含有登記了的設備驅動程式的表。這些表中包含了一些例程(rountines)的指標和其他一些信息以用來支援核心與那些設備的介面。

5.2 區塊設備（Block Devices）

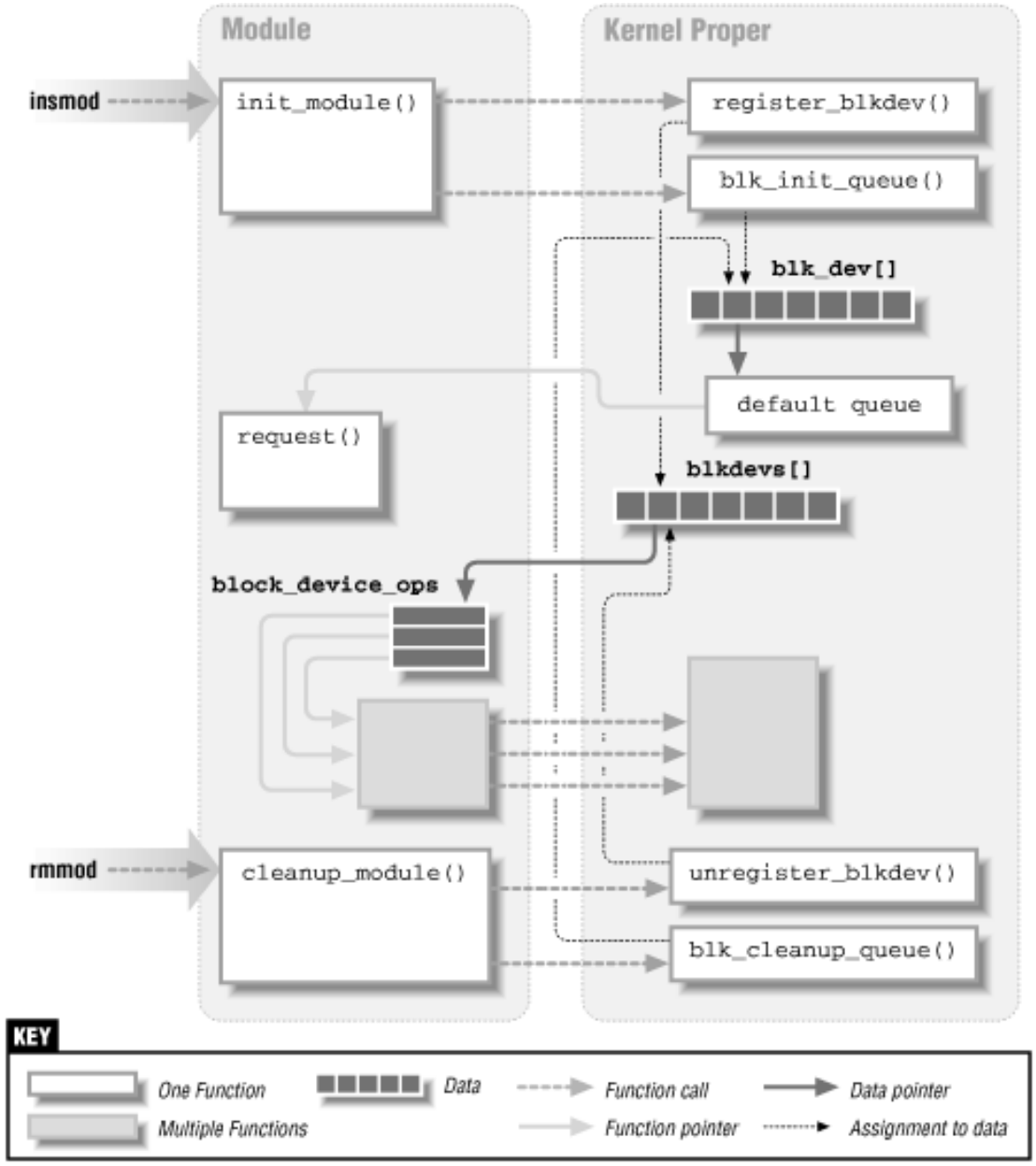
區塊設備同樣支援以檔案的形式被存取。當遇到打開區塊設備操作時，有一套用來提供對應的檔案操作的機制。Linux 在 blkdevs 向量中維護登記了的區塊設備。blkdevs 向量使用設備的主設備號作為其索引。向量的每一個入口仍是一個 device_struct 資料結構，這些數據結構是屬於區塊設備的。SCSI 設備和 IDE 設備是其中兩個例子。這些設備資料結構在核心中登記並為核心提供對應於其設備的檔案操作。對應於某類設備的設備驅動程序提供實現這些介面的細節。例如，一個 SCSI 設備驅動程式必須為 SCSI 子系統提供介面。SCSI 子系統利用這些介面，提供給核心一個一致的檔案介面。

除了檔案操作介面，每個區塊設備還必須提供緩衝區介面。每一個區塊設備驅動程式在一個 blk_dev 向量中添加其入口。blk_dev 向量的每個元素是一個 blk_dev_struct 數據結構。向量的索引仍然是設備的主設備號。blk_dev_struct 資料結構中含有一個請求例程的位址和一個指向 “request" 資料結構的指標。每一個 “request" 資料結構代表了一個從緩衝區到驅動程式的讀或寫資料塊的請求。



區塊設備的緩衝

每次一個緩衝區想要讀或寫一塊資料到根目錄/上一個已登記了的設備，它將插入一個"request"數據結構在 blk_dev_struct 中。每一個請求含有一個指向一個或多個”buffer_head"的資料結構。每一個 buffer_head 是讀或寫一個塊數據的請求。Buffer_head 結構是被緩衝區鎖住的。因此有可能存在一個程序正在等待對這個緩衝區操作的完成。每一個“request"資料結構是從一個靜態的鏈結串列中(all_requests)分配而來。如果一個請求(request)被加在一個空的請求隊列上，設備驅動程式的請求函數將被立即呼叫來處理這個請求。否則，驅動程式將依序地處理請求序列中的所有請求。



註冊一個區塊設備驅動程式

一旦設備驅動程式完成一個請求，它必須從這個請求中移去每一個 buffer_head 結構，將它們標誌成為更新並釋放對其的鎖。對一個 buffer_head 鎖的釋放將喚醒所有在睡眠中等待這個塊操作完成的程序。一個例子是：當要解釋一個檔案名時，EXT2 檔案系統必須從區塊設備中讀取下一個 EXT2 目錄項。這個程序將睡眠在那個含有目錄項的 buffer_head 上直到被設備驅動程式喚醒。這個 request 資料結構將被回收從而可以被其他的塊請求使用。

為了讓存取的效率高一點，我們使用了系統的兩個元件：區塊緩衝快取（block buffer cache）以及要求管理者（request manager）。

1. 區塊緩衝區快取

緩衝區快取在主記憶體中配置大小不固定的一組分頁，再將每個分頁切割成大小不相同的緩衝區。另外再建立相對應每一個緩衝區的描述器（buffer descriptor），稱為 buffer_heads。

Buffer_heads 中存放核心保有緩衝區所要用的到的資訊，主要的就是核心所用的辨識資訊。在緩衝區的辨識資訊中有三個欄位：緩衝區所屬的區塊裝置、在區裝置之中的資料偏移值、以及緩衝區的大小。緩衝區在兩種情況下會被送進未使用串列之中：第一種是檔案系統將緩衝區送進未使用串列；另一種則是當核心需要更多的緩衝區時，有必要去重新使用現存的緩衝區。

2. 要求管理者

要求管理者負責讀入及寫出緩衝區內容到裝置驅動程式。在每個區塊裝置驅動程式之中各有一個自己要求串列，排班是使用無方向性的電梯演算法（undirectional elevator algorithm ,C-SCAN algorithm）。演算法藉由調整在每個裝置的串列之中要求的先後順序來達到排班的目的。一個區塊裝置要到 I/O 工作完成之後它才會從串列中被移除，此時才接著下一個串列中的請求。

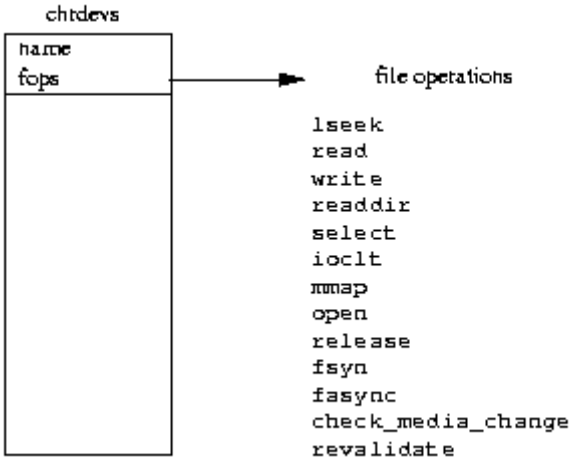
當做了新的 I/O 要求後，要求管理者會去把裝置佇列之中的要求合併在一起。當要求開始被處理之後，符合聚合要求的 buffer_heads 會一個一個去解除鎖定。

另一個關於要求管理者的特性是 I/O 要求能夠以跳過緩衝區快取的情況下進行。使用暫時性的 buffer_heads 來標示一個記憶體分頁的內容，只要這個分頁的最一個緩衝區完成它的 I/O 工作之後，整個分頁會解除鎖定，這 buffer_heads 也會被丟棄。

5.3 字元設備（Character Device）

字元設備是對不固定大小的資料區塊進行隨機存取的裝置驅動程式,在對字元裝置進行檔案的讀寫要求時，核心幾乎不會去進行任何預先處理的工作，只是簡單的把要求送到裝置上，然後讓裝置去處理這個要求。

字元式裝置可對其存取如同一般檔案,而這些對字元式裝置的存取功能便市由字元式裝置驅動程式實作出來的，通常這類驅動程式會提供開、關、讀、寫這四個標準系統呼叫。字元式裝置的特性是其傳輸資料的方式是一次只傳一個位元組的資料,而且是次序性(sequential)的而非隨機的(random)，字元式裝置本身通常只是資料傳輸通道，此類的裝置如終端機,鍵盤,滑鼠等。



字元設備

字元設備，Linux 中最簡單的設備，是通過”檔案“的形式被存取。應用程式使用標准的系統調用“打開”，“讀”，“寫”，和“關閉”字元設備就像它是一個檔案一樣，即使這個設備是一個被 PPP 監控程式(Daemon)用來將 Linux 系統連接上網的 Modem。當一個字元設備初始化時，它的設備驅動程式在 Linux 核心中登記，通過添加一個入口項(Entry)在含有 device_struct 資料結構的 chrdevs 向量中。這個設備的主設備號(例如，4 對於 tty 設備)被用來作為其在這個向量的索引。一個設備的主索引號是固定的。

chrdevs 向量的每一個入口項是一個 device_struct 資料結構，含有兩個元素。一個指向那個登記”在這個入口處“的設備驅動程式名字的指標；一個指向一系列檔案操作函數位址的指標。這些檔案操作函數位於這個字元設備的驅動程式裡並負責處理相應的具體的檔案操作如：打開，讀，寫和關閉。檔案 /proc/devices 中對於字元設備的內容是從 chrdevs 向量獲取的。

當一個代表一個字元設備的字元特殊檔案被打開時(例如/dev/cua0)，系統必須正確地工作保証相應的字元設備驅動程式的檔案操作例程被調用。就像一個普通檔案或目錄一樣，每一個設備特殊檔案對應一個 VFS inode。這個 VFS inode(資料結構)中含有這個設備的主和次設備號。VFS inode 是當一個特殊設備檔案名被查詢時，由文件系統所創建。

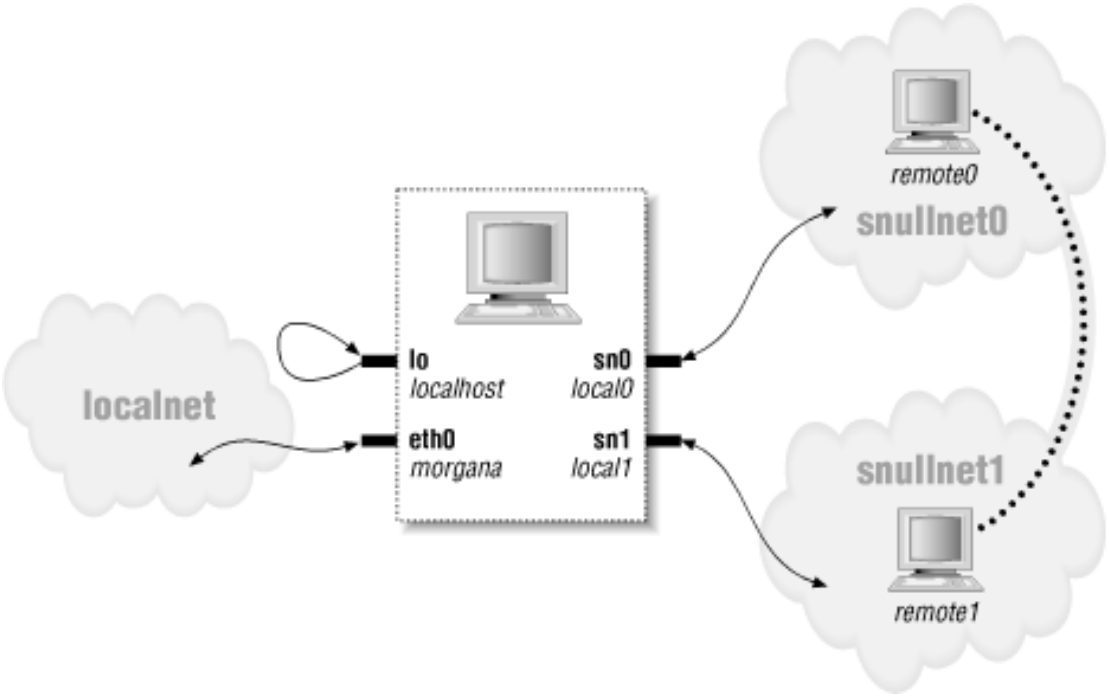
每一個 VFS inode 與一套檔案操作相聯繫。每當一個代表字元特殊檔案的 VFS inode 被創建時，對應這個 VFS inode 的檔案操作被設置成預設的字元設備操作。

當一個字元特殊檔案被一個應用程式打開時，這個“open"操作將使用這個設備的主設備號作為 chrdevs 向量的索引來尋找對應這個設備的檔案操作集的(例程的)位址。並且還要設置一個描述這個字元特殊檔案的資料結構--file，使得 file 結構中關於檔案操作的指標指向設備驅動程式中相應的部份。經過這些之後，所有用戶層的文件操作將被映射到對於這個字元設備的設備驅動程式提供的檔案操作。

5.4 網路設備

網路介面是 Linux 裝置的第三種標準類型。不同於字元式裝置和區塊式裝置，網路裝置並不依附在檔案系統中，而是在核心層就直接處理封包的傳遞與接收，因此也不為行程所開啟的檔案所限制。

與其他 Linux 設備驅動程式一樣，網路設備驅動程式也可以被預先構造在核心中。每一個潛在的網路設備都對應於一個 device 資料結構。這些結構組成一個由 dev_base 指向的鏈結串列。如果網路層需要網路設備完成一個特定的任務，它呼叫一個位址已在 device 結構中的網路設備服務例程。在最開始，device 結構中只含有初始化(initialization)和探測(probe)函數的位址。



主機如何找到它的界面

網路設備驅動程式需要解決兩個問題。首先，不是所有的已建構在核心中的網路設備驅動程式都有相應的週邊存在。第二，不管什麼樣的設備驅動程式，系統中的 ethernet 設備名始終是/dev/eth0, /dev/eth1 等等。網路設備不存在的問題很容易解決。因為當為每個網路設備初始化的例程被呼叫時，其函數返回值將顯示實體設備是否存在。如果不存在，這個驅動程式對應的 device 資料結構將從被 dev_base 指向的鏈結串列中被移去。如果驅動程式確實發現一個設備，device 結構的剩餘部份將被填充。這些包括設備信息和設備驅動程式中的支援函數的位址。

第二個問題是關於如何動態地將標準的/dev/ethN 設備特殊檔案名賦予值給系統中的 ethernet 設備。 在 device 鏈中，有 8 個標準記錄。從 eth0, eth1 到 eth7。它們的初始化都是一樣的：依次檢測是否每個 ethernet 設備驅動程式有相應的實體設備存在直到發現一個。當找到一個 ethernet 設備時，驅動程式填充其目前占據的 ethN device 資料結構。與此同時，網路設備驅動程式初始化剛剛找到的實體設備，找出該實體設備想要占據的 IRQ 號，DMA 通道等等。一個驅動程式有可能檢測到幾個它所控制的網路設備 ，在這種情況下，驅動程式將占據幾個/dev/ethN device 資料結構。當所有的 8 個標準的/dev/ethN 都被分配光之後，系統將不檢測其他的 ethernet 設備。

網路介面的角色和已掛載的(mounted)區塊式裝置相似，區塊式裝置暫存它的一些特色在 blk_dev 陣列和其它核心結構內，然後它便可以根據需求傳送與接收區塊資料；網路介面也以類似的方式運作，它會暫存資料在特定的資料結構，當需要跟外界交換封包時便可正常運作。 雖說類似，已掛載的磁碟和封包傳遞介面仍存有一些不同之處，首先，磁碟是以特殊檔案的形式存在於 /dev 的目錄下，而網路介面則無如此的進入點，由於網路介面不是資料流導向的裝置，因此無法輕易地依附在檔案系統的節點上，檔案系統的一般操作，如開、關、讀、寫，當應用至網路介面時並不合理，網路介面和應用程式之間的資料傳遞不是透過和檔案相關的標準系統呼叫，而是像 socket()、bind()、listen()、accept() 和 connect()這類的系統呼叫，所以在 Unix 系統裡"任何東西都是檔案"的觀念，在網路裝置並不適用；再者，區塊裝置驅動程式只對來自核心的要求(request)有反應，然而，網路驅動程式必須非同步(asynchronously)地接收外界的封包，因此，區塊裝置驅動程式會"被要求"送一個緩衝到核心；相對地，網路驅動程式會"要求"將進來的封包推向核心，應付網路的核心介面其操作模式的設計是不同的。

5.5 驅動程式

Linux 作業系統支援上百種驅動程式，幾乎會每一種實體裝置，如網路卡，繪圖卡，皆需有一個對應的驅動程式，不但如此，功能相同製造商不同的裝置也需有不同的驅動程式，再者，即使功能、製造商皆相同的裝置，在不同的作業系統時，如 Linux 與 FreeBSD，亦需有另一組驅動程式。在自由軟體大行其道，Linux 與 FreeBSD 熱門之際，各家硬體製造商無不希望自己的產品也能與這些作業系統相容，以佔有市場利基，因此必須不斷的為新硬體開發相容的驅動程式。驅動程式與作業系統的關係有相容性的問題，作業系統核心可約略分為:

1. 行程管理(Process management): 主要負責建立與刪除行程、行程對外的 I/O、行程之間溝通及排程 (scheduling)，簡而言之，行程管理為 CPU 提供一個抽象介面，好使數個行程可以不互斥地共同運作。

2. 記憶體管理(memory management): 負責分配記憶體資源給某一行程，並為該行程建立虛擬位址空間。

3. 檔案系統 (Filesystem): 在 Linux 中，幾乎每件事都可以檔案系統處理，大部分的硬體也遵循這樣的概念，硬體的檔案概念主要是放在 /dev 下。Linux 支援多個檔案系統格式，如 Linux 標準的 ext2 及常見的 FAT。

4. 裝置控制 (Device control): 系統操作最後還是得對應到某個硬體，除了 CPU 及記憶體外，元件的控制操作是相對應的程式碼執行，這組程式碼便是所謂的驅動程式。驅動程式可以模組化 (module) 的形式呈現，因此，系統可以一開始便將部分必用的裝置驅動程式，如鍵盤、硬碟，載入核心成為核心的一部分，再將其他的裝置驅動程式模組化，在需要時再載入以降低核心的大小。模組化是 Unix 的重要特色，此架構允許系統在運作期間擴充核心程式碼，給予使用者極大的彈性。

5. 網路 (Networking): 網路運作也需由作業系統管理，因網路並非特定行程，封包到來便是一非同步事件 (asynchronous)，系統必須負責應用程式和網路介面間的封包傳遞並根據網路狀況控制應用程式的進行。

5.6 直接記憶體存取-DMA(Direct Memeory Access)

DMA 代表 'Direct Memory Access'(直接內存訪問)。就是允許設備從 CPU 手中接管系統總線，并直接把數據傳送到主內存. 通常 CPU 分兩步來傳送數據: 1. 從設備的 I/O 存儲空間讀數據, 把數據放在 CPU 內部。 2. CPU 把數據從

其內部送到主內存。 DMA 方式通常用一步就可把數據從設備直接送到主內存。設備硬件必須內置有這種能力并不是所有的設備都可以使用 DMA 的。 從 DMA 傳輸占用系統總線開始的傳輸過程中 CPU 就不做什麼了。

當一個設備試圖進行 DMA 時， 它會發出一個請求(用改變總線的 DMA 請求連線的電平的方式)。 DMA 請求也可以用中斷的方式來實現， 但會有一定的延時， 所以為了快速， 就用一種特殊類型的中斷 'DMA-請求' 來實現。 像中斷一樣，把 'DMA-請求線' 編號來識別是哪個設備發出的請求。 這些編號就叫 DMA-通道。 因為 DMA 傳輸使用系統總線(同一時間只能有一個使用)， 所以它們實際上用同一個通道， 編號主要用來識別誰在使用通道。主板上的硬件寄存器紀錄各通道的當前狀態。 要發出一個 DMA 請求， 設備必須知道自己的 DMA 通道號， 通道號由物理設備存儲在自己內部。

資料量比較少時，使用中斷驅動設備驅動程式能順利地在硬體設備和記憶體之間交換資料。例如串列傳輸速率為 9600 的 modem 可以每毫秒傳輸一個字元。如果硬體設備引起中斷和調用設備驅動中斷所消耗的中斷時延比較大（如 2 毫秒）則系統的綜合資料傳輸率會很低。則 9600 串列傳輸速率 modem 的資料傳輸只能利用 0.002%的 CPU 處理時間。高速設備如硬碟控制器或者乙太網設備資料傳輸率將更高。SCSI 設備的資料傳輸率可達到每秒 40M 位元組。

直接記憶體存取，或 DMA，被提出用來解決傳輸上述大批量資料的問題。一個 DMA 控制器允許設備與記憶體之間發送或接收資料，但不影響處理器 CPU。PC 的 ISA DMA 控制器有 8 個 DMA 通道。第 7 個通道被用來為設備驅動程式服務。每一個 DMA 通道與一個 16 位的位址暫存器器和一個 16 位的計數暫存器相關聯。當想要發起一次資料交換時，設備驅動程式設置相應 DMA 通道的位址，計數暫存器的大小，這次資料傳輸的方向(讀或寫)。然後通知設備可以啟動 DMA 操作。當 DMA 結束時，設備才中斷系統。因此，在資料傳輸的過程中，CPU 可以作其他的事情。

設備驅動使用 DMA 時必須十分小心。首先 DMA 控制器沒有任何虛擬記憶體的概念，它只存取系統中的實體記憶體。同時用作 DMA 傳輸緩衝的記憶體空間必須是連續實體記憶體塊。這意味著不能在進程虛擬位址空間內直接使用 DMA。但是你可以將進程的物理頁面加鎖以防止在 DMA 操作過程中被交換到交換設備上去。另外 DMA 控制器所存取實體記憶體有限。DMA 通道位址寄存器代表 DMA 位址的高 16 位元而頁面寄存器記錄的是其餘 8 位元。所以 DMA 請求被限制到記憶體最低 16M 位元組中。

DMA 通道是“短缺”資源，只有 7 個通道。而且通道不能被設備驅動程式間共享。就像中斷一樣，一個設備驅動程式必須能夠知道哪一個 DMA 通道它要使用。有些設備使用固定的中斷號，就像有些設備使用固定的中斷號一樣。例如，軟碟設備使用的 DMA 通道一直是通道 2。有時一個設備的 DMA 通道可以由跳線來設置。許多以太(Ethernet)設備使用這種技術。一些更靈活的設備可以通過其 CSR 得知目前系統中哪些 DMA 通道是空著的。從而設備驅動程式可以隨便挑選一個 DMA 通道使用。

Linux 通過 dma_chan（每個 DMA 通道一個）陣列來跟蹤 DMA 通道的使用情況。dma_chan 結構中包含有兩個域，一個是指向此 DMA 通道擁有者的指標，另一個指示 DMA 通道是否已經被分配出去。當敲入 cat /proc/dma 列印出來的結果就是 dma_chan 結構陣列。

第6章 Linux 核心系統

6.1 簡介

許多人都會搞不清楚 Linux 的核心(kernel)、系統(system)與安裝套件(distribution)，這三者有何差異，Linux 的核心完全是由 Linux 共同體(Linux Community)從無到有所開發的，為 Linux 計劃中的主要核心，而一個完整的 Linux 系統仍須要其它的元件來構成，但是這些構成 Linux 系統的元件，就不一定屬於 Linux 計劃本身所擁有，可能使用了許多其它計劃的東西，例如：Berkerly 的 BSD 開發工具、MIT 的 X Window 系統，以及 GUN 計劃的東西。

不過這些技術的交流或元件的分享，其實是雙向的，例如：Linux 下的網路管理工具，是由 4.3BSD 中所衍生的，而在一些新的 BSD 衍生版本，如：FreeBSD，則反過來向 Linux 借程式碼。

在早期，任何人想要擁有 Linux 系統，可以從網路上取得 Linux 的系統元件的程式碼，必須自行編譯之後才能使用，安裝過程相當麻煩(須非常了解硬體狀況)，但 Linux 日漸成熟，就開始有人或團體發展安裝套件，簡化原本複雜的安裝工作，而這些安裝套件其實就是已編譯好的 Linux 元件組合。這些安裝套件提供的不只是基本的 Linux 系統，通常還包括了額外的管理工具及應用程式等。市面上已有數不清版本的安裝套件，其中最有名的是 Red Hat、Debian、Slackware，其它還有 Caldera、Craftworks、Work-Group Solutions 等，雖然版本很多，但不影響之間的相容性(核心都一樣)。

6.2 Linux 系統元件

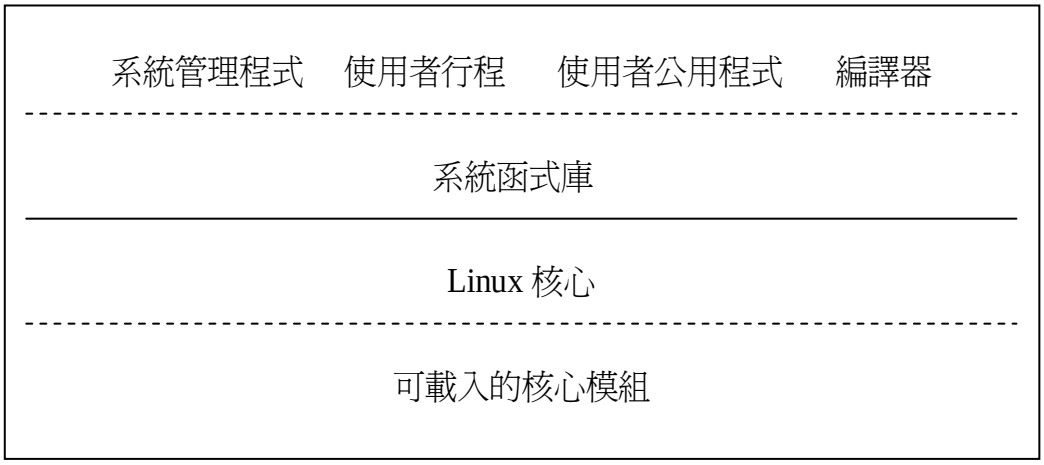
Linux 系統主要是由三個部份所組成：

1. 核心(kernel)：負責分配及管理系統的資源，如：虛擬記憶體、行程管理等。
2. 系統函式庫(system library)：為一組的函式，系統函式庫的功能繁多，最基本就是作為應用程式跟核心之間溝通的橋樑，許多的系統功能都是透由系統函式庫所完成的，但是它們不具有像核心程式一般的特權(privilege)，執行一個系統呼叫，涉及到將控制權由沒有特權的使用者

模式轉移到有特權的核心模式，應用程式並不像 UNIX 能直接向核心溝通。

除了上述基本的系統呼叫功能外，系統函式庫也提供了一些與系統叫無關的常式(routine)，如：排序演算法、數學函式、字串處理常式等。

3. 系統公用程式(system utility)：利用系統函式庫所完成的管理程式、系統公用程式、使用者公用程式。系統公用程式可能是系統初始化所須的程式，如：網路組態設定、核心模組載入程式，常駐的伺服器程式也算是系統公用程式，如：處理使用者登入要求、網路連線要求、列表機佇列的程式。



Linux 系統的組成元件

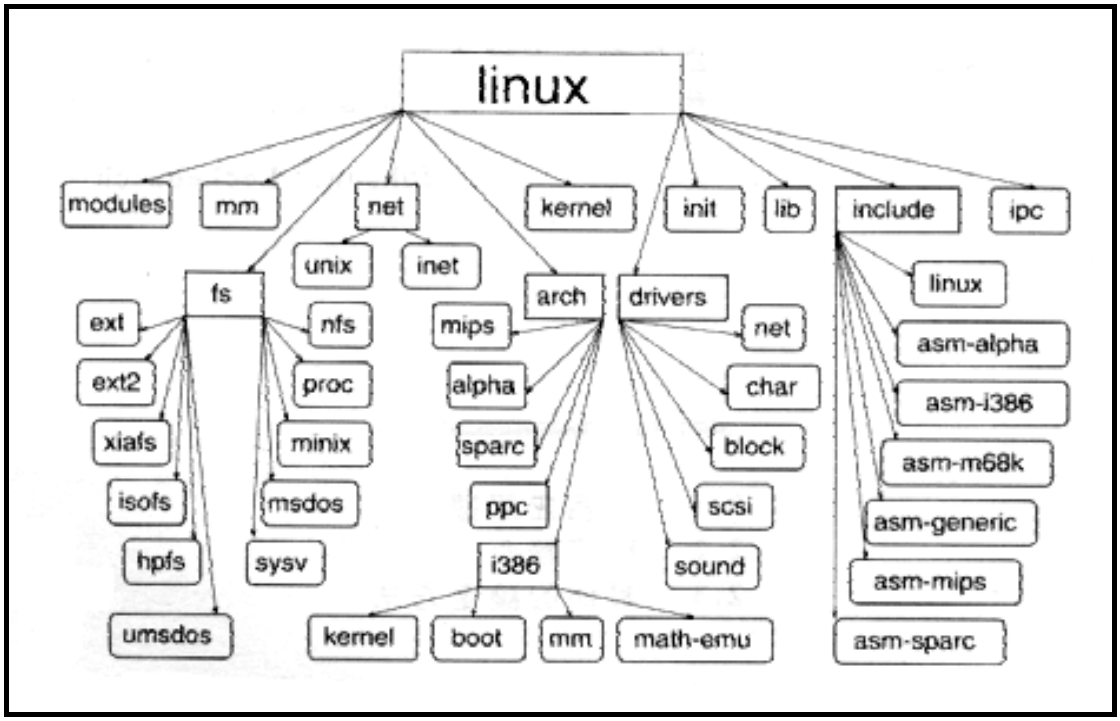
上圖為 Linux 系統的組成元件的層次關係圖，核心和其它系統元件最大的差異是，系統裡的程式是以處理器的特權模式執行，可以自由使用電腦裡的資源(physical resource)，這種特權模式在 Linux 中稱為核心模式與監督模式(monitor mode)相同。

現在許多的作業系統大多採用訊息傳遞的架構(message passing architecture)，但是 Linux 仍是採用與 UNIX 相同的方式，所有的核心程式、驅動程式、檔案系統等，都放在同一個空間裡，主要的原因是為了效率上的考量。在 Linux 裡，公用程式、使用者程式甚至是部份核心模組都可以動態地載入或移出，核心不必事先知道那些模組會被載入。

6.3 Linux 核心原始程式碼

Linux 與 Windows 最大的不同點在於,Linux 的原始程式碼採取了 GNU 的版權原則，也就是說，任何人都有權來使用、複製及修改這些程式而不必支付任何費用。Linux 開放原始碼使我們了解到作業系統本身，是由哪些部份所組成；藉此，我們能夠更加了解作業系統整體的運作方式，也才能知道系統程式是如何執行與管理。

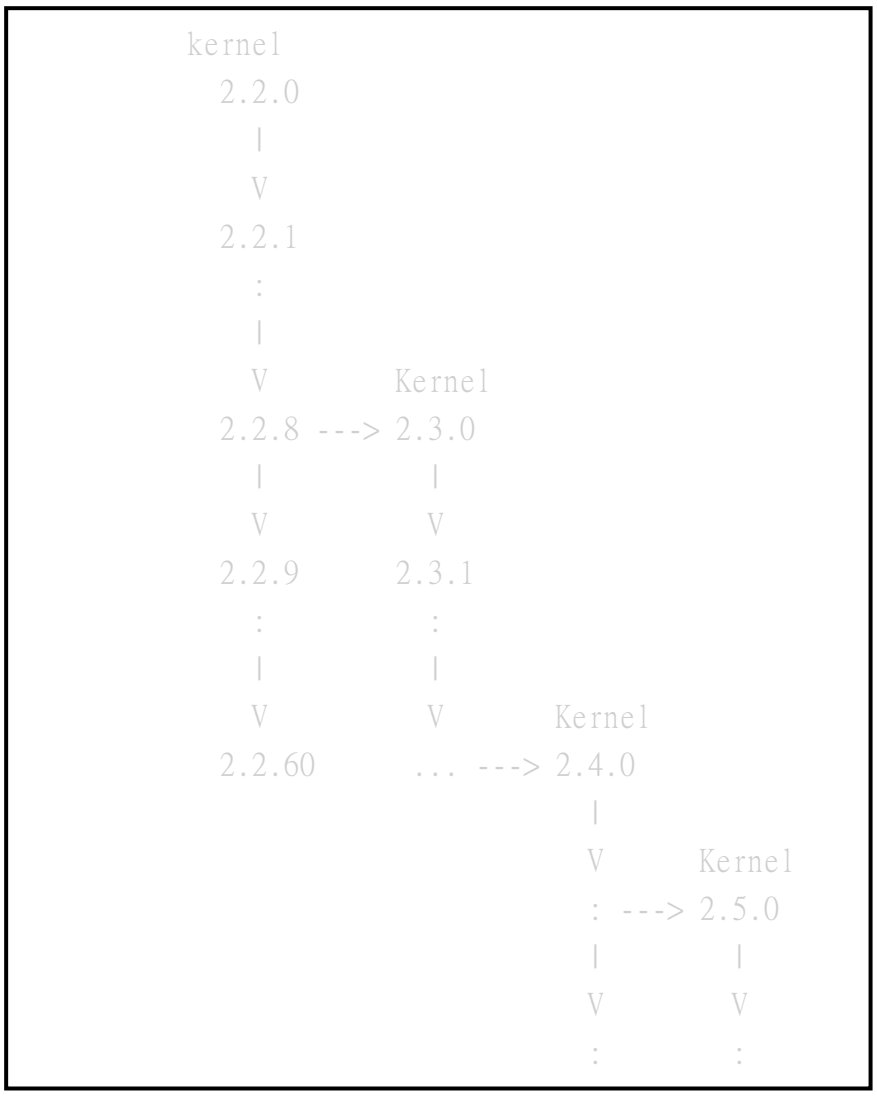
一般來說，核心的原始程式碼都是被放在 /usr/src/linux 這個目錄下。也由於 Linux 核心的原始程式碼非常的龐大，為了方便管理，核心程式被分成幾個部份，分別儲存於 /usr/src/linux 這個目錄下的不同目錄內。



原始碼目錄架構 [Linux 核心研究篇]

目錄名稱	內　容
init	啟動核心時所需的函式
kernel、arch/i386/kernel	核心的中心
mm、arch/i386/mm	記憶體管理部分
fs	虛擬檔案系統介面
drives	驅動程式
ipc	處理程序間通信機制
net	不同的網路協定
lib	一些標準 C 函式
module	編譯核心程式時所產生的模組
include	與核心相關的標頭檔

Linux 的版本編號主要有三個欄位，如：kernel-2.a.b。若 a 為偶數，代表此版本是正式發行的版本；若為奇數，就表示這個版本為測試版本。b 則是代表這個版本的修正次數。並非愈新的版本就是愈好的，當然，愈新的版本代表愈強大的功能，但也代表著愈會有穩定性的問題。



核心版本的維護流程圖[Study Area]

作業系統採用核心的好處有可攜性、穩定性以及效能：

- 可攜性

  Linux 是 UNIX-like 的作業系統，原系統再經過多次修改過後，已能應用於多種 CPU，如：Alpha、MIPS、Power PC、SPARC…等等。展現了相當優良的可攜性。

- 穩定性

  Linux 的核心發行版本在正式發行前，皆會經由全世界熱心的程式設計師來做檢查，一但發現有 bug，便可即時發佈消息並提供修正。

- 效能

  Linux 的核心使用「most-used pieces」的方式為排程來做最有效率的應用，因此，在穩定性和安全性的考量下，Linux 還是有相當優秀的效能表現。

6.4 Linux 核心模組

Linux 是單核心的結構，也就是說，它是一個大程式，其中任一函數都可以存取公共資料結構和其它函數。作業系統的另外一種可能的結構是多核心式的，各功能塊自成一體，相互之間由嚴格的通信機制相連。單核結構在添加新模組時，一種方法是重新調整設置，所以非常費時。例如，你想在核心中加入一個新的驅動程式，你就必須重新設定，重建核心。當然，還有另外一個辦法，Linux 允許動態裝載和卸載模組。Linux 模組是一段可以在機器起動後任意時間被動態連接的代碼。不需要的時候，就可以將它們從核心中卸載。大多數的 Linux 模組是設備驅動程式或虛擬設備驅動程式，如網路驅動程式，檔案系統...等。

你可以使用 insmod 和 rmmod 命令來裝載和卸載 Linux 模組，核心自己也可以使用核心常駐程式(Kerneld) 來按需求裝載和卸載模組。按需求動態裝載模組可以使得核心保持最小，並且更具靈活性。大量使用動態裝載模組的核心，其大小會相當的小。例如，讓 Linux 核心只在裝載 VFAT 分區時，才自動上載 VFAT 檔案系統模組。當卸載 VFAT 分區時，核心會檢測到，並自動卸載 VFAT 檔案系統模組。在測試新程式時，你如果不想每次都重建核心，動態裝載模組是非常有用的。但是，運用模組會多消耗一些記憶體，並對速度有一定影響。而且模組裝載程式是一段程式碼，它的資料將占用一部份記憶體。這樣會造成不能直接存取核心資源，造成效率不高的問題。

一旦 Linux 模組被裝載後，它就和一般核心程式碼一樣，對其它核心程式碼，享受同樣的存取權限。換句話說，Linux 核心模組可以像其它核心程式碼，或驅動程式一樣使系統崩潰。模組可以使用核心資源，但首先它需知道到哪去找。.例如，一個模組要使用 Kmalloc()（核心記憶體分配程式），但在模組建立時，它並不知道到哪兒去找 Kmalloc()，所以在它被裝載時，核心必須先設定模組中所有 Kmalloc() 使用的函數指標。核心有一個所有資源使用的列表，在模組被裝載時，核心重設所有資源使用的函數指標。Linux 允許堆疊式模組，即一個模

組呼叫另一個模組的函數。例如，由於 VFAT 檔案系統可以看成是 FAT 檔案系統的延伸，所以 VFAT 檔案系統模組需要使用 FAT 檔案系統提供的服務。一個模組使用另一模組的資源與使用核心資源很相似，唯一的不同是，被使用的模組需被先載入。一旦模組被載入後，核心將修改它的核心符號表(KERAEL SYMOBOL TABLE)，加入新載入模組提供的所有資源和符號。所以另一個模組被載入時， 它就可以使用所有已載入模組提供的服務。

當卸掉一模組時，核心先確定該模組不會再被使用，然後通過某種方式通知它。在該模組被核心卸載之前，該模組須釋放所有其占用的系統資源。例如，記憶體或中斷，當模組被卸載後，核心便會從核心符號表中刪除所有該模組提供的資源。

如果模組程式碼不嚴謹，它將使整個作業系統崩潰。另一個問題是，如果你載入的是其它版本服務的模組，那怎麼辦？例如，一個模組使用一個內函數，但提供了錯誤的輸入參數，這將會導致執行錯誤。不過核心可以在模組被載入時選擇性地通過嚴格的版本檢查來杜絕這種現像。

為了支援核心模組的運作，下列三個元件是必須的：

1. 模組管理(module management)：負責將模組載入記憶體，載入的工作不只是把程式搬進核心記憶體而已，還必須保証新載入的模組所參考到的核心符號(kernel symbol)及進入點(entry point)可以指向正確的核心所在。Linux 為了解決這個問題，將模組載入工作分成兩個部份，第一部份即管理核心之模組，第二部份則負責處理模組的符號參考。

   在模組管理部份，核心提供一個介面讓模組管理程式(module-management program)可以跟模組須求提出者(module requestor)作連接，當某個行程要求使用某個尚未載入的模組時，核心將會通知管理程式，在模組被載入後，管理程式仍會定時詢問核心，看看動態載入的模組是否有在使用，如果沒有的話將被移除。
2. 驅動程式登記(driver registration)：在核心之中，會有一個註冊表(registration table)來儲存目前已知的驅動程式的資訊。
3. 衝突排解機制(conflict resolution mechanism)：在現今電腦上可能安裝了許多不同的裝置，例如：網路卡、SCSI 卡、顯示卡等，所以硬體組態管理就顯得相當重要。Linux 提供一個集中式的衝突排解機制，該機制的目標如下：
   a. 避免多個驅動程式存取相同的硬體資源而造成的衝突。
   b. 避免自動探測(auto probe)影響現存的裝置驅動程式。

6.5 模組裝載

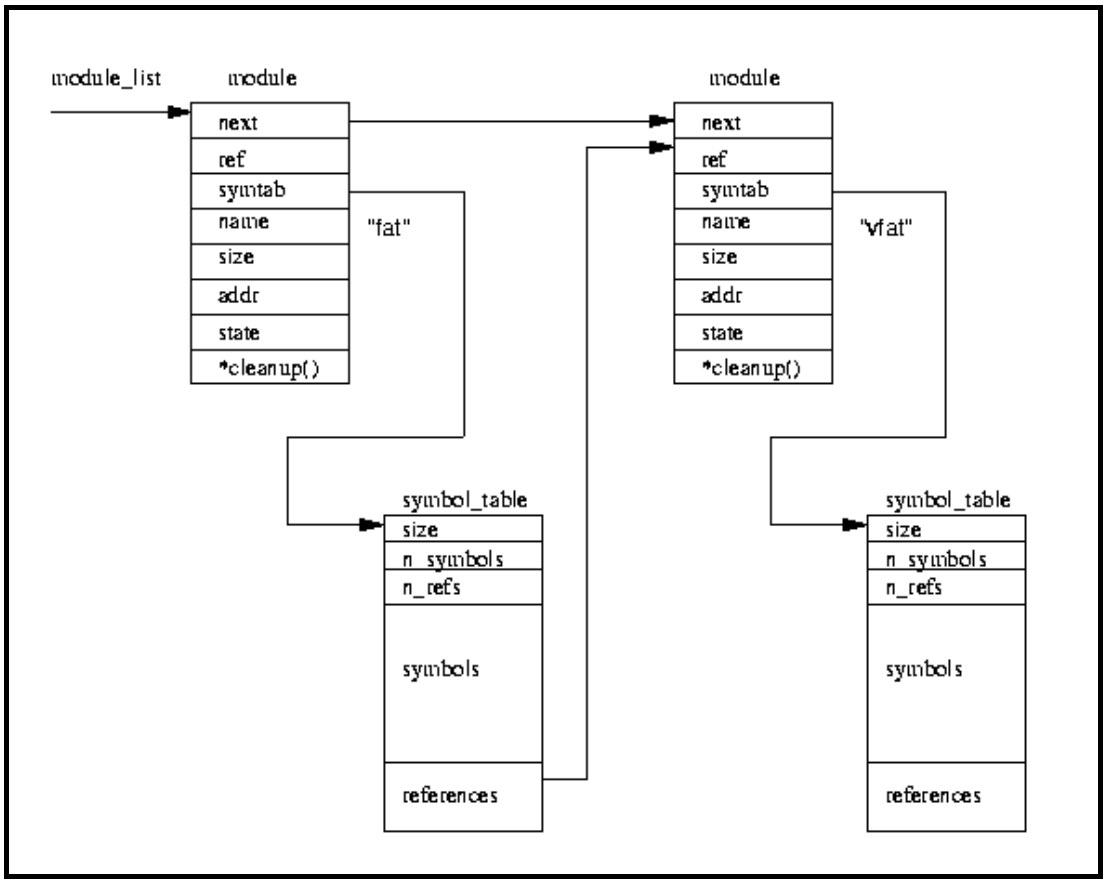
載入模組有兩種方法。第一種是通過 insmod 命令來載入；另一種更聰明的方法是在模組被使用到時自動載入，這叫需求載入(DEMAND LOADING)。例如，當用戶在裝一個不在核心中的檔案系統，核心會自動調用核心常駐程式(KERNELD)來載入對應的處理模組。核心常駐程式是一個具有超級用戶權限的普通用戶程式。當它被啟動時(通常在系統啟動時)，它將開啟一個和核心之間的程序間通信管道(IPC CHANNEL)。核心再利用這條管道來通知程序常駐程式去完成各種任務。

核心駐留程式的主要任務是裝載和卸載模組，但它也能完成其他的一些任務。如按需開啟和關閉一條通過序列的 PPP link。KERNELD 自己並不完成這些任務，它是使用如 insmod 這樣的命令來完成，KERNELD 只是一個核心的代理，協調完成各項任務。

在載入模組時，insmod 命令必須先找到要被載入的模組。需求載入的模組通常被放在 /lib/modules/kernel-version 下，這些模組與一般系統程式都是已連接好的目標程式碼，不同的地方在於模組是可重定位的映像檔案。也就是說，模組並不是從一個固定的位址開始執行的，模組可以是 a.out，也可以是 ELF 格式的目標程式碼。insmod 透過一個有系統權限的呼叫來找到核心中可被使用的資源。

系統(資源)符號由名和值兩個部份組成。核心用 module_list 指標指向其管理的所有模組所串成的鏈結串列。核心的輸出符號表在第一個 module 資料結構中，並不是核心所有的符號都能被模組使用，可使用符號必須被加入輸出符號表中，而輸出符號表是與核心一起編譯連接的。例如，當一驅動程式想控制某一系統中斷時，它需要呼叫 ” request_irq” 這個系統函數，其在核心中的值，你可以看/proc/ksyms 檔案或用 ksyms 來查詢。ksyms 命令可以顯示所有核心輸出符號的值，也可以顯示載入模組的輸出符號的值。當 insmod 載入模組時，它先將模組載入虛擬記憶體，根據核心的輸出符號，重設所有核心資源函數使用的指標。即在模組的函數呼叫處寫入對應符號的實體位址。

當 insmod 重設完核心輸出符號的位址後，它將呼叫一個系統函數，要求核心分配足夠的空間，這時記憶體就會分配一個新的 module 資料結構和足夠的記憶體來裝載這個新模組，並把這個 module 資料結構加到模組鏈結串列的最後，並設置成未初使化(UNINITALIZED)。



核心模組鏈結串列[David A Rusling]

上圖顯示的是核心載入 FAT 和 VFAT 兩模組後的模組鏈結串列。鏈結串列的第一模組並沒有顯示出來，那是一個虛擬模組，只是用來記錄核心的輸出符號表。你可以用 insmod 命令來列出所有載入模組及它們之間的關係。insmod 只是格式化的輸出記錄核心鏈結串列的 /proc/module 檔案。insmod 可以存取核心分配給新載入模組的記憶體，它先將模組寫入這塊記憶體，然後對它進行重定位處理，使模組可以從這個位址開始執行。由於每次模組被載入時，無論在不在同一台機器上，都不大可能分配到相同的記憶體位址，所以重定位(即重設它的函數指標)是必須的。

新載入模組也可以輸出符號，insmod 會為這些符號建一個表。另外，每一個模組必須有自己的初始和清理(即析構)函數。這兩個函數不能被輸出，但它的位址將在初使化時由 insmod 傳給核心。

當一個新模組被載入核心時，它要更新系統符號表及被它使用到的模組。核心中被使用到的模組都需在其符號表的最後保留一列指向呼叫模組的指標。上圖顯示 VFAT 檔案系統依賴於 FAT 檔案系統，所以，在 FAT 模組中有一個指向 VFAT 的指標，這個指標是在 VFAT 被載入時加入的。核心將呼叫模組的初使化函數，

如果成功，它將繼續完成安裝新模組的任務。模組的清理函數的位址將被存在它的 module 資料結構中。當模組要被卸載時，它就會被呼叫。

6.6 模組卸載

用 rmmod 命令可以卸掉一個指定模組，但當需求載入模組沒用時，它就會被核心自動卸掉，KERNELD 每次被啟動時，它會呼叫一個系統函數將所有沒用的模組從核心中卸載。例如，如果你裝了一個 ISO9660 的 CDROM，並且它的檔案系統是一個需求載入模組，那麼當你卸載 CDROM 後不久，ISO9660 檔案系統也會被從核心中卸載。你可以在起動 KERNELD 時，設置其被啟動的時間間隔。

當還在被其他模組使用時，模組是不能被卸載的。例如，當你還在用 VFAT 檔案系統時，VFAT 模組不會被卸載。當你看 insmod 命令的輸出時，你會發現每個模組都帶有一個計數器。這個計數器記錄依賴於該模組的模組數，也就是使用到該模組的模組數。

例如:

```
Module:         #pages:  Used by:
msdos               5                  1
vfat                4                  1 (autoclean)
fat                 6    [vfat msdos]  2 (autoclean)
```

在上面的例子中， VFAT 和 MSDOS 都依賴於 FAT 模組，所以 FAT 模組的計數器為 2，VFAT 和 MODOS 的都為 1，表示只有檔案系統依賴於它們。如果再載入一個 VFAT 檔案系統，則 VFAT 模組的計數器將變成 2。

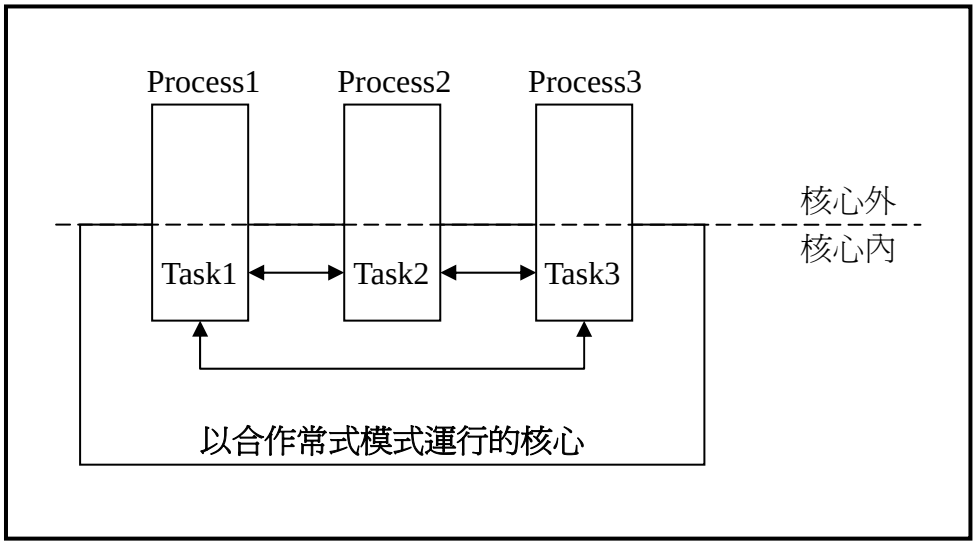
模組的計數器是它映像的第一個長字(longword)。這個長字同時也記錄了 autoclean 和 visited 兩個標誌，只有需求載入模組才使用到這兩個標誌。autoclean 用來讓系統識別哪一個模組需被自動卸載。visited 標誌表示該模組是否還在被其它模組使用。每次 KERNELD 試圖卸載已經沒用的需求載入模組時，系統會檢查所有模組，它只注意標示為 autoclean 並正在運行的模組，如果這個模組沒有設置 visited 標誌，則它將被卸載。否則，系統就清掉 visited 標誌，並繼續檢查下一模組。

當一個模組可以被卸載時，系統會呼叫它的清理函數來釋放它所占用的所有系統資源。該模組的 module 資料結構將被標為 deleted，並從模組鏈結串列中去除，所有它依賴的模組會修改它們的指標，表示該模組已不再依賴它們了。所有該模組占用的記憶體便會被釋放掉。

6.7 行程

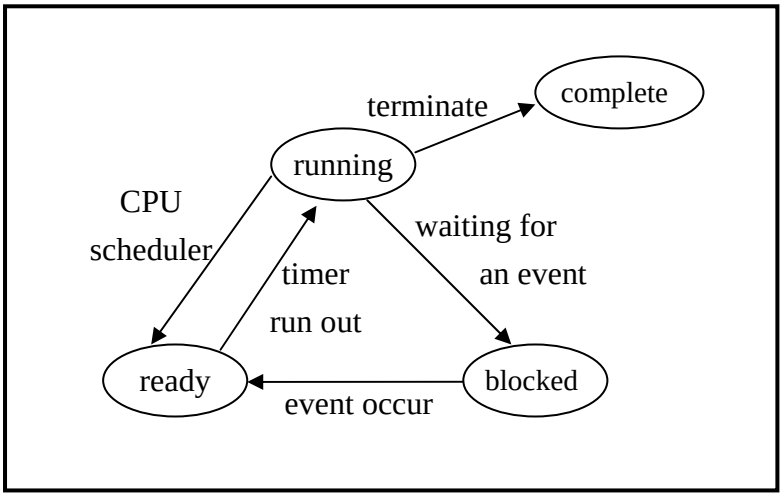
行程（process)，又可稱為處理程序。以 Linux 下的處理程序來看，核心只是一個服務的提供者，每個處理程序獨立的運作，彼此間並無法直接影響。處理程序各有各的記憶體空間，這個空間是受到保護的，所以處理程序間無法互相修改內容。但以 Linux 系統內部來看，作業系統是唯一執行的程式，且這個程式可以存取所有的資源。不同的任務（task，即為從外部來看的處理程序)是以合作常式（co-routines)的型式執行，每個任務都可以決定是否要將控制權轉移給其他任務，每個任務也都可以使用其他任務的所有資源並加以修改。

一個任務可能有某些部份在處理器中是以較低優先等級的使用者模式(User Mode)來執行，由核心外來看，這些部份就是處理程序。以這些處理程序的觀點來看，系統便是以多工型態在運行。



[Linux 核心研究篇]

行程自產生到結束有三種主要的狀態，執行（running)、等待（blocked）以及預備（ready)，其間的關係可用來下圖描述：



行程主要狀態圖

1. 執行狀態（running)：處理程序正在執行中，只有透過中斷或系統呼叫才能使處理程序離開執行狀態。
2. 等待狀態（blocked)：處理程序正等候某事件的發生(例如：I/O 動作)，只有該事件發生後處理程序才會繼續。
3. 預備狀態（ready)：處理程序正在排隊等候處理器的執行權，而目前正由其他處理程序使用處理機中。

在 Linux 中，一個正在執行的程式我們稱他為一個處理程序。由於 Linux 是一個多工的作業系統，因此可以在同一個時間執行多個處理程序(以使用者的角度來看，事實上同一時間還是只能執行一個處理程序)，每個處理程序都擁有一個唯一的 ID，稱為 Process ID，而這個 PID 主要是用來區別各個處理程序。

你可以執行 ps 指令列出系統正在執行的處理程序，其結果如下：

```
[vic@dhcp-2044-2 vic]$ ps
PID    TTY      TIME      CMD
1114   pts/0    00:00:00  bash
1225   pts/0    00:00:00  ps
```

有一個指令—top，就像 Windows 裡的「Windows 工作管理員」一樣，它是用來監控系統的資源(包括記憶體、swap 分割區和 CPU 使用率…等等)，且會周期性的更新內容。其結果如下：

```
[vic@dhcp-2044-2 vic]$ top
  9:50pm  up 41 min,  1 user,  load average: 1.62, 1.30, 0.65
86 processes: 83 sleeping, 3 running, 0 zombie, 0 stopped
CPU states: 33.0% user,  7.0% system,  0.0% nice, 59.8% idle
Mem:    94100K av,   93028K used,    1072K free,       0K shrd,    624K buff
Swap:  192772K av,   60964K used,  131808K free                   55628K cached
PID USER     PRI  NI  SIZE  RSS  SHARE STAT %CPU %MEM TIME COMMAND
1106 vic      16   0  9712  8404  7172  R    17.5  8.9  0:19 gnome-terminal
 921 root      6 -10 18300  6796  4232  S    14.9  7.2  0:52 X
1226 vic      16   0  1024  1024   812  R     2.0  1.0  0:04 top
1047 vic      15   0 11084  6772  5820  S     0.5  7.1  0:08 rhn-applet-gui
1020 vic      15   0  7564  5752  5040  S     0.1  6.1  0:07 metacity
   1 root     15   0   444   412   392  S     0.0  0.4  0:04 init
   2 root     15   0     0     0     0  SW    0.0  0.0  0:00 keventd
   3 root     15   0     0     0     0  SW    0.0  0.0  0:00 kapmd
   4 root     34  19     0     0     0  SWN   0.0  0.0  0:00 ksoftirqd_CPU0
   5 root     15   0     0     0     0  SW    0.0  0.0  0:02 kswapd
   6 root     25   0     0     0     0  SW    0.0  0.0  0:00 bdflush
   7 root     15   0     0     0     0  SW    0.0  0.0  0:00 kupdated
   8 root     25   0     0     0     0  SW    0.0  0.0  0:00 mdrecoveryd
  12 root     15   0     0     0     0  SW    0.0  0.0  0:00 kjournald
  64 root     15   0     0     0     0  SW    0.0  0.0  0:00 khubd
 156 root     15   0     0     0     0  SW    0.0  0.0  0:00 kjournald
```

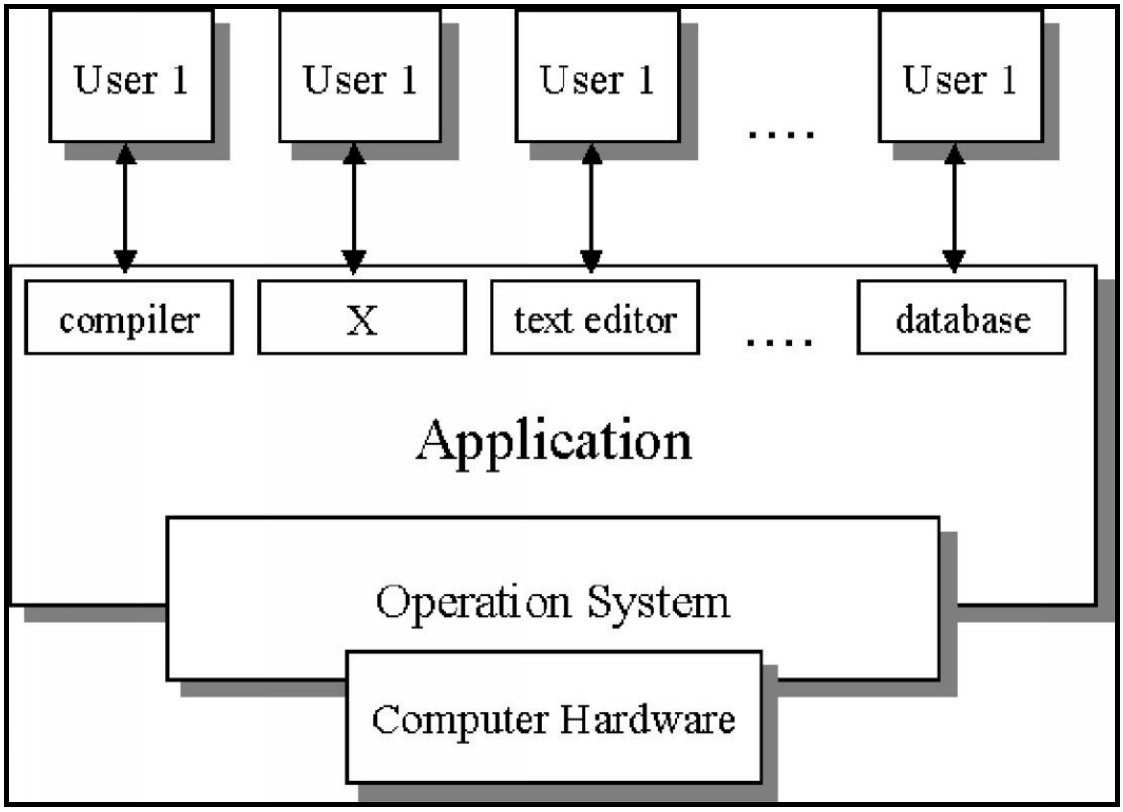
當你在 shell 下要執行一個程式時，shell 會向核心發出一個 fork request，fork 的功能是從一個已存在的處理程序分裂出一個新的處理程序。我們知道，shell 本身也是一個處理程序，所以當 shell 向核心發出一個 fork request 的時候，一個幾乎和 shell 相同(環境與開啟的檔案)的處理程序就會被建立，這時，shell 就是所謂的父處理程序，而新建立起的處理程序便是子處理程序。

所有的處理程序都是以這樣的方式建立,所以每個處理程序都會有一個父處理程序。由於 Linux 系統再開始的時候最先執行的程式是 init (Prosses ID 為 1)，因此，init 是唯一可以直接由 Linux 核心執行的處理程序，在使用者登入後所建立的每個處理程序都會將 shell 當作祖先，而 shell 則將 init 當作父處理程序。init 是所有處理程序最上層的祖先，所有的處理程序都是由 init 所衍生出來的。

第7章 系統管理

7.1 系統概述

作業系統是現代電腦系統維持正常運作不可或缺的一環，它是在電腦上執行的一個特殊程式，而且通常在電腦一開機就進駐記憶體執行，主要是提供其他程式執行、正常運作所需的環境，並且作為系統上所有程式跟電腦硬體之間的溝通橋樑。



電腦系統架構簡圖 [Xlinux 安裝手冊與說明]

上圖簡化了作業系統的功能，事實上，作業系統是一個非常複雜的程式，它所擔負的工作也不只是驅動週邊設備而已，像 Linux 這樣的一個完整的現代作業系統，它所要負責的工作大致上可以分成以下幾個部份與功能：

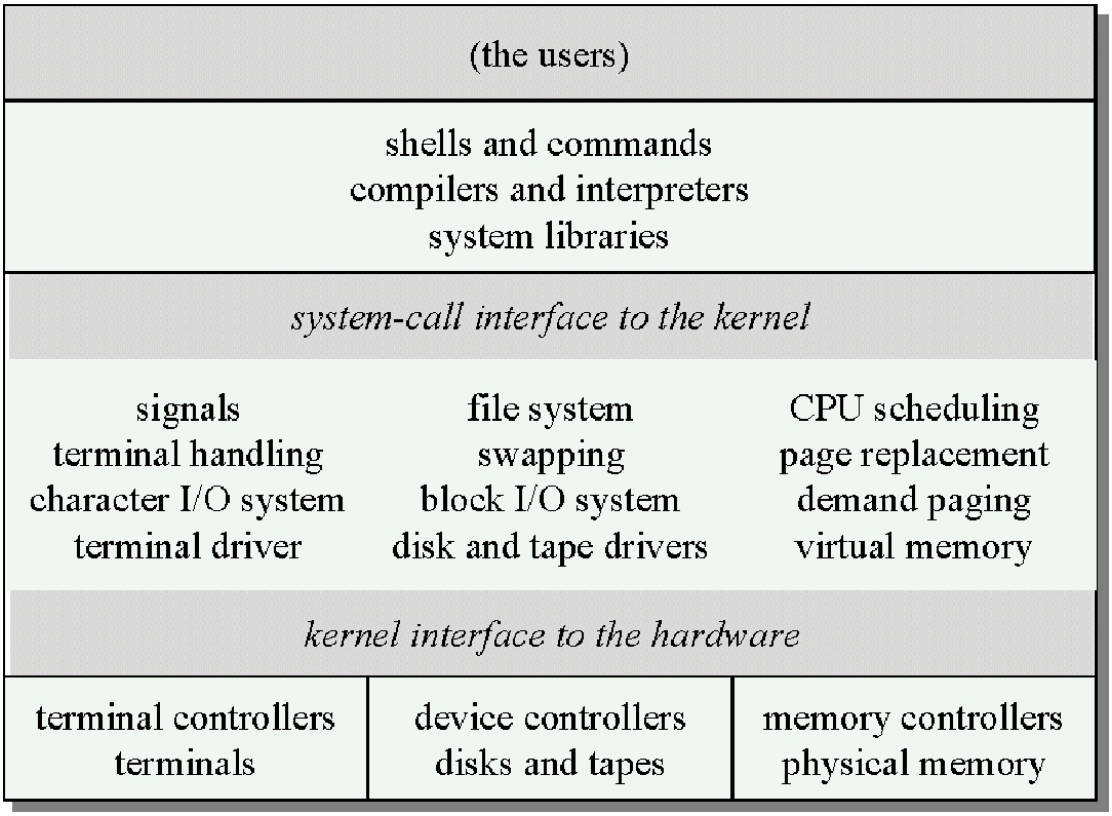
1. 程序管理 (Process Management)
2. 主記憶體管理 (Main-Memory Management)
3. 延伸記憶體管理 (Secondary-Storage Management)
4. 輸出入系統管理 (I/O System Management)

5. 檔案系統管理 (File System Management)

6. 保護子系統 (Protection System)

7. 網路子系統 (Networking System)

這個架構並不是一成不變的，而會依各個作業系統的實作規格而有所不同，在 Linux 系統上，是以所謂的 "系統呼叫" 來實作這些功能，一般的應用程式是透過系統呼叫的方式跟作業系統要求服務，下圖便是一般 Linux 系統提供的系統服務架構。



Linux 系統服務架構 [Xlinux 安裝手冊與說明]

前面兩張圖非常簡略的描述了作業系統的功能架構，這邊所謂的作業系統是用比較狹義的定義，並不指整個包括系統工具的程式集合，主要是指提供系統低階服務的系統核心部份。

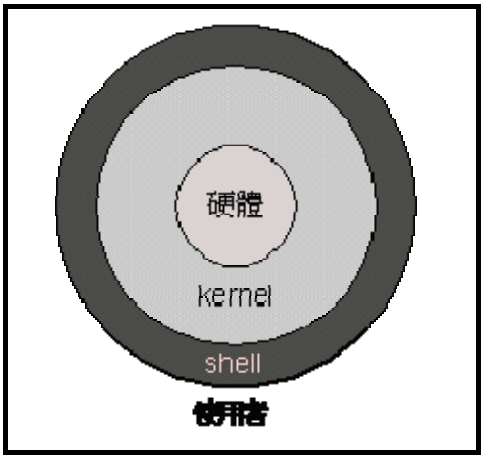
7.2 Shell 指令解譯程式

Shell 是 Linux 系統的指令解譯程式，為何說它是“程式”呢? 先前我們在介紹核心的時候，就已經了解到，整個 Linux 系統皆藉由核心來對系統中所有的

行程作處理。因此，Shell 不僅單單扮演著指令解譯程式的角色，還肩負著保護核心以及作為使用者與系統交談的介面。由於 shell 是使用者與作業系統間的主要介面，許多使用者會將 shell 與 Linux 視為一體，但其實 shell 與核心是分開的，shell 只是 Linux 作業系統中的一支程式而已。使用者登入系統後，會出現在 home directory 的目錄下，同時系統便會自動執行 shell 程式，他接受由使用者輸入的指令，執行並交給核心去處理。

那麼，kernel 又如何知道我們鍵盤輸入的東西是什麼呢? 那就是我們這裡介紹的 shell 所負責的事情了。因為電腦本身所處理的數據，都是二進位的機器碼，和我們人類習慣使用的語言很不一樣。比方說，輸入 pwd 命令，我們知道這是 print working directory 的意思(非常簡單的人類語音)，但作為 kernel 來說，它並不知道 pwd 是什麼，kernel 只會看機器碼，這時候，shell 就會幫我們將 pwd 翻譯為 kernel 能理解的程式碼。所以，我們在使用電腦的時候，基本上就是和 shell 打交道，而不是直接和 kernel 溝通，更不是直接控制硬體。

簡單來看，我們就這樣來看待它們的關係: 光從字面來解析的話，shell 就是“殼”，kernel 就是“核”。好比一個果實一樣，您第一眼看到的就是殼，把殼扒開才看的到裡面的核。shell 就是使用者和 kernel 之間的界面，將使用者下的命令翻譯給 kernel 處理，關係如下圖:



Shell 架構關係圖 [Study Area]

Linux 的 shell 程式有許多不同的版本，如：sh、bash、tcsh、csh、pdksh、zsh…等等。bash 是大部份 Linux 版本的預設值，當然，使用者可以依自己的習慣，安裝適合的 shell 程式。當你要使用其他版本的 shell 時，有幾種方法，第一種是可以輸入 echo $SHELL，$SHELL 是 shell 的值。第二種是修改/etc/passwd 檔案，每個帳號在 passwd 檔案中都會有一行的紀錄，其中最後一個欄位所紀錄的就是使用者登入時所要使用的 shell，改變這個欄位的值便可改變 shell，如：

```
root:x:0:0:root:/root:/bin/bash
```

第三種是某些 Linux 套件，如 Slackware Linux，可以使用指令 usermod 來更改成要使用的 shell。

就如同 DOS 一般，Shell 也有提示符號。只不過長相不大相同，而這“長相”也可以依據不同版本的 Shell 讓使用者自行設定。Linux 的 Shell 輸入指令的格式習慣與以往的 DOS 有些許的不同：

$ 指令 –選項(參數) 對象	**Linux Shell**
c:\>指令 對象 /選項(參數)	**DOS**

當你在 Shell 下輸入指令的時候，你所鍵入的資料都會先被儲存在一個緩衝區(Buffer)中，直到 Enter 鍵被按下後，在緩衝區裡的資料才會被送給核心去執行，在此之前，那些被鍵入的資料都是可以編輯的。Linux 預設的 Shell—BASH 會有一個歷程清單，這個清單紀錄著先前你所執行過的所有指令，你只需要按方向鍵上，就可以往前尋找之前已經執行過的指令，並可再執行一次，省去輸入的時間。

Shell 是如何解譯你輸入的資料呢? Shell 在處理時有三個步驟：

1. 讀取
2. 分解指令
3. 執行

讀取僅僅是單純的將使用者的輸入讀入。分解指令便是將讀入的資料加以分解，並分出指令、選項及參數。第三步驟則是執行指令，並將選項及參數傳給指令本身。

shell 也有支援一些特殊字元如下：

字 元	功 能
*	萬用字元
?	對應於命令列任何單一字元
~	tilde，代表使用者的簽入目錄
..	上一層的目錄
.	目前的工作目錄
[ccc]	對應於括號內的字元，也可使用-來表示一段範圍
[!ccc]	對應於任何括號內字元以外的其他字元
{}	延伸既有的字串，無論是檔案或目錄
&	指令以幕後的方式執行
\	跳脫符號，用以解除特殊字元的特殊意義

;	分隔符號，分隔多個指令
>	將輸出流重導至檔案
<	指令的輸入流由檔案導入
>>	將輸出流加到已存在的檔案後面
>&n	將輸出寫到檔案描述詞為 n 的檔案(已被開啟，0~19)
<&n	將輸入改由檔案描述詞為 n 的檔案讀取
>&-	關閉標準輸出檔
<&-	關閉標準輸入檔
|	建立管線，使一指令的輸出成為另一指令的輸入
$	bash 的提示符號，也是 shell 語言的位置參數
#	註解
$$	AND
||	OR
‘	取代指令字元

[Linux 聖經]

7.3 帳號管理

Linux 預設的系統管理員帳號為 root，一直用 root 帳號登入不是一個好主意，容易洩漏系統管理員的密碼。一般在安裝新系統時，安裝步驟都會要求安裝者先建立新的使用者帳號及密碼，當然，如果略過的話，在安裝完成後還是可以再來建立使用者帳號與密碼的。

1. 建立帳號(adduser)

```
/usr/sbin/adduser
usage:
adduser [-u uid [-o]] [-g group] [-G group,...] [-d home] [-s shell]
adduser -D [-g group] [-b base] [-s shell] [-f inactive] [-e expire]
```

這裡我們使用最基本的方法; adduser 命令。

```
[root@linux /root]# adduser vic
[root@linux /root]#
```

再用 vic 登入，登入畫面如下：

```
Red Hat Linux release 8.0 (Psyche)
Kernel 2.4.18-18.8.0 on an i586
login: vic
Password:
Login incorrect
login:
```

因為沒有設定新帳號的密碼，所以不知密碼便無法登入。

2. 使用者密碼

passwd 命令可以用來:

- 為新建立的使用者分配密碼
- 修改已存在的使用者的密碼
- 修改您登錄的使用者的密碼

前兩種情況實際上是一樣的;一個新建立的使用者和一個已存在很久的使用者沒什麼區別(至少對 passwd 是這樣的)。必須以 root 身份登入，才能夠修改使用者帳號的密碼，當然，使用者本身也可以修改自己的密碼。關於 passwd 指令的參數，請執行：

```
[root@linux /root]# passwd –help
or
[root@linux /root]# man passwd
```

(man 為 Linux 中查詢指令用法的程式)

修改密碼方式如下：

```
[root@linux /root]# passwd vic
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully
[root@linux /root]#
```

再你輸入密碼的同時，密碼並不會再螢幕上顯示出來，您還必須輸入兩次密碼，以確保您沒有輸入錯誤。

3. 刪除使用者的帳號(userdel)

/usr/sbin/userdel

當你要刪除使用者的帳號時，可下指令：

```
[root@linux /root]# userdel 帳號
若欲將其自家目錄也一併清除時
[root@linux /root]# userdel －r 帳號
```

值得一提的是，為了系統安全減少使用 root 系統管理者帳號登入，但若需要管理系統時，先登出再改以 root 帳號登入過於費時。這個時候可以使用 su 指令，在輸入系統管理者的密碼後，即可暫時變成超級使用者(super user)，super user 也是 root 的另一種稱呼。

接下來介紹一下密碼檔與群組檔的結構。

密碼檔為 /etc/passwd，是由一筆一筆的紀錄所組成，每一筆密碼記錄共有 7 個欄位，紀錄格式及各欄位意義如下：

```
id：password：userid：groupid：comment：home：shell
```

欄位名稱	意義
id	使用者帳號
password	密碼
userid	使用者編號
groupid	所屬群組編號
comment	個人資料說明
home	家目錄
shell	shell 路徑

例如：

```
vic:x:500:500:vic:/home/vic:/bin/bash
joan:x:500:500:joan:/home/joan:/bin/bash
```

基於安全的理由，密碼的部份會予以編碼，並且存入 /etc/shadow 中，該 shadow 檔只有 root 權限才能讀取。

群組檔為 /etc/group，與密碼檔相同是由許多筆組別記錄組成，每一筆記錄有 4 個欄位，紀錄格式及各欄位的意義如下：

```
group：password：groupid：members
```

欄位名稱	意義
group	群組名稱
password	密碼(一般不顯示)
groupid	群組編號
members	所屬成員列表(以逗號分隔)

例如:

classmate::502:vic, joan

7.4 檔案權限管理

Linux 上每一個檔案都有所謂的所有人名稱(Owner)、群組名稱(Group)以及檔案的使用權限等。因此，每一位系統的使用者都有權力，適當的修改自己家目錄下的檔案權限，以維護私密檔案的安全。

對 Linux 來說，檔案的存取權比你只是擁有檔案、目錄存取權要來的重要。存取權不但可以決定哪些人或者哪些群組能夠存取檔案，同時也能夠決定檔案的型態。接下來將介紹三個檔案權限管理的指令：

1. chown

chown 顧名思義，就是改變檔案的擁有者，當你改變完檔案擁有者後，你將不再具有該檔案的任何權限。

語法：chown　user　file/directory_name 範例： $ chown　vic　test.txt

2. chgrp

chgrp 指令在改變檔案所屬的群組，其用法與 chown 相同。記得當你改變群組後，要注意一下檔案的權限是否需要修改。

語法：chgrp　group_name　file/directory_name 範例： $ chgrp　classmate　/bin/shutdown

3. chmod

chmod 是修改檔案權限的指令。chmod 指令需要給定一個權限參數，這權限參數有兩種形式：

◆英文表示法

英文表示法就是一般我們執行“ls”指令時，每一列最開頭的 10 位。第一位代表此為一連結檔案(l)或目錄(d)，接下來的 9 位三個三個一組，每組的三位分別以「r」、「w」及「x」代表，「r」

代表可讀,「w」代表可寫,「x」代表可執行。前三位是使用者(User)本身的權限，以「u」表示；中間三位是與使用者同一群組(Group)之使用者的權限，以「g」表示；最後三位則是其他使用者(Other)的權限，以「o」表示。

◆數字表示法

數字表示法比英文表示法簡單。英文表示法中的三組，在這裡是用數字代替。每一組中的三位以「0」或「1」代替，以二進位來看，有權限為1；沒有為0。因此，最大為7 (111)，最小為0 (000)。

第8章 X Window System

8.1 簡介

GUI (Graphical User Interface，圖形使用者介面)的理念，最早是來自Xerox (全錄)的實驗室，而 Apple (蘋果)公司在 1984 年初時，首先將這概念使用在其產品- Mac(麥金塔)電腦上。這種完全圖像式的使用介面，徹底顛覆人類使用電腦的方式。同年，MIT (Massachusetts Institute of Technology，美國麻省理工學院)在雅典那 (Athena)計畫中發展出了 X-Window System 的第一個版本—X1。

1987 年舉行的一次 X 技術研討會中，許多工作站的銷售商(如：SUN、IBM、HP)發表聲明支持 X Window System 成為工作站標準的圖形介面。同年，X 第 11 版(X11)發表，此版本對於 X Window System 已經有了完整且妥善的規範，此後的 X 版本也都已 11 為基礎稍作修正。

在 Linux 出現之前，X Window System 早已發展了一段時間，雖然早有計畫要將 X Window System 移植到 x86 架構的 UNIX 系統，但由於不夠穩定，直到 1992 年才由四位程式開發者改善原有的成果，成為 Xfree86 計畫，Linux 上的 X Window System 這才誕生。

X windows 有下列幾種別稱：

- X
- X window system
- X Version 11
- X Window System, Version 11
- X11

8.2 X Window System 的特色

由於一開始 MIT 在設計 X 時便以網路為設計基準，並以分散式架構為目標，因此，根據這種理念所實作出來的視窗系統自然結合了網路、分散式使用以及跨平台使用的諸多優點，主要可分為四點：

1. 無特定使用介面

   X Window System 對於使用者介面採用完全開放的政策，只定義了

關於圖形的一些基本原則，並不提供任何圖樣。因此，如果你單純啟動X Server(稍後會介紹)，只能看到原始的視窗，這個環境中的視窗不能移動位置，不能改變大小及形狀，也無法被圖像化，沒有任何其他如標題列等東西，甚至連基本的外框都沒有。這些外加的功能修飾，需靠介面程式自行加入，所以你還不須選擇一個介面程式來執行，而這介面程式就是所謂的「Window Manager」。

Window Manager 介於 X 應用軟體和 X server 之間，負責控制每一個 X 視窗在螢幕上的特性和動作，提供一個良好的使用者介面。Window Manager 的基本功能如下：

- 建立應用程式視窗
- 移動視窗
- 縮放視窗

許多市面上的 Window Manager 軟體除了以上的基本功能之外，還提供了一些其它的擴充功能：

- 彈跳式選單
- 虛擬工作區間(virtual workspace)

由於開放的政策使然，每個 Window Manager 的外觀，甚至操作方式也往往大不相同，如果不習慣，大可執行另一種 Window Manager。也因為如此，你可以建立屬於自己風格的 X Window。

目前最常見的 Window Manager 有下列幾種：

- Enlightenment
- WindowMaker
- AfterStep
- BlackBox
- IceWM

坊間書籍對於這幾種視窗管理程式有許多詳細的介紹，因此在這就不詳加說明。

2. Client/Server 主從式架構

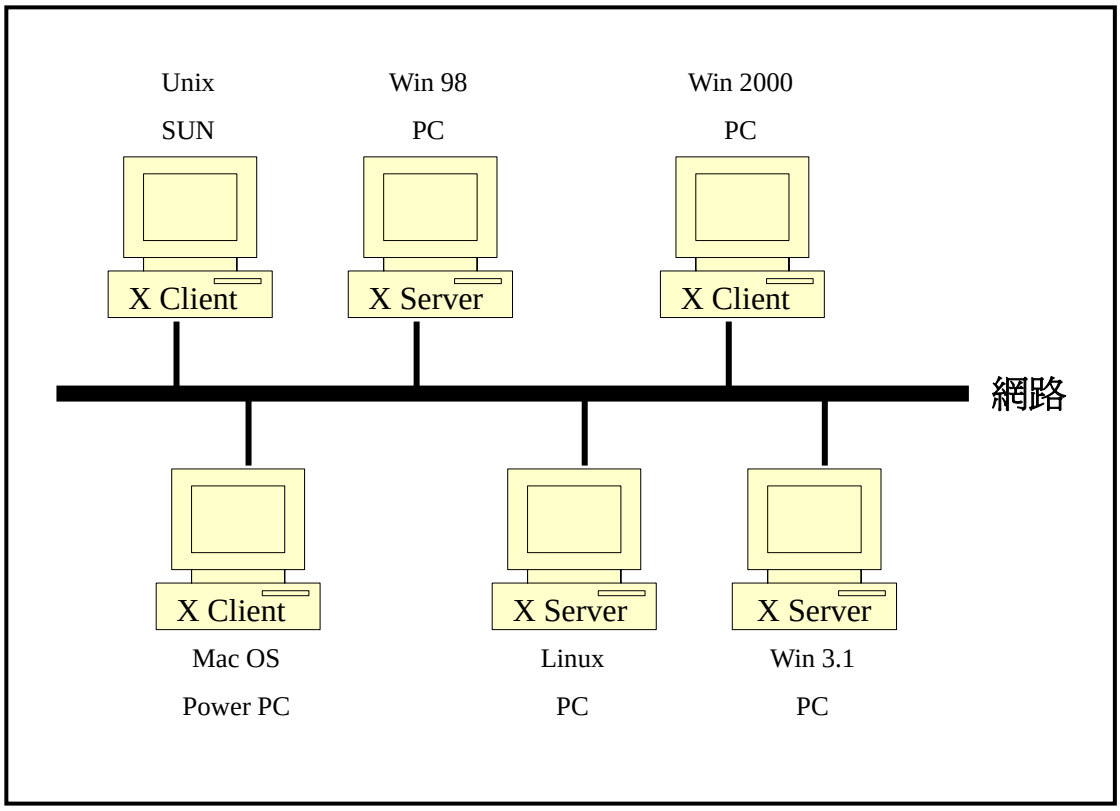
主從式架構是 X Winodw System 最大的特色，這樣的設計，使得 X Window 能夠悠遊於不同作業平台間，盡情發揮遠端執行、資源共享等眾多優點，這是其他視窗系統難以達成的。

X Window 主要可分為兩部份：X Server 與 X Client。X Server

是 X Window 的主要核心，它包括了啟動 X Server 所需的函式庫、執行檔、字型…等等，X Server 控制了實體的顯示器和輸出入裝置如鍵盤和滑鼠等。它的主要工作有資源的管理及控制，事件的處理，錯誤訊息的處理等。X Server 取得資源後便會依照 client 的要求加以回應，通常是將其顯示在螢幕上，例如使用不同的字型，改變顏色等。X Client 就是 X Window 中的應用程式，它會要求 X Server 執行各種特定的動作，如將視窗背景顏色改成綠色等等。很多其它視窗中屬於系統的功能在 X Window 中都只是一個 client，例如前面提到過的視窗經理，以及桌面經理(desktop manager)，檔案經理(file manager)等。由於 X Window 系統的這種特殊架構，使得軟體發展者有了更大的彈性空間。

3. 跨平台使用

因為是主從式的架構，X Window 的 Server 與 Client 各斯其職，X Client 只須通知 X Server，X Server 自動會幫 X Client 完成。基於這樣的特性，只要兩者之間使用相同的通訊協定(X Protocol)，應用程式便可跨平台的使用。

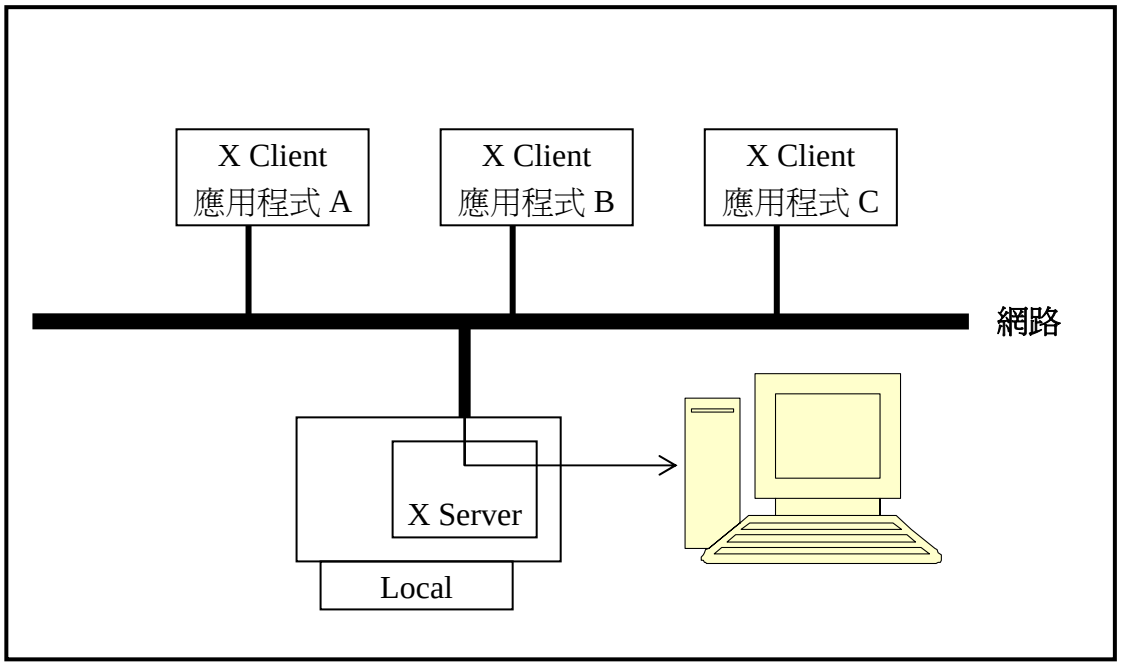


跨平台使用[X Window 徹底研究]

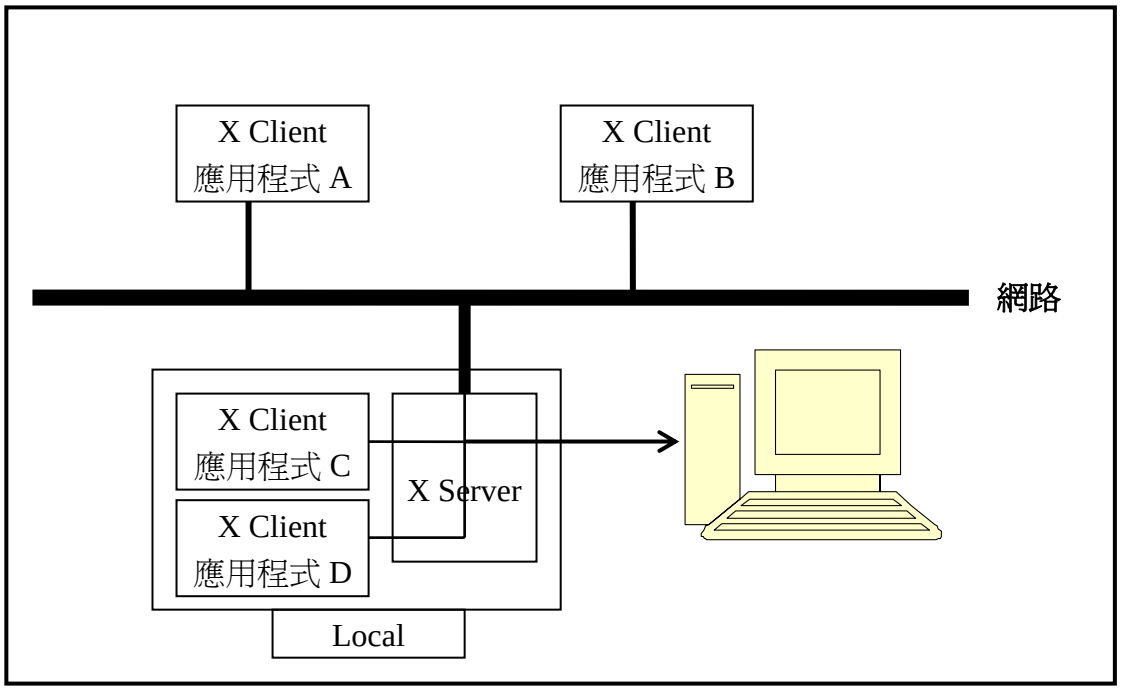
4. 網路透通性

X Client 可以將執行結果同時顯示在多部 X Server 上，例如老師上課時可在每位同學的螢幕前顯示示範圖形。當然也可以執行遠端的應

用程式，並將結果顯示在自己的電腦上，完全感覺不出是在使用遠端的應用程式，這樣的特性即是「網路透通性」。



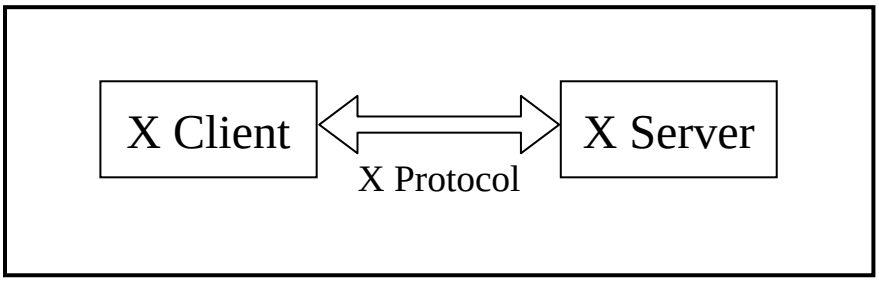
顯示遠方程式輸出[X Window 徹底研究]



同時顯示遠方和區域的程式輸出[X Window 徹底研究]

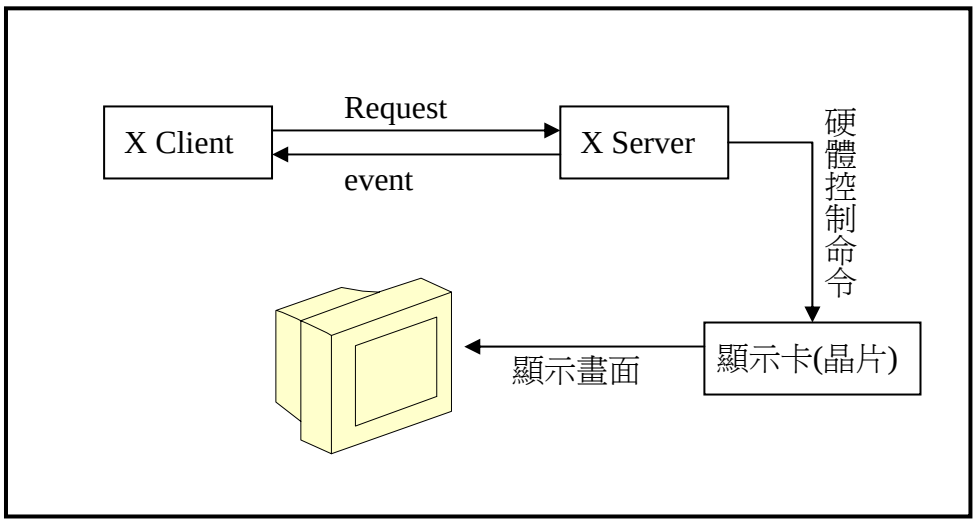
8.3 X Window System 的運作方式

前一段曾經提到 X Window System 採用了 Client/Server 架構，主要包括 X Server 與 X Client 兩部份。事實上，除了這兩部份還有另一個重要的角色：X Protocol，X Protocol 主要是負責 X Server 與 X Client 間的溝通。



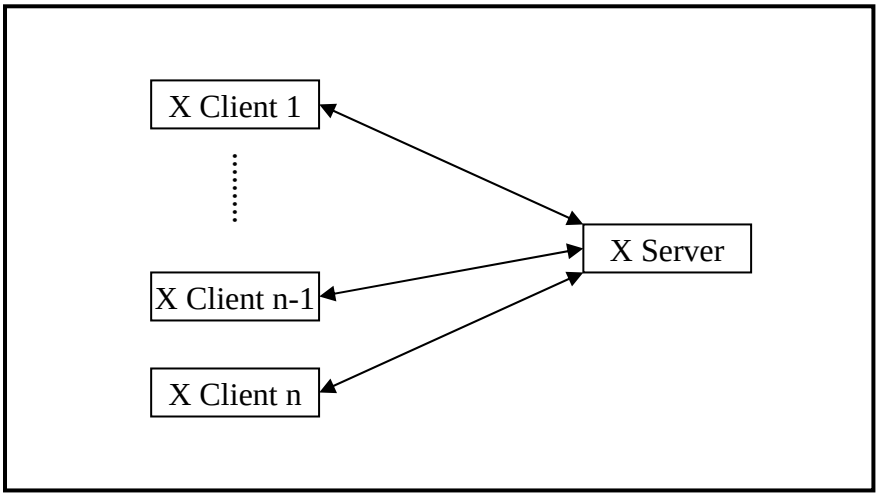
溝通管道［X Window 徹底研究］

X Server 控制所有硬體資源，是輸出入的程式，它接受輸入裝置的訊息，再把訊息傳送給 X Client。相反的，X Client 所傳送過來的訊息，透過 X Server 適當反應後，輸出到輸出設備。



運作模式［X Window 徹底研究］

X Server 傳送給 X Client 的訊息稱為 event，其內容主要是經由使用者操作所產生的變化。而 X Client 傳送給 X Server 的訊息則稱為 Request，主要是 X Client 在經過運算後，要求 X Server 改變現有視窗狀態，如建立視窗、改變視窗大小、輸出…等等。因為具有多工的特性，X Server 可同時接受多個 X Client 應用程式的 Request，每個應用程式都可以有自己的視窗，且同時執行。



同時接受多個 request［X Window 徹底研究］

8.4 整合式作業環境

整合式作業環境與視窗管理程式(Window Manager)最大的不同在於，視窗管理程式的功能較為陽春，而整合式作業環境提供了較有整體感與親和力的介面，且功能更加的完善且出色。目前 Linux 套件大多包含有兩種主要的整合式作業環境 GNOME（GNU Network Object Model Environment，GNU 網路物件模型環境）與 KDE（K Desktop Environment，K 桌面環境），本節將以介紹 KDE 整合式作業環境為主。

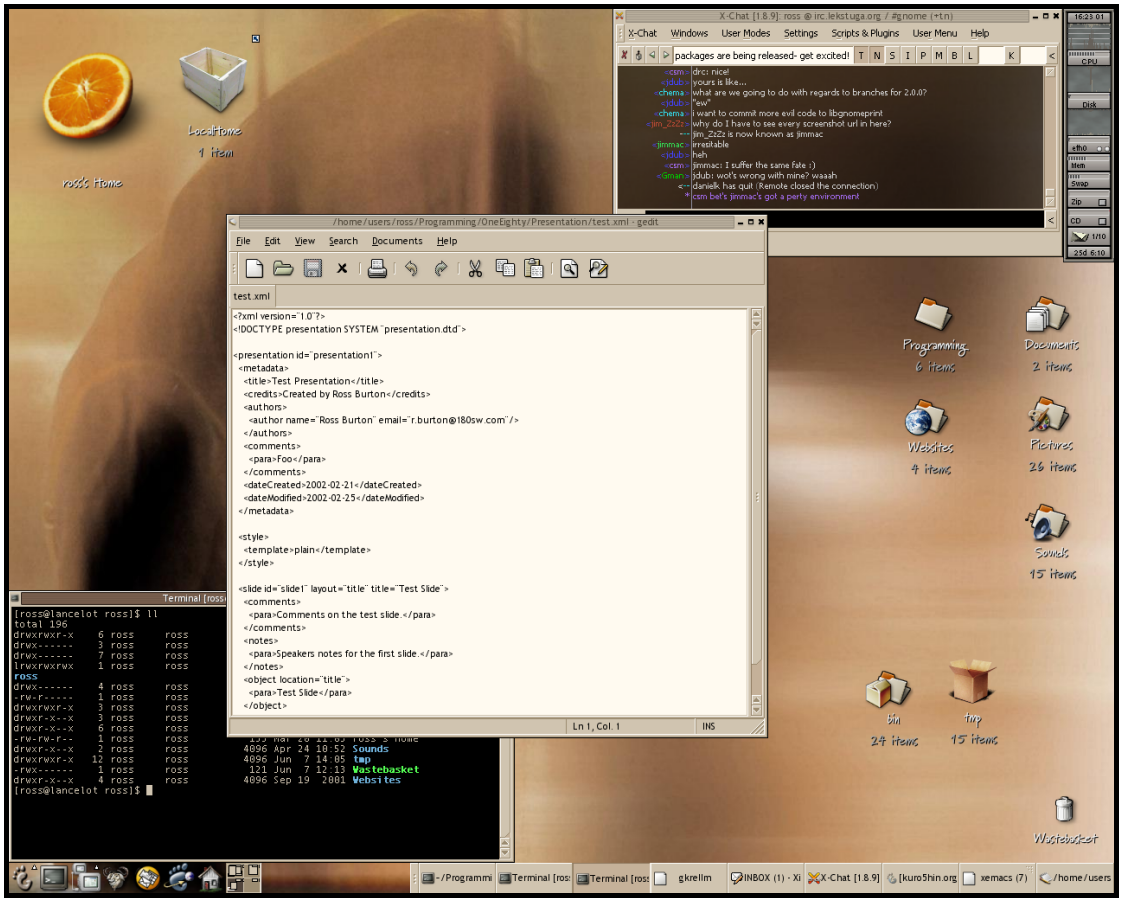
8.4.1 GNOME

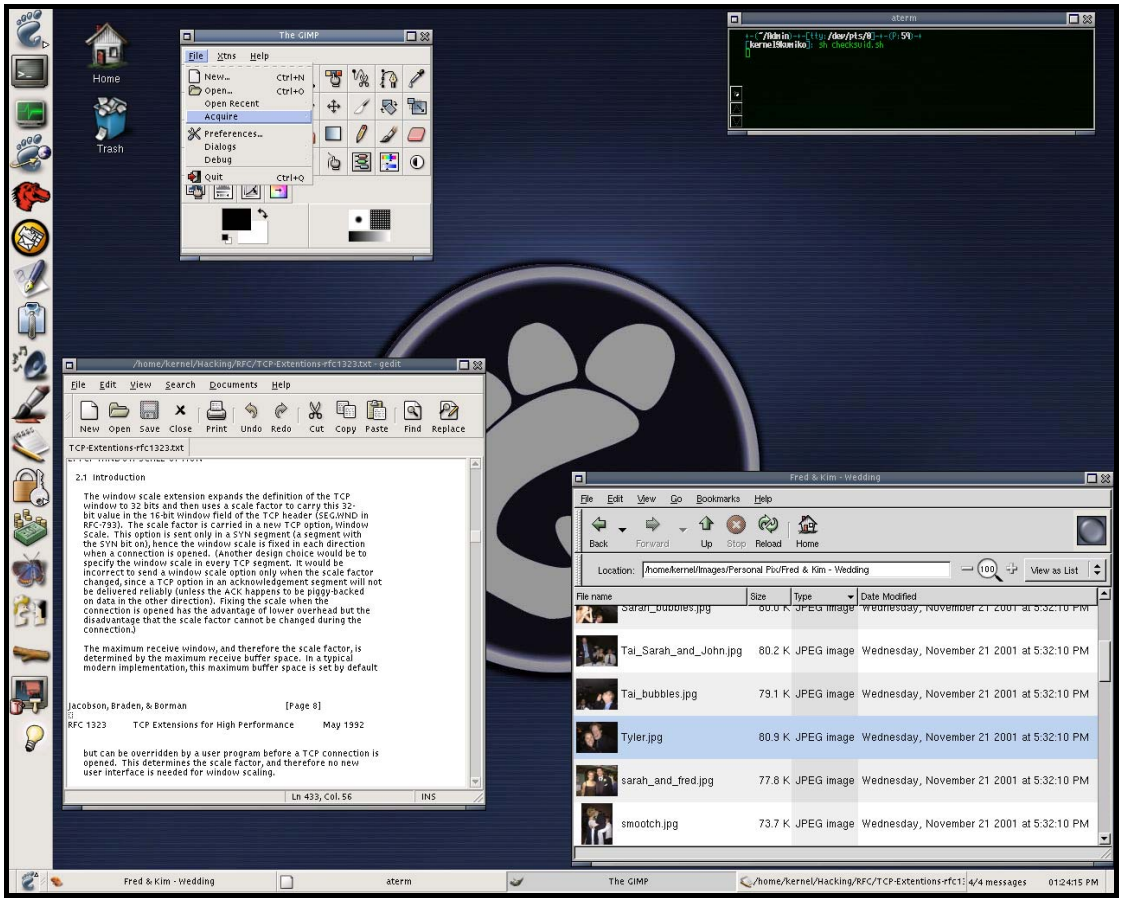
GNOME 是一個有良好使用者介面的桌面環境，用戶可以很容易的使用和配置他們的機器。GNOME 包括一個面板(用來運行程式和顯示機器狀態)，一個桌面(資料和應用程式放在其上)，一套標準的桌面工具和應用程式，一套是應用程式間合作和保持一致性的規範。其他作業系統和環境的用戶應該對 GNOME 提供的圖形驅動的環境感到很熟悉和易於使用。

GNOME 是由分佈於世界各地的程式師所寫的完全開放原始碼，對於用戶來說，有許多優點。GNOME 是高度可配置的，這使得你可以按照你的想像與意願來配置你的桌面。GNOME 的場景管理器可以記住以前的配置，所以只要你按照你喜歡的方式設置好一切事物，它們將保持成你所喜歡的樣式。GNOME 支援多種語言，而且你可以在不改變軟體的情況下，增加對更多語言的支援。GNOME 甚至支

持好幾種拖放協定，這樣可以獲得與 GNOME 不相容的程式間的最大的可相互操作性。

從開發者角度來看，GNOME 也具有一系列的優點，這些優點間接的也將有利於一般的用戶。開發者不需要購買昂貴的許可證來使得他們的商業程式相容於 GNOME。事實上，GNOME 是賣主中立的，介面的任何控制項不受哪一個公司的控制，修改和重新發行是不受限制的。GNOME 應用程式可以用許多種語言進行開發，所以你可以不必拘泥于某種單一的語言。GNOME 使用 CORBA，這就允許軟體的控制項之間可以無縫地互相操作，不管軟體控制項是用什麼語言開發的，甚至也可以不管是運行在什麼類型的機器上。最後，GNOME 可以運行在一系列類 Unix 的作業系統上，包括 Linux。GNOME 是 GNU Network Object Model Enviroment，因此 GNOME 是更大的 GNU 工程的一部分。GNU 工程開始於 1984 年，目的是為了發展一套完全免費的類似 Unix 的作業系統。





兩組不同風格的 GNOME [http://www.gnome.org]

8.4.2 KDE

XteamLinux 開發小組認為 KDE 的表現最為穩定，而且，由於其介面酷似 windows95/98 環境，也比較適合國內用戶的使用習慣，因此 XteamLindows 中文版採用了最新了 KDE 版本作為標準的中文圖形用戶介面。

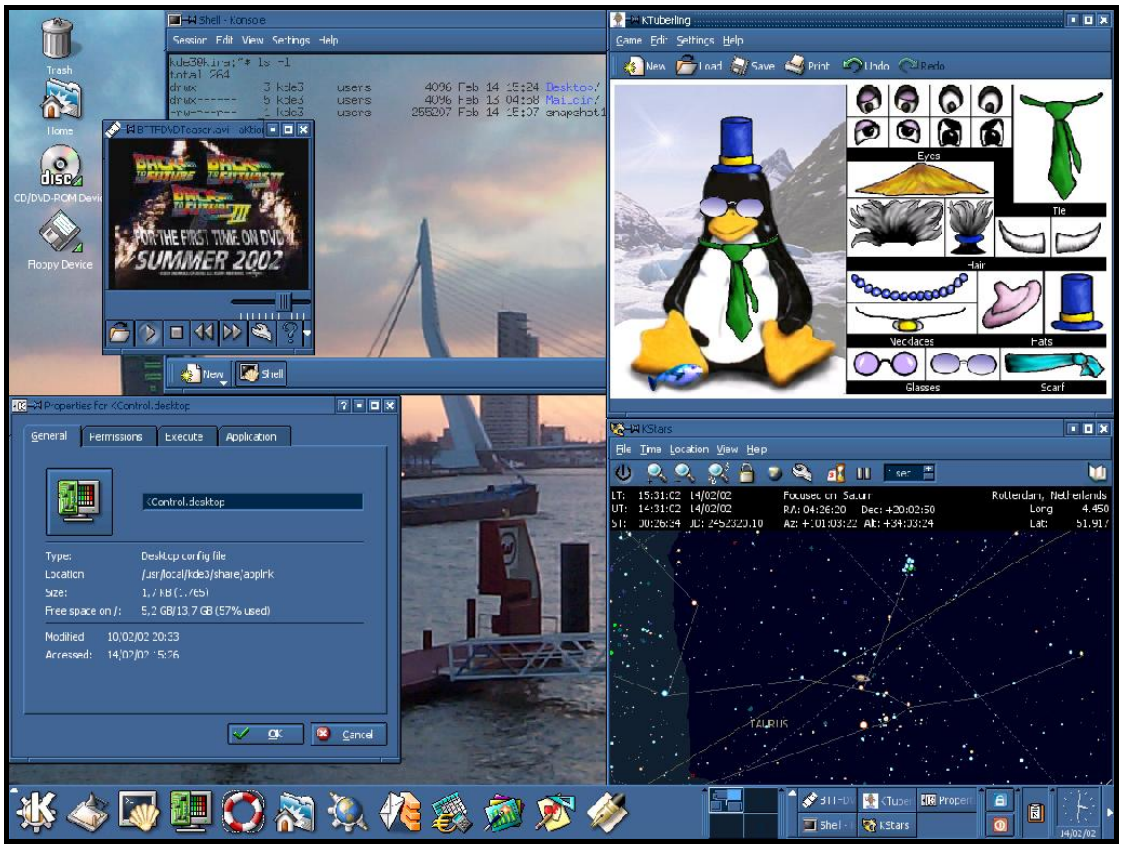
KDE，是指「K 桌面環境」。這個計畫開始於 1997 年，並且始終以其快速的發展速度使世界吃驚。KDE 完全免費，而且安裝容易。除了因為使用了有點商業化的 QT 工具箱而受指責外，幾乎所有的 Linux 發佈版本都包括 KDE 運行環境。KDE 為 Linux 的使用者提供了極大的便利，KDE 的一名開發者有一個三歲的女兒，她正是由於 KDE 而使用 Linux。

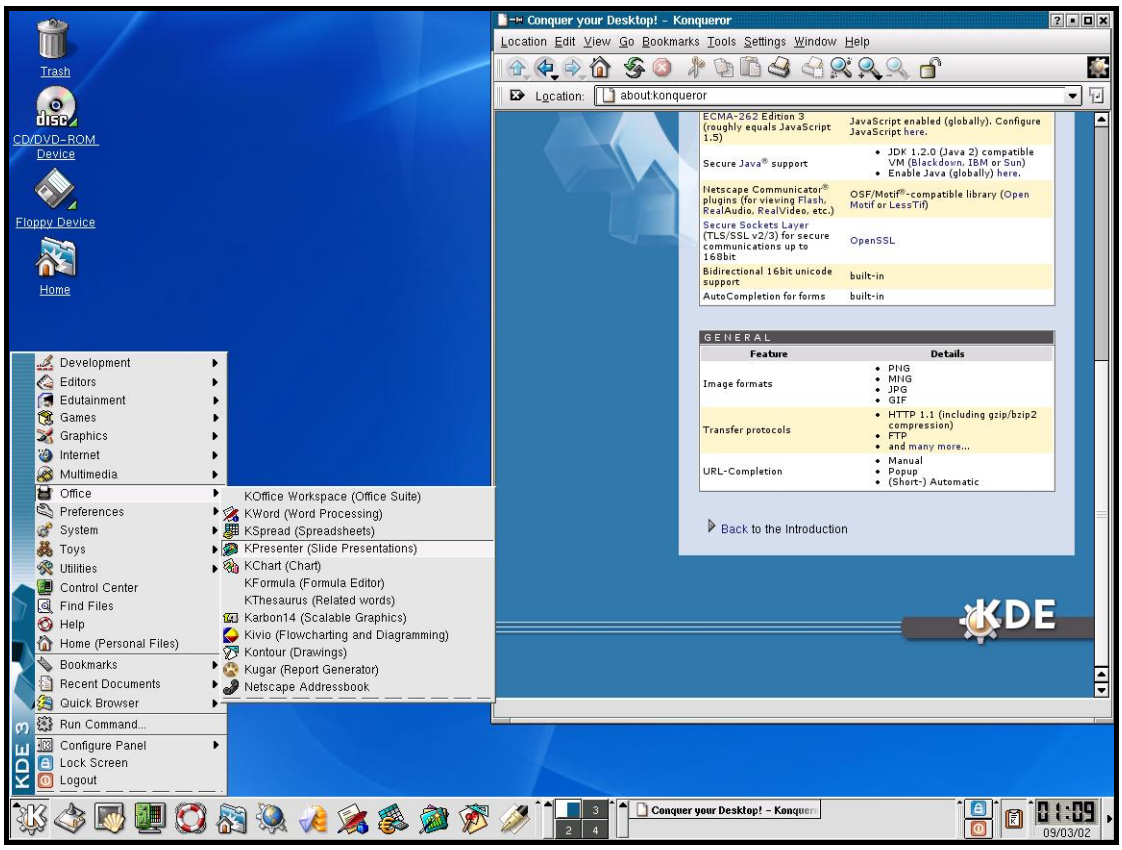
KDE 是 K Desktop Environment 的縮寫。按 KDE 網站 (http://www.kde.org)上的說法，是一個“用於 UNIX 工作站上的強大的圖形工作環境”，它包含了大量的 UNIX 應用程式，甚至象 Win98 一樣內置了一個檔

管理/Web 流覽合一的流覽器。怪不得有人說 KDE 是 PC 機上所能見到的最漂亮的圖形介面了。KDE 是完全免費的，並且提供全部原始程式。

使用 KDE 時，面板上的“開始”功能表、視窗修飾(易於修改)都不時地使你想起視窗系統。一個給人印象最深的特性是 KDE 的拖放功能。你可以用檔管理器 FTP 到一個站點，並且拖動一個檔到你的 PC 上來，或者拖動一個檔到 FTP 站點上去。你也可以刪除一個 FTP 站點上的檔，或者編輯一個文字檔案。KDE 還允許你在 web 或 KMail 用戶端拖動鏈結。

另一個便利的特性是磁碟導航器(Disk Navigator)，它是一個有點象歷史記錄的檔夾，但是有一些附加特性。使用這些特性可以使你流覽你的檔案系統或者重新裝入最近使用過的項目，甚至從你的桌面上裝入項目。KDE 使用起來是如此的簡單，簡便並不意味著缺陷，KDE 包含 60 多個程式以及一個混音器，和許多使用工具，它們均包含在一張 CD 上發佈。它本身就是一個強大的工具來簡化你的工作。





兩組不同風格的 KDE [http://www.kde.org]

當你作為單用戶打開 Linux 並且運行了它的桌面應用程式時，實際上你已經使用了 KDE 的圖型環境。一般情況下你會在同一時間內在你的桌面上開啟多個應用程式。即使你能通過使用 Alt+Tab 鍵，或者任務欄內的按鈕來在這些應用程式中進行切換，使你的桌面保持非常有條理的，也是有一定難度的。KDE 提供了多重的桌面(虛擬桌面)使你能夠在每一個桌面上放置不同的應用程式，對你的工作按照你的需要分類。所有這些桌面是一個 KDE 的組成部份，它們能夠使你比較容易的對你的繁忙的工作空間進行視覺化安排。

在一般的情況下，KDE 包括四個桌面。你可以點擊面板上的一個桌面按鈕來使用這些桌面(它們被命名為一、二、三、四)按 Ctrl+F1 來選擇桌面"一"等等，Ctrl+F4 鍵代表桌面"四"。 當你回復到一個桌面，所有的應用程式將會與你離開這個視窗切換到另一個視窗時一樣。你可以將應用程式視窗移動到任何桌面，當然，你也可以改變 KDE 桌面的數量和名字。

8.4.3 KDE 應用程式

KDE 內建許多的應用程式，大致上分為下面幾類：

1. KDE 的基本應用程式
2. KDE 提供的系統管理程式
3. KDE 實用工具
4. KDE 提供的遊戲
5. 影像處理程式
6. KDE 網路程式
7. KDE 的多媒體程式

究竟 KDE 安裝了哪些應用程式呢？下面一一列表為你說明：

1. KDE 的基本應用程式

軟體名稱	功能描述
Kaudio	Audio Player
Kbgndwm	KWM 的背景桌布模組
Kcontrol	Central control panel
Kdehelp	線上幫助
Kdm xdm	replacement
Kfind	圖形介面的 find
Kfm	有 Web Brower 功能的檔管理員
Kfontmanager	字體管理程式
Kmenuedit	可讓你很容易設置工作欄應用程式選單的編輯工具
Kpanel	桌面平臺程式
Krootwm	給根視窗(root window)使用的 KWM 模組
Kscreensaver	螢幕保護程式
Kvt	視窗終端模似器(和 xterm、rxvt 一樣)
Kwm	KDE 的 Window Manager
Kwmcom	通信工具，即圖形化的 minicom
Kwmpager	Pager module for KWM

2. KDE 提供的系統管理程式

軟體名稱	功能描述
Kuser	用戶管理程式
Ksysv	編輯 Sys-style init 設置檔的程式

3. KDE 實用工具

軟體名稱	功能描述
Karm	時鐘

Kcalc	計算器
Kedit	編輯器
Kfloppy	可將磁片 format 成 dos 或 ext2 格式
Khexdit	16 進制碼編輯器
Kjots	Notes taking utility
Ktjettool	印表機設置程式
Knotes	郵件程式
Kzip	ZIP 格式壓縮

4. KDE 提供的遊戲

軟體名稱	功能描述
Kabalone	Puzzle
Kasteroids	Asteroids game
Kmahjongg	麻將
Kmines	踩地雷
Kpat	Patience card game
Kpoker	Poker game
Kreversi	Reversi (Otello) game
Ksame	Same game
Kshisen	Shisen
Ksnake	蛇行刁手
Ktetris	俄羅斯方塊

5. 影像處理程式

軟體名稱	功能描述
Kdvi	可看 TeX DVI 檔格式的流覽器
Kfax	可看 Fax 文件
Kfract	可畫幾何圖形的軟體
Kghostview	類似 ghostview 的 PostScript Viewer
Kpaint	類似 xpaint 工具
Kview	程式 可看 Gif/Jpeg 圖檔的

6. KDE 網路程式

軟體名稱	功能描述

Kmail	電子郵件軟體
Knu	網路工具，如 ping、traceroute
Korn	電子郵件信箱
Kppp	PPP 撥接程式
Krn	NEWS 流覽器

7. KDE 的多媒體程式

軟體名稱	功能描述
Kmedial	media player
Kmid	midi/karaoke player
Kmidi	midi to wav player/conveter
Kmix	混音器
Kscd	CD player 和 xmcd 類似

第9章 網路

9.1 簡介

Linux 作業系統支援強大的網路功能，在你安裝完系統後，不需作什麼特別的設定，可能已經有好幾個伺服器跑起來了。本章會介紹幾個較常用的網路相關指令，以及 Linux 系統對網路的支援。

9.2 網路相關指令

Linux 有著強大的網路功能，幾個網路相關的常用指令也非常的實用，關於指令的詳細用法，請使用 man 或下給指令“--help”參數：

1. ping

   ping 一般主要是用來測試本機是否與網路上的某一台電腦相通，也可作為效能偵測工具。

2. traceroute

   traceroute 用來偵測本機連到其他主機時，封包所走的路徑中，經過了哪些路由器，及其延遲時間。

3. mtr 與 xmtr

   mtr 是一個綜合 ping 與 traceroute 功能的指令，它會一直傳送封包給本機到連線主機之間的所有路由器，並不斷地更新結果，xmtr 則是在 X Window 下具有圖形介面的版本。

4. finger

   Finger 是一個可以傳回電腦上有註冊的使用者資料的程式。如果使用 "finger vic" 那麼它就會顯示出您目前使用的主機上，vic 的個人資料。另外 "*finger vic@mail.im.tku.edu.tw*" 則會顯示在淡江大學資管研究所中某台主機上的 vic 使用者的資料。

5. nslookup

   nslookup 是個查詢主機網域名稱及 IP 位址的指令，除了顯示所要查詢的結果外，還會顯示是經由哪一台名稱伺服器查詢得到的結果。

6. netstat

netstat 是顯示目前本機所提供網路服務的狀態。

9.3 Linux 的網路通訊協定

Linux 支援許多(幾乎全部)不同的網路通訊協定，如：

1. TCP/IP 通訊協定

TCP/IP 通訊協定，採用階層式的結構，以便將應用程式與網路硬體隔離開來。雖然他的設計理念是基於階層式的模型，但是他的重點放在提供互連性，而不只是死板地遵循層級功能。這就是為什麼 TCP/IP 通訊協定堆疊，會成為一個網路互連通訊協定的非官方標準，而與 OSI 國際標準相抗衡。

Linux 從一開始就已經提供 TCP/IP 的網路能力，成為一個穩固、快速、和可靠的軟體實作，同時也是 Linux 成功的關鍵因素之一。

2. IPV6

IPv6，下一代網際網路通訊協定，是 IPv4 通訊協定的升級版本，主要是用來解決定址上的許多問題。這些問題包括：可用的 IP 位址不足、缺乏處理即時性資料流的機制、缺乏網路層級的安全控制...等等。採用擴充定址法之後，就能增加 IP 定址的空間(IPv6 的位址長度是 IPv4 的四倍)，同時對路由選擇的效率而言有著極大的影響。

IPv4 採用的是分級式定址法，依網路的大小規模分級成 ClassA、B、C 沒有彈性，造成路由表的爆增；而 IPv6 採用的是分類式定址法，僅區分使用類型的範圍，其餘依實際需要以 CIDR 方式分配，讓位址空間得到更有效率的分配，並能減緩路由表的成長。

3. IPX/SPX

IPX/SPX(網際網路封包交換/循序封包交換)是由 Novell 公司以"全錄(Xerox)網路系統"(XNS)通訊協定為藍本，發展出來的專屬通訊協定。IPX/SPX 通訊協定在 1980 年代初期很有名，成為 Novell 公司 NetWare 產品的代名詞。NetWare 成為第一代區域網路，網路作業系統(NOS)的非官方標準。Novell 公司同時也為他們的網路作業系統，加上商業應用程式套件與用戶端網路連接工具。

Linux 對 IPX/SPX 通訊協定有非常完整的實作，使得他能夠被設定成：

- IPX 路由器(router)

- IPX 橋接器(bridge)
- NCP 用戶端且/或 NCP 伺服器端(檔案共享)
- Novell 列印用戶端，Novell 列印伺服器端

以及去：

- 開啟 PPP/IPX 通訊協定，讓 Linux 成為一個 PPP 的伺服器端/用戶端
- 藉由 IP 隧道(tunnel)，讓二個跑 IPX 通訊協定的網路，能夠透過唯一的 IP 路徑連通。

4. AppleTalk

Appletalk 就是"蘋果牌電腦"網路互連通訊協定群的代名詞。他採用點-對-點對等式的網路模型，並提供基本的網路功能，例如檔案及印表機的共享。每部機器可以同時成為用戶端與伺服器端，但是每部"蘋果牌電腦"都得裝上必需的軟硬體才行。

Linux 提供有全功能的 Appletalk 網路功能。Netatalk 就是一個核心層次的 AppleTalk 通訊協定堆疊實作，基本上他是由 BSD 版本衍生出來的系統。他能支援 AppleTalk 的路由選擇，透過 AFP(AppleShare)提供 Unix 和 AFS 檔案系統的服務，提供 Unix 印表機服務，以及透過"印表機存取協定"(PAP)存取 AppleTalk 印表機。

5. 廣域網路(WAN)通訊協定：X.25、FrameRelay…等

許多第三協力廠商提供 Linux 使用之 T-1，T-3，X.25 以及 FrameRelay 等產品。一般而言，這類的連線需要特別的硬體。廠商除了提供硬體之外，也會提供通訊協定驅動程式的支援。

6. ISDN 通訊協定

Linux 的核心有內建的 ISDN 能力。核心模組 Isdn4linux 可以控制 ISDNPC 卡，並且可以將之模擬成使用 Hayes 命令集("AT"命令)的數據機。他可能被應用的範圍，從簡單地使用終端機程式，透過 HDLC 通訊協定(內附於裝置中)來連線；到以 PPP 通訊協定，對 Internet 作全功能網路連線，來使用聲音的應用。

7. PPP、SLIP、PLIP 等通訊協定

Linux 的核心有內建的 PPP(點-對-點通訊協定)，SLIP(串列線路使用 IP)，以及 PLIP(並列線路使用 IP)等通訊協定支援。PPP 通訊協定是一般個人使用者，接取其 ISP(Internet 服務提供商)最常用的方法。PLIP 通訊協定則是二部機器時便宜的連線方法，使用並列埠和一個特製的纜線，連線速度可達 10kBps 到 20kBps。

8. ATM 通訊協定

Linux 的核心有內建的業餘無線電通訊協定支援。尤其令人感興趣的是他支援 AX.25。AX.25 通訊協定提供連接導向與非連接導向二種操作模式，使用時不是以自己的方式，作點-對-點的連線；就是載送其他通訊協定，如 TCP/IP 和 NetRom。

結構上他與等級 2 的 X.25 通訊協定類似，但是做了些許的擴充，使得他更適合應用在業餘無線電的通訊環境。

9. 業餘無線電通訊協定：AX.25

Linux 對 ATM 通訊協定的支援，目前只到 pre-alpha 版本的階段。目前有一個實驗性的實作被發表出來，他支援純 ATM 連線(PVCs 和 SVCs)，ATM 網路上跑 IP 通訊協定(Ipover ATM)，ATM 網路模擬區域網路(LAN emulation)，...等等功能。

9.4 Linux 的網路服務

Linux 提供了多種網路服務，這些服務包括有：

1. 電子郵件服務

Sendmail 是 Linux 平臺上，電子郵件伺服程式的非官方標準。他具有穩定和可擴充的特性，經過適當的設定再配合必要的硬體，他就能夠承受上千個使用者的負荷，而不會有任何閃失。當然，還有其他的電子郵件伺服器(也就是 MTA，電子郵件傳遞代理程式)，例如，smail 和 qmail 就是被設計來取代 sendmail 的。

2. 網頁伺服器

市面上所發行的 Linux 版本大多會包括 Apache 伺服器。Apache 是目前網際網路上最多人使用的網頁伺服器，在網際網路上超過一半以上的站台使用 Apache 伺服器，或由其衍生出來的產品。Apache 伺服器的優點包括有，模組化設計、穩定、以及速度。使用適當的硬體與設定，他可以支援到最高的負荷：Yahoo、Altavista、GeoCities、Hotmail 等站台，就是採用 Apache 伺服器的客戶指定規格，製作出來的版本。

3. FTP(File Transfer Protocol，檔案傳輸協定)伺服器

FTP 就是"檔案傳輸協定"(File Transfer Protocol)的簡寫。FTP 伺服器允許用戶端，連線與取回(下載)檔案的要求。目前 Linux 上存在有多種 FTP 伺服器與用戶端程式，他們通常會放在大部分的 Linux 發行版本中。有文字模式的用戶端程式，也有 GUI 模式的。

4. NEWS(網路新聞)服務

Usenet(所謂的網路新聞)是一個大的告示板系統，他上面包含有各種的討論話題，而他採用的是階層式的架構。網路上的電腦經由網際網路以 NNTP 通訊協定來互換文章。Linux 上有多個被實作出來的軟體，有給負荷極重的站台使用的，也有給只接收一些新聞群組(news groups)的小站台使用的。

5. DNS(Domain Name System，網域名稱)系統服務

DNS 伺服器的功能，就是將網域名稱(對人有意義的文字)轉換成 IP 位址。單獨一台 DNS 伺服器，並不會知道世界上所有的 IP 位址；可是他可以向其他伺服器，詢問自己所不知道的位址。DNS 伺服器不是將所詢問的 IP 位址回應給使用者,就是回報所詢問的名稱在資料表中找不到。

在 Linux 上，網域名稱查詢服務是由一個名為 named 的程式來完成的。他是網際網路軟體聯盟(Internet Software Consortium)所提供 bind 套裝程式的一部分。

6. DHCP 與 bootp 通訊協定

DHCP 與 bootp 通訊協定，允許用戶端向伺服器索取網路資訊(例如他們自己的 IP 位址)。有許多組織都開始來使用他，因為他使得網路管理變的容易多了，特別是在大型網路中，或是網路上有很多機動的使用者。

7. 網路資訊服務(NIS)

網路資訊服務(NIS)，提供一個簡易的網路查詢服務，他是由資料庫與處理程序所構成。他的目的就是在提供整個網路上所有必需知道的資訊，給網路上所有的機器。他使得一個使用者，能夠簽入網路上任何一台執行 NIS 的機器，而不需要管理者為使用者，在每台機器上加上密碼，只需加入到主資料庫即可。

8. 認證服務

在 Linux/WindowNT 系列混用的網路中，有各種認證使用者的方法。

9. 遠端執行應用程式服務

Linux 支援以遠端和分散的方式，來執行應用程式，如：

- Telnet

Telnet 是一個程式，他讓人們使用遠端的電腦，就好像是實際在該電腦面前一樣。Telnet 是 Linux 上最強大的工具之一，他使得真正的遠端管理機器成為可能。在使用者的觀點中,他也是個有趣的程式,因為他讓使用者能夠在 Internet 的任何地方，以遠端的方式取用其檔案及程式。將他與 X 伺服器合用，不論是坐在控制操作台(console)前面，或是地球的另一邊，除了時間的延遲外，感覺上並沒有不同。Telnet 伺服器監控程式(daemons)和用戶端程式，在大部 Linux 的發行版本中都可以找到。若想在遠端操作介面(remote shell)的連線期間(sessions)將所有內容加密，可以透過 SSH 的方式，他讓安全的遠端管理工作成為可能。

- 遠端命令

  在 Linux 上，遠端命令方式的出現，讓我們能夠透過操作"介面"(shell)與遠端的電腦溝通。例如：rlogin，讓我們能夠以 telnet 類似的方法，簽入遠端機器；rcp，讓我們能夠與遠端機器之間，做遠端的檔案傳輸...等等。最後順便一提，透過"遠端操作介面"(remote shell)下命令的程式 rsh，讓我們不必實際地簽入遠端機器，就能在該機器上執行命令。

- X-Window

  X 視窗系統於 1980 年代末期在 MIT 被發展出來，他很快地成為 UNIX 圖形工作站的工業標準。這個軟體可以免費取得，他極具通用性，而且適合執行的硬體平臺範圍廣泛。任何 X 視窗系統,由二個不同的部分組成--X 伺服器與 1 或多個 X 用戶端。瞭解伺服器與用戶端間不同之處在那裡，是件重要的事情。伺服器直接控制螢幕的顯示，並且監控所有的輸出入裝置，例如鍵盤、滑鼠或螢幕。用戶端，則正好相反，無法直接取用螢幕-他要透過伺服器，才能來操作所有的輸出入動作。用戶端就是"真正"執行運算工作的地方—執行應用程式或是其他工作。每當用戶端與伺服器連線時，伺服器就會開啟一或多個視窗，以便為該用戶端，操作輸出入動作。
  簡而言之，X 視窗系統讓使用者能夠簽入遠端機器，執行行程(process)，例如開啟一個網頁瀏覽程式，並將其輸出結果顯示在自己的機器上。因為行程實際是在用戶端上執行，伺服器端僅需要非常少量 CPU 計算能力。因此想要設計一部，主要功能純粹作為 X 視窗伺服器使用的電腦是可行的，而他就

是所謂的 X-終端機。Linux 上有免費的 X 視窗系統移植程式套件，該程式套件通常會附在大多數 Linux 的發行版本中。

- VNC(VirtualNetworkComputing，虛擬網路計算作業)

  VNC 基本上是一個遠端顯示系統，讓我們不僅在執行程式的機器上能看到計算作業的桌面環境，而且在 Internet 的任何地方，即使使用各種不同的機器架構，也都能看的到。Linux 以及許多其他的作業平臺，都存在有用戶端與伺服器的程式。你可能會在 Windows 系列的機器上執行 MS-Word 程式，而將輸出結果顯示在 Linux 機器上。反之亦然，你可能會在 Linux 機器上執行應用程式，而將輸出結果顯示在別部 Linux 或 Windows 機器上。你若有一個 Java 的用戶端，你也可以在網頁瀏覽程式中，執行遠端顯示的計算作業。

9.4 Linux 核心對網路的支援

Linux 核心本身對網路的支援也是相當包羅萬象的，這邊舉一些常用的功能：

1. Router 路由器

   Linux 的核心有內建的路由選擇(routing)功能。一部 Linux 機器，可以被建構成一台 IP 或 IPX 路由器他的花費僅是商業路由器的零頭而已。最近發表的核心，包含了一些特殊的功能選項，都是用來設定路由器的：

   - 多目的傳播(Multicasting)：可讓 Linux 機器成為一個，將 IP 封包傳播到多個目的位址的路由器。使用 MBONE 時，就需要這種路由器，MBONE 是 Internet 上，一種需要高頻寬的網路，他能夠載送聲音和影像的廣播信號。
   - 策略性 IP 路由選擇(IP policy routing)：一般路由器處理所收到的封包時，僅以封包的最終目的位址為路由選擇的依據，但是路由的選擇，也可以將來源位址與封包所抵達的網路介面，一起納入考慮。

2. Bridge 橋接器

   Linux 的核心有內建的乙太網路橋接器(ethernet bridge)支援，他的作用就是讓連接過來，不同乙太網區段(Ethernet segments)上面的各個節點，使用起來感覺就像是，在同一個乙太網路上。多部橋接器放在一起，再加上 IEEE802.1 標準的 spanning tree 演算法的使用，

可以建構一個更大的乙太網路。有的程式套件，可以過濾 IP，IPX 或 MAC 位址。

3. IPMasquerading，IP 偽裝

IP 偽裝在 Linux 上是一個發展中的網路功能。如果一部 Linux 主機連接至 Internet，而且其 IP 偽裝功能被開啟，則連上他的其他電腦(不論是在相同的 LAN 上，或是透過數據機連上來的)就算是他們沒有使用正式分配的 IP 位址，都同樣可以通達 Internet。他降低了上網的費用，因為可以多人使用同一條數據機連線來上 Internet，同時他也增加了安全性，從某些方面來看，他的功能像是一個防火牆，因為外界網路無法連接，非正式分配的 IP 位址。

4. IPaliasing，IP 別名

這個 Linux 核心所提供的功能，使得我們可以在同一個低階網路裝置的驅動程式下，設定多重的網路位址(例如，在一片乙太網路卡裝置上，設定二個 IP 位址)。通常我們會依照，伺服器程式所監看網路位址的不同，而來區分不同的服務功能，如：多重主機(multihosting)、虛擬網域(virtual domains)、虛擬主機服務(virtual hosting services)。

5. FireWall，防火牆

防火牆是一個將私有網路，從公眾範圍(整個網際網路)保護與獨立出來的裝置。他的設計使他能夠依據每個封包所含之來源位址，目的位址埠，以及封包形態等資訊，來控制封包的流通與否。Linux 上存在有不同類型的防火牆工具套件，同時核心也有內建的防火牆支援。除了核心內建的支援外，還有 TIS 和 SOCKS 二種防火牆工具套件。這二種防火牆工具套件非常完整，若能與其他工具合併使用，則可阻斷/重導各類的網路流量與協定。而且經由設定檔案或 GUI 程式，可以實作出不同的網路流量控制策略。

6. PortFowarding，埠轉遞

有互動交談能力網頁站台越來越多了，他們使用 cgi-bins 或 Javaapplets 程式，來存取資料庫或其他服務。因為這類存取方式，可能造成安全上的問題，所以資料庫所在的機器，不應該直接連上 Internet。埠轉遞功能對這類存取問題，提供了一個還算理想的解決方案。透過防火牆，進入到特定埠編號的 IP 封包，可以被改寫，然後轉遞到內部實際提供服務的伺服器上。內部伺服器所回覆的封包也會被改寫，使得他看起來是來自防火牆。

7. VPNChannel，VirtualPrivateNetwork 虛擬私人網路

點對點隧道通訊協定(Point-to-Point Tunneling Protocol，簡稱 PPTP)就是在 Internet 上使用保密的虛擬私有網路(VPN)的一種網路技術，關於這類技術，在 Windows 個案研究中已經有提到，因此不再詳述。目前 WindowsNT 伺服器，已將 PPTP 與遠端存取服務(RAS)伺服器整合在一起。透過 PPTP，使用者可利用電話撥接至當地的 ISP，或直接連上 Internet，來取用自己公司的網路服務，使用起來感覺就好像是，坐在自己的辦公桌前一樣。然而 PPTP 是一個封閉的通訊協定，而且他的保密性最近也正遭受到質疑。

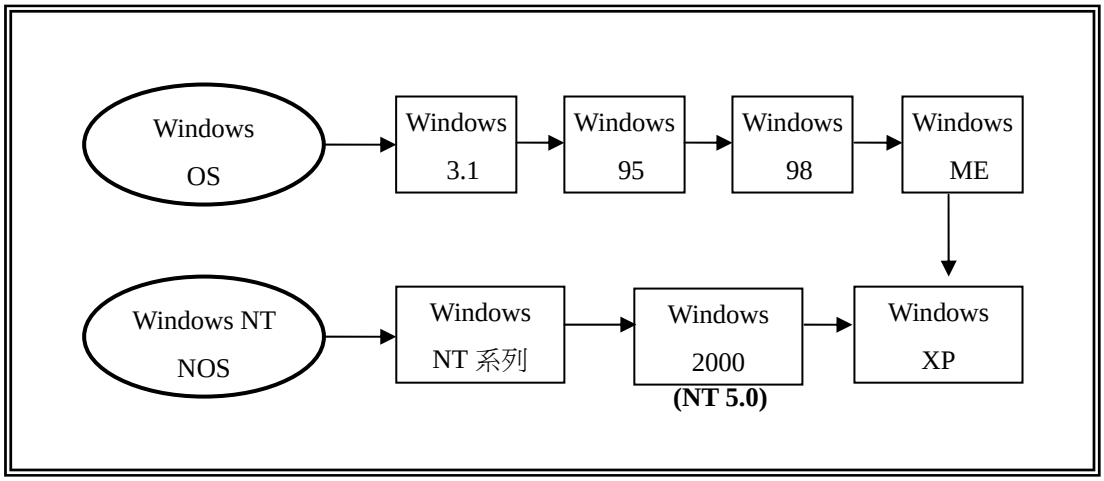
單元三 Windows 作業系統個案研究

第1章 Windows 作業系統的發展

1.1 簡介

Windows 作業系統的誕生，徹底改變了人們使用個人電腦的操作模式。過去 DOS 時代需要熟記指令才能做到的事，現在只要透過滑鼠即可以完成大部分的工作。簡易富親和力的操作介面縮小了學習的障礙，也使得 Windows 作業系統大量且快速的佔有作業系統的市場。未來，圖形化的加強及多媒體的應用整合將更趨明顯，整個 PC 作業環境的介面越來越方便。

Microsoft Windows 家族主要包括了兩大分支，一支是以一般家庭為對象屬於單人多工的 Windows OS（ Operation System，作業系統 ），另一支則是多人多工的 Windows NT NOS（ Network Operation System，網路作業系統 ）主要為企業或有需求者提供各項伺服器的服務。再接下來的幾個單元，我們將以 Windows 2000 作為教學平台，一一詳細的介紹。



1.2 Windows 3.1

1992 年 4 月微軟公司發表了 MS-Windows 3.1 作業系統後，圖形的親和力打破了學習電腦的障礙，更加速了軟體工業的發展。Windows 3.1 算是微軟視窗作

業系統的始祖，號稱「所見及所得」，指的是說使用者在螢幕上看到任何文字、符號及圖形的樣子，和從印表機印出來的結果是一模一樣，這正是圖形導向的最大特色。

1.3 Windows 95/98/ME

1995 年微軟推出了具有多功能力的 32 位元圖形作業系統－Windows 95，其 32 位元的處理能力讓個人電腦的效率有了大幅的改善，也讓多工的環境更加的穩定。

在 Windows 95 成功佔領了個人電腦作業系統的市場後，1998 年微軟順勢發表了強調娛樂性及結合網路的平台 Windows 98，結合了微軟本身的 IE 瀏覽器 Internet Explorer，支援更多更新的硬體裝置，減少開關機的時間，及增加娛樂性的新功能等。

Windows ME (Millennium)於 2000 年推出，是 Windows 98 的升級版本，這一個版本強調的是多媒體的應用，可以用來做數位攝影、音樂、電腦遊戲以及遨遊網際網路。

1.4 Windows NT/2000

NT 代表的是 New Technology，Windows NT 是真正的 32 位元多人多工處理作業系統，可使用於多種 CPU 的電腦系統 (PC、PowerPC、Alpha)，支援先佔式多工及多處理機，能直接執行 Windows 95 的許多應用程式。從 NT 4.0 開始，就有著和 Windows 95 一樣的圖形使用者介面，且在穩定性、安全性及執行效率上，都遠遠的超過 Windows 95，堪稱微軟有史以來最穩定、最強大的作業系統。

Windows 2000 大幅的擴展了 Windows NT 的現有功能，提供穩定性及反應性更加的操作體驗，新增的「加密檔案系統」也能完全確保資訊的私密性及安全性。

下表列出了 Windows 2000 的最新功能：

檔案與列印	磁碟配額(Disk Quotas)：限制使用者磁碟用量，提供更細微的磁碟設備管理，避免備無限量的成長需求。 動態磁碟管理(Dynamic Volume Management)：維護儲存設備不用離線或關機。即使當磁碟機已滿載或失效，企業仍可照常運作。IT 人員可以更主動更有效的管理儲存資源。

	階層式存放管理(Hierarchical Storage Management)：最符合成本效益的資料管理方法，常用資料存放高效能磁碟，不常存取的放在較平價的媒體。 網際網路印表(Internet Printing)：透過指定印表機名稱與適當的授權，可執行跨網際網路的印表。例如，您可將產品新型錄，直接送到出版商的印表機。
網際網路與應用程式	IIS 5.0 與 XML：支援網際網路標準 COM+：元件化程式的整合服務，讓開發人員使用任何語言、任何工具，皆可輕鬆開發與使用軟體元件。 終端機服務(Terminal Services)：無論是何種桌上型電腦，即使不執行 Windows 系統的電腦，仍可執行 Windows 程式，提供一致的電腦環境。也可做為伺服器的遠端控管。 高可用性(High availability)：Windows 2000 Advanced Server 的叢集服務，提供容錯轉移功能，可將正在執行的工作，自動轉到另一台伺服器上，確保運作不中斷。
通訊服務	虛擬私人網路(Virtual Private Networking)：運用點對點通道通訊協定(PPTP)或第二層通道通訊協定(L2TP)，即可透過 Internet 或其他網路安全地連接遠端伺服器，存取其資源。 遠端存取服務(Remote Access Services)：支援多重通訊協定，指定撥號路由器與遠端存取伺服服務，可選擇搭配數百種網路介面卡，提供低成本解決方案。 電話語音(IP Telephony)：可整合使用者資訊與其電話資訊；運用電話語音應用程式介面(TAPI)，與應用程式整合。 RADIUS 驗證(RADIUS Authentication and Accounting)：提供網際網路身份驗證，讓遠端存取使用更安全。
管理基礎平台	遠端管理(Remote Management)：包括透過終端機服務的遠端管理，MMC 工作委派，管理工具的遠端管理，Windows Script Host 與遠端安裝服務 公鑰基礎架構(Public Key Infrastructure)：身份驗證與安全資料傳輸的保證 系統準備工具 (System Preparation Tool)：需要自動部署時，系統管理者可運用此工具製作作業系統映像。

[http://www.microsoft.com/Taiwan/windows2000/promo/function.htm,台灣微軟]

1.5 Windows XP

Windows XP 是微軟繼 Windows 2000 和 Windows Millennium 之後，最新一版的作業系統。它整合了 Windows 2000 的威力（標準安全保護、管理能力和可靠性）與 Windows 98 和 Windows Me 的最佳功能（隨插即用、好用的使用者介面、以及創新的支援服務），聚集了所有的 Windows 作業系統，共同建立前所未有的最佳視窗作業系統。 Windows XP 是以加強型的 Windows 2000 程式碼庫為基礎，針對家庭使用者和商業使用者分別撰寫不同的版本：Windows XP 家庭版和 Windows XP 專業版。

第2章 網路作業系統

2.1 設計原則

NT 的設計原則包括了擴充性、可攜性、可靠性、相容性、效能及國際性，兼顧了系統的未來發展與實用性。

擴充性 (Extensibility)，科技的發展一日千里，為了要維持競爭力與實用性，NT 使用了階層式的架構來實現它的擴充性。階層式的架構中，每個階層負責不同的處理，核心模式中的核心、管理程式提供系統基本的服務，而在使用者模式中的許多伺服器子系統則負責使用者與系統間的溝通，子系統的模組化架構使得不論是新增或移除子系統，都不會影響到核心模式中的管理程式。為了不影響系統的運作，NT 的 I/O 系統使用的是載入式驅動程式，如此一來，系統在運作時也可以隨時增加新的網路、檔案、I/O 元件，而不會中斷系統的服務。

可攜性 (Portable)，當硬體已經無法負荷系統提供的服務，出現了瓶頸，需要將系統移轉至其他硬體架構時，可攜性便顯得重要。為了具有可攜性，NT 設計了動態鏈結程式庫 (Dynamic Link Library)，它其實就是位於核心模式中的硬體抽象層 (Hardware Abstraction Layer，HAL)，由硬體抽象層直接處理硬體，將 NT 的其餘部分與執行平台硬體的差異性隔離。

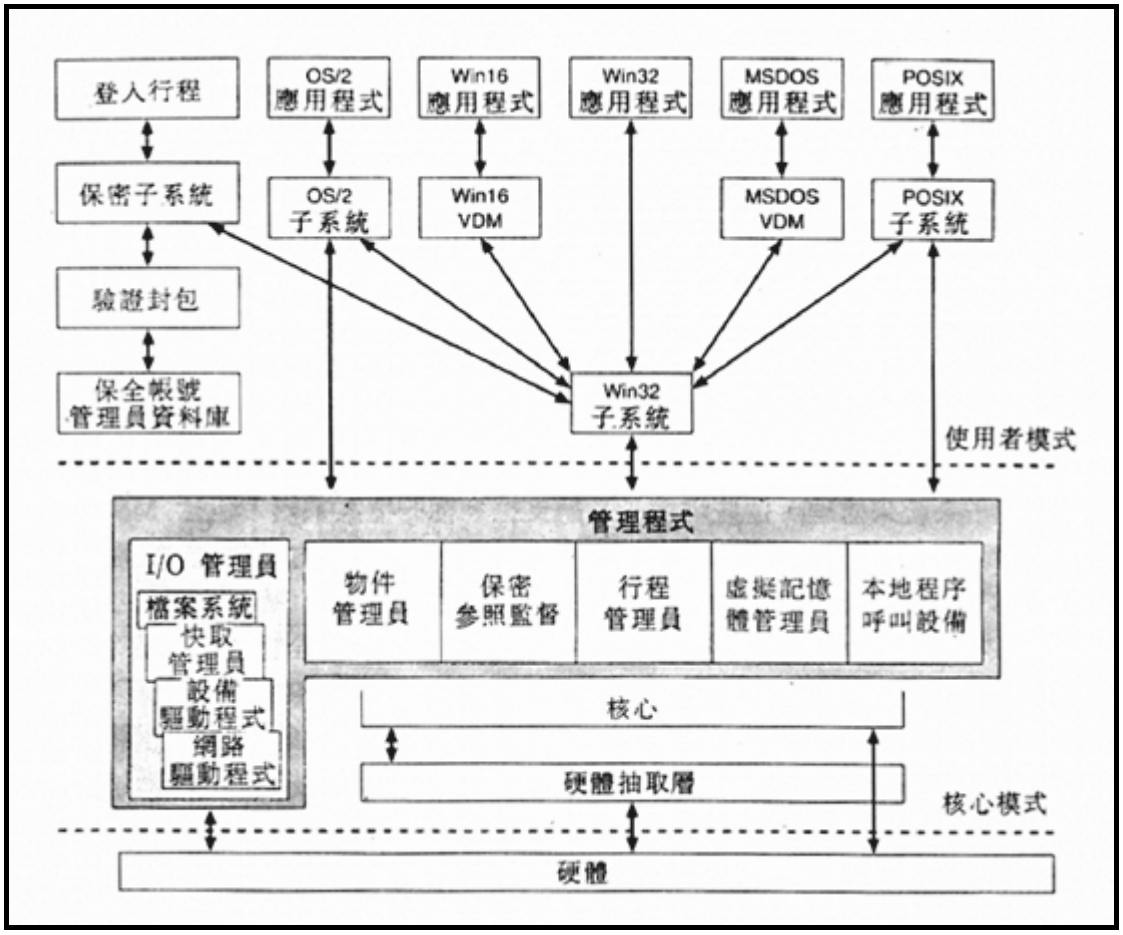
可靠性 (Reliability)，NT 利用硬體的方式保護虛擬記憶體來防堵缺陷和破壞，利用軟體保護機制來保護作業系統的資源。NT 所提供的 NTFS 檔案系統也具有自動復原的功能。

相容性 (Compatibility)，依據 IEEE 1003.1 (Portable Operating System Interface for Computer Environments，POSIX)的標準，NT 提供了應用程式原始階層的相容性。先前有提到，在 NT 的階層式架構中的使用者模式包含了許多伺服器子系統，其中環境子系統 (Environmental Subsystem)模擬了不同的作業系統。所以，包括 DOS、16 或 32 位元的 Windows、OS/2、Lan Manger 編譯後的可執行檔 (二進位)，皆可以在 NT 上執行，但為了可靠性與安全性，NT 不允許這類的應用程式直接存取硬體的輸入輸出埠。

效能 (Performance)，NT 提供了對稱式的多工處理，最多可支援到 8 個處理器，而在多處理機的電腦上可以有多個執行緒 (Thread)同時執行。除了核心之外，優先權高的執行緒可以先取得子系統中的執行緒，使得系統可以快速的反應外界的事件。子系統間也可藉著本地行程的呼叫，有效且快速的傳遞訊息。

國際性（International），NT 系統中使用了 Win32 API NLS（Natural Language Support, 自然語言支援)功能處理對地區設定有影響的資料，根據不同國家的習慣提供特殊的副程式來規劃日期、時間以及貨幣。NT 採用了 Unicode 作為其字元集，雖然也支援 ANSI（或稱 ASCII)字元，但是 NT 會先將其轉成 Unicode 在進行處理。

2.2 系統組成



[作業系統概念的應用]

NT 階層式的架構與模組化的子系統結構可以上圖清楚的表示。核心模式扮演著應用軟體與硬體之間的中介角色，核心、硬體抽象層、管理程式皆在核心模式中執行。使用者模式中可劃分為兩大子系統，環境子系統與保密子系統，環境子系統主要是在模擬其他不同種類的作業系統，讓其他作業系統的應用程式也能在 NT 系統下執行，保密子系統則是提供系統安全驗證的功能

2.3 硬體抽象層 (Hardware Abstraction Layer)

硬體抽象層 (Hardware Abstraction Layer)其實是一個軟體的階層，它輸出虛擬機器介面提供給核心、管理單元以及設備的驅動程式使用，主要是為了隱藏硬體間的差異，使得 NT 系統具有可攜性。硬體只需要一個版本的驅動程式而不需要每種作業系統一個版本，考量到效能的問題，I/O 的驅動程式，可以直接存取硬體。

2.4 核心 (Kernel)

核心是整個作業系統階層式架構的精髓，它負責為應用軟體配置記憶體，與硬體溝通、協調資源，並分配處理器時間。微軟為了符合系統管理員的需求，在 Windows 2000 的核心上作了幾項改變：

- **程序稽核與 CPU 節流**

  程序稽核能夠能利用來紀錄個別程序所消耗的處理器週期，可計算出個別程序的 CPU 使用率。CPU 節流則可以防止已經終止的程序，繼續的佔用大量處理器的時間，使得其他程序無法正常運作。

- Spin Count

  Spin Count 可以增進在多重處理器系統下，有多個程序必須同時存取相同資源時的效能。它負責控制一個程序在等待某項資源前，主動先嘗試存取的次數，若所有嘗試都失敗後，才會開始等待。

- Scatter/Gather I/O

  利用將資料從不連續的系統記憶體，搬移到磁碟連續的空間，以增進應用程式伺服器的效能。對於管理者而言，這項工作是完全透明的，並不需要作額外的設定。

- Quantum

  Quantum 是執行緒 (Thread)的一種屬性，代表執行緒可以使用處理器的時間。由於 CPU 控制權再執行緒間切換是需要時間的，所以如果應用程式的優先權較高，就是配置了較短的 Quantum，提供較平順的多工作業。如果背景服務的優先權較高，則是配置較長的 Quantum，可以增進背景服務 (網路服務)的效能。

- Windows Driver Model，WDM

驅動程式扮演著作業系統與硬體之間的溝通角色，在早期，相同的一件硬體配備，在不同的作業系統下需要不同版本的驅動程式。新的 WDM (Windows Driver Model)技術可以讓 Windows 2000 與 Windows 98 兩種不同的作業系統，使用同一份驅動程式，減低了廠商與管理者的負擔。

新的 WDM Kernel Streaming 架構增進了即時串流媒體 (real-time streaming media)的效能表現，整合至核心中，不再需要像以前一樣使用額外的軟體來處理串流媒體。

- **企業記憶體架構**

  針對企業中處理大型資料庫的伺服器，Windows 2000 採用 EMA (Enterprise Memory Architecture，企業記憶體架構)，讓擁有 64 位元處理器的伺服器能夠定址至 32GB 的記憶體。

- **I_2O 支援**

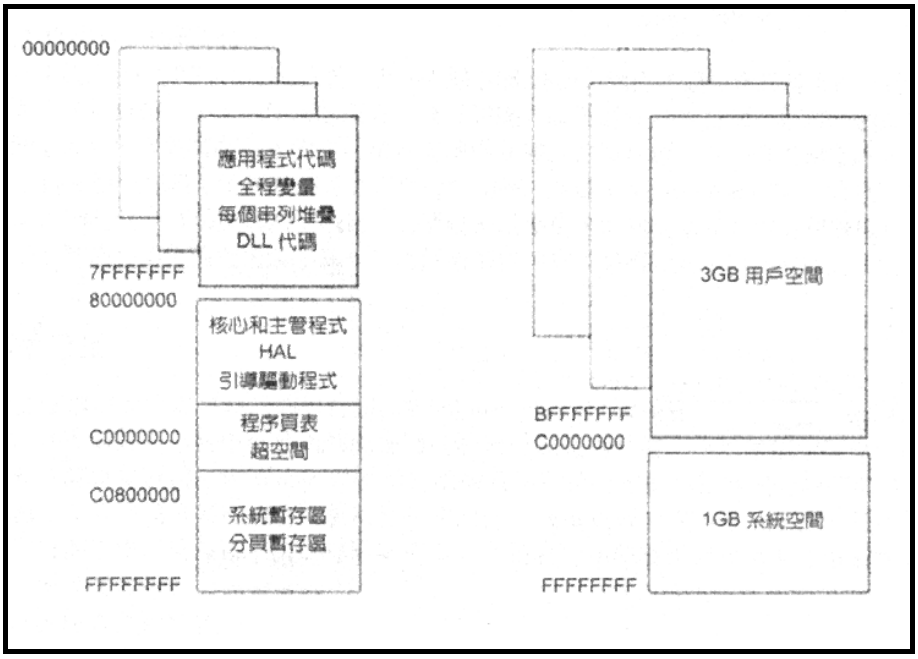
  I_2O (Intelligent I/O Architecture)可以降低系統處理器的負擔，同時也能增加系統輸出入的效能。

2.5 記憶體管理

Windows 2000 管理程式中有一個記憶體管理器，它是在 Ntoskrnl.exe 中。如同其他 Windows 2000 的執行程式一樣，記憶體管理器是完全可重入的，它支援多程序平行執行。為了實行可重入，記憶體管理器使用了幾個不同的內部同步機制來控制他自身資料結構的呼叫，如旋轉鎖 (Spin-lock)和執行程式資源。

2.4.1 位址空間的分配

Windows 2000 作業系統的記憶體架構是屬於分頁需求 (demand-paged)與虛擬的記憶體系統，由於是 Windows 2000 是 32 位元的作業系統，因此可以定址到 2^{32} 的線性位址空間，也就是說，每個處理程序都可以存取到 4GB 的記憶體。這 4GB 的空間中，處理程序可以佔有 2GB 的專用位址空間，作業系統則占用剩下的 2GB 位址空間。



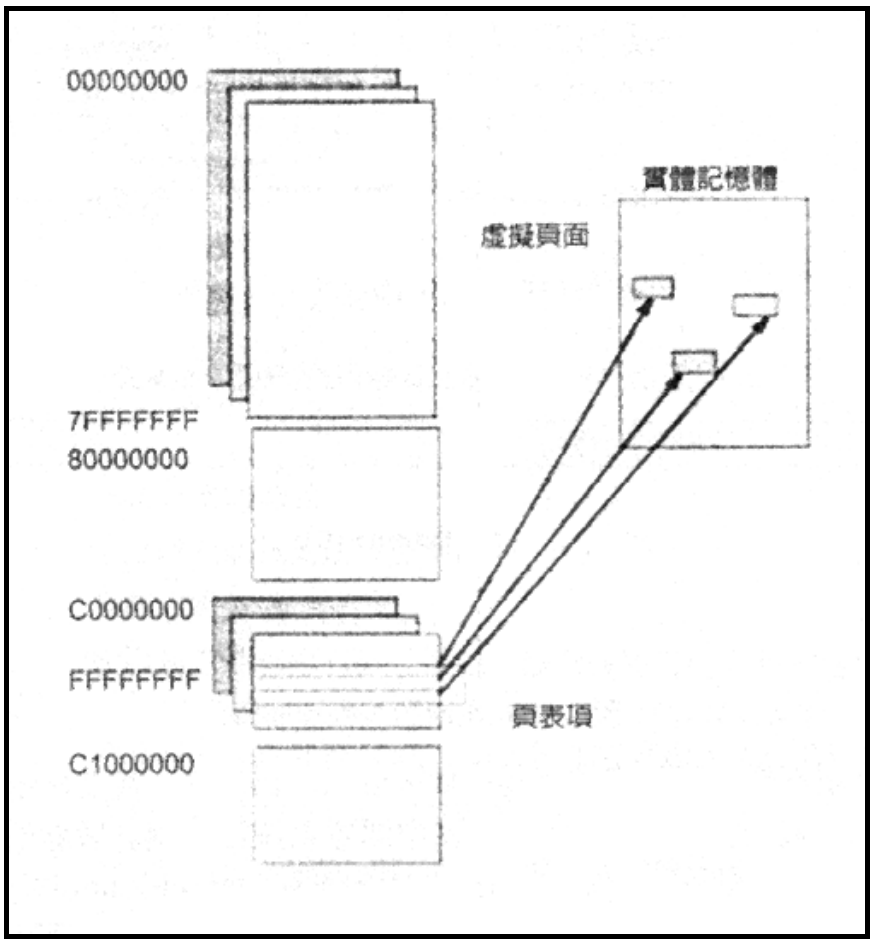
[作業系統原理-Windows 核心剖析]

Windows 2000 advanced server 與 Windows 2000 Datacenter server 允許使用者擁有 3GB 的位址空間(處理程序佔有 2GB 的專用位址空間，剩下的 1GB 位址空間則作為系統空間)，這個特性是為滿足一些應用程式的需求而採用的臨時解決方法，例如，資料庫伺服器可能較需要在記憶體中保存比 2GB 位址空間更多的資料。

對於需要呼叫整個 3GB 位址空間的程序來說，程序映射檔案必須在映射頭，透過指定鏈結標幟 /LARGEADDRESSAWARE 來設置 IMAGE FILE LARGE- ADDRESS AWARE 標幟 (flag)，否則應用程式將不會看到大於 0x7FFFFFFF 的虛擬位址空間。

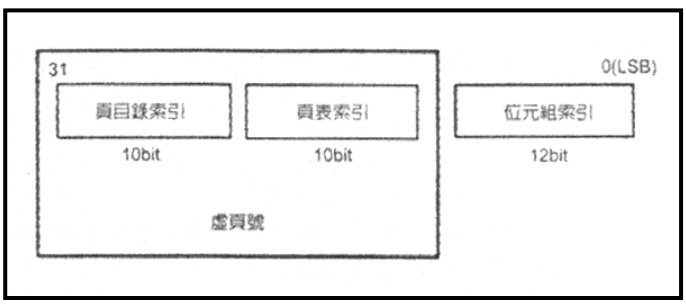
2.4.2 位址空間的轉換

由於用戶應用程式是以 32 位元虛擬位址方式編址，所以 CPU 必須先利用記憶體管理器建立和維護的資料結構將虛擬位址轉換為實體位址。

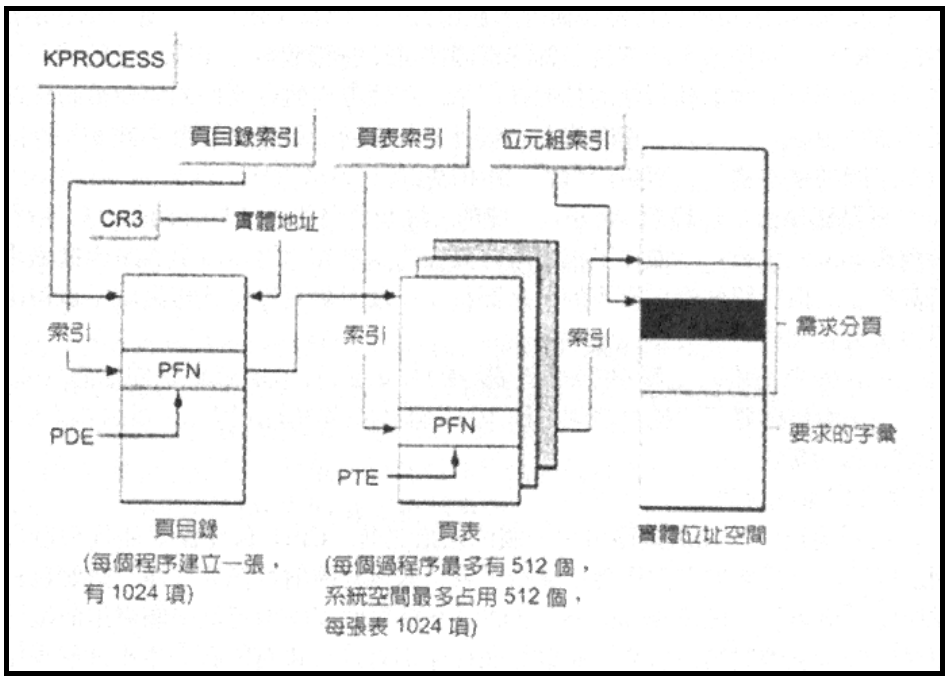


[作業系統原理-Windows 核心剖析]

虛擬位址轉換



Windows 2000 在 x86 的結構上利用兩段式頁表結構來實現虛擬位址向實體位址的轉換。32 位元的虛擬位址被分為三個獨立的分量：頁目錄索引、頁表索引、位元組索引，頁面的大小及頁表項(Page Table Entry，PTE)決定了頁目錄和頁表索引的寬度。假設，一個頁面包含了 4096 個位元組，於是位元組索引便被確定為 12 位元寬(2^{12} = 4096)。



[作業系統原理-Windows 核心剖析]

「頁目錄索引」用來指出虛擬位址的頁目錄在頁表中的位置。
「頁表索引」則用來確定頁表象在頁表中的位置。
「頁表項」包含了虛擬位址被映射到的實體位址。
「位元組索引」能在實體頁面中找到某個特定的位址。

轉換步驟

1. 依照頁目錄索引在頁目錄中找到頁目錄項 (Page Directory entry，PDE) 的位置，而頁目錄項包含了頁框編號 (Page Frame Number，PFN)。
2. 先利用頁框編號找到所需的頁表，再依照頁表索引於頁表中指出頁表象的位置。
3. 頁表項用於確定頁框的位置，若所需的頁面是有效的，頁表項就會包含實體記憶體中的一個頁框編號，相對應的虛擬頁面就包含在此實體頁框中；若頁表項中表明所需的頁面是無效的，記憶體管理器會嘗試的使該頁面有效。

2.5.3 記憶體保護機制

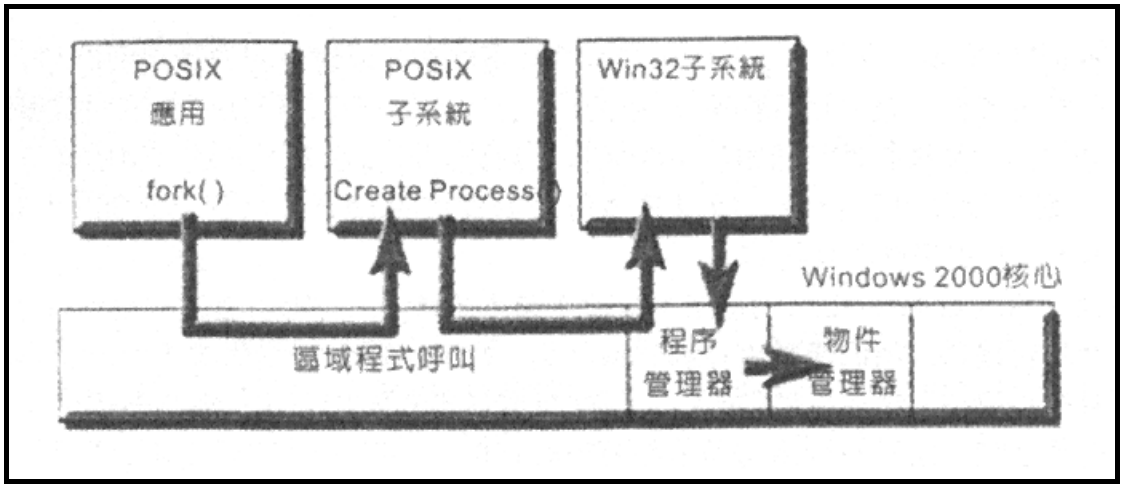
Windows 2000 提供了四種記憶體的保護機制，防止用戶程序有意或無意的破壞其他程序或系統的位址空間：

1. 所有系統範圍核心元件使用的資料結構和記憶體緩衝區只能在核心模式下呼叫，使用者模式的執行緒不能呼叫這些頁面。如果使用者模式的執行緒試圖嘗試呼叫，硬體會產生一個錯誤訊息，接著記憶體管理器會向執行緒報告一個呼叫衝突。
2. 每個程序都有一個獨立、專用的位址空間，禁止其他程序的執行緒呼叫。唯一例外的是，若該程序與其他程序共用頁面，或另一程序具有對該程序物件的虛擬記憶體的讀寫許可權時，可以透過函數的呼叫來使用。當執行緒呼叫一個位址時，透過控制虛擬位址的轉換，Windows 2000 可以保證在程序中執行的執行緒不會錯誤地呼叫始於另外一個程序的頁面。
3. 所有 Windows 2000 支援的處理器提供了一些硬體記憶體的保護措施（如：讀/寫、唯讀等）。在程序的位址空間中，內碼表被標示為唯讀可以防止被用戶執行緒修改。
4. 共享記憶體區域物件具有標準的 Windows 2000 存取控制表（ACL），當程序試圖存取共享記憶體時會檢查存取控制表，這樣對共享記憶體的呼叫也就被限制在具有適當存取權的程序中。

2.6 行程管理

Windows 2000 中的行程是系統資源分配的基本單位，行程被視為物件來管理，可透過相應控制碼 (handle)來呼叫行程物件。行程物件的屬性包括：行程標識 (PID)、資源呼叫權杖 (Access Token)、行程的基本優先順序 (Base Priority)和處理器相關性 (Processor Affinity)等。

為了支援 Win32、OS/2、POSIX 等多種執行環境子系統，Windows 2000 核心的行程之間並沒有任何的關係，各執行環境子系統分別建立、維護和表達各自的行程關係。Windows 2000 把 Win32 環境子系統設計成整個系統的主子系統，一些基本的行程管理功能被放置在 Win32 子系統中，POSIX 和 OS/2 等其他子系統會利用 Win32 子系統的功能來管理自身的功能。在 Windows 2000 中，與一個執行環境子系統中的應用行程相關的行程控制區塊資訊會分布在本執行環境子系統、Win32 子系統和系統核心中。

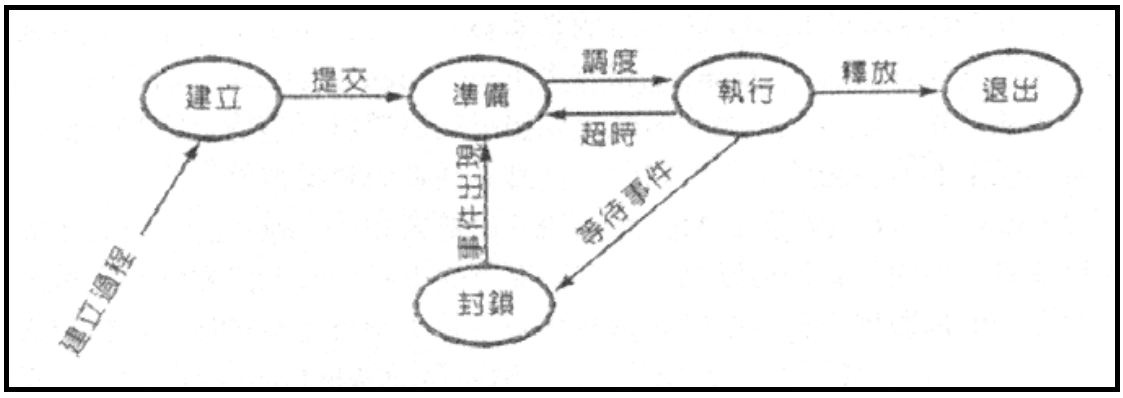


[作業系統原理-Windows 核心剖析]

2.6.1 行程的狀態轉換

行程在從建立到中止的程序中，一直都是處於不斷變化的狀態。所有的作業系統，都脫離不了下列五種狀態：執行狀態 (Running)、準備狀態 (Ready)、封鎖狀態 (Blocked)、建立狀態 (New)、退出狀態 (Exit)。

在五種狀態模型中，行程在執行程序中主要是在準備、執行和封鎖三種狀態間轉換，建立狀態和退出狀態描述行程建立和退出的過程。



[作業系統原理-Windows 核心剖析]

1. 執行狀態 (Running)：行程佔用處理器資源，處於此狀態的行程的數目小於或等於處理器的數目。在沒有其他行程可以執行時，通常會自動執行系統的閒置行程。
2. 準備狀態 (Ready)：行程已經獲得處理器外的所需資源，等待分配處理器資源，只要分配處理器行程就可以執行。準備行程可按多個優先順序來劃分佇列。例如，當行程由於 I/O 操作完成而進入準備狀態時，排入高優先權的佇列；行程用完時間配額 (Time Quantum)而進入準備狀態時，排入低優先權

的佇列。

3. 封鎖狀態 (Blocked):當行程因為要等待 I/O 操作或行程同步等條件而暫停執行時，就處於封鎖狀態。
4. 建立狀態 (New):行程在建立的狀態時，是不能執行的。作業系統在建立狀態要進行的工作包括分配和建立行程控制區塊表項、建立資源表並分配資源、載入程式並建立位址空間表等。
5. 退出狀態 (Exit):行程已經結束執行，回收行程控制區塊之外的其他資源，並讓其他行程從行程控制區塊中收集相關資訊。

2.6.2 執行緒排程

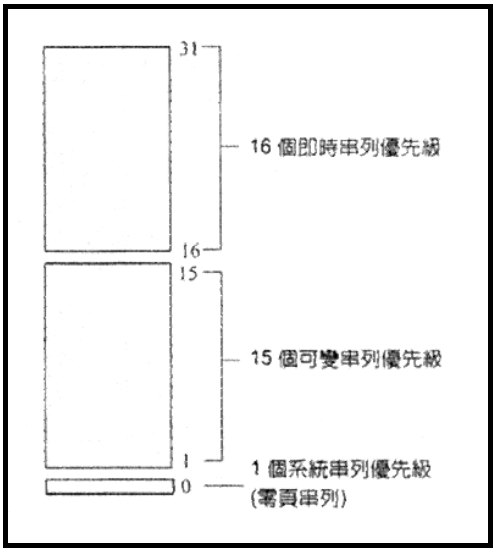
Windows 2000 作業系統中，處理器調度的調度物件是執行緒 (thread)，也稱為執行緒排程。Windows 2000 的執行緒排成並不是採用單一的排程演算法，而是採用多種排程演算法的整合，根據系統實際的需要來加強。

Windows 2000 是基於優先順序的搶先式多處理器排程系統。排程系統總是先執行優先順序較高的準備執行緒，當一個執行緒被排程進入執行狀態時，它可以執行一段時間配額，時間配額指的就是 Windows 2000 系統允許一個執行緒執行一次的最大時間長度。一旦時間配額用完，執行便會被中斷，接著排程系統會判斷是否需要降低該執行緒的優先順序，並找出是否有其他高優先順序或相同優先順序的執行緒在等待執行。時間配額是可以修改的，由於 Windows 2000 的搶先式排程，一個執行緒的一次排程可能沒有用完它的時間配額，如果一個高優先順序的執行緒進入等待狀態，執行緒在還沒用完時間配額的情況下，被高優先順序的執行緒搶先執行。

2.6.3 執行緒優先順序

Windows 2000 內部使用 32 個執行緒優先順序，範圍從 0 到 31，並分成三個部份：

1. 16 個即時執行緒優先順序
2. 15 個可變執行緒優先順序
3. 1 個系統執行緒優先順序 (僅用於對系統中閒置實體頁面進行零頁執行緒)



[作業系統原理-Windows 核心剖析]

執行緒優先順序的指定可以從兩個方向來進行:用戶可以透過 Win32 應用程式排程介面來指定執行緒的優先順序，系統核心也可以控制執行緒的優先順序。Win32 應用程式排程介面可以在行程建立時指定其優先順序類型為即時、高階、中上、中階、中下、和閒置，並進一步的在行程內各執行緒建立時指定執行緒的相對優先順序為相對即時、相對高階、相對中上、相對中階、相對中下、相對低階和相對閒置。

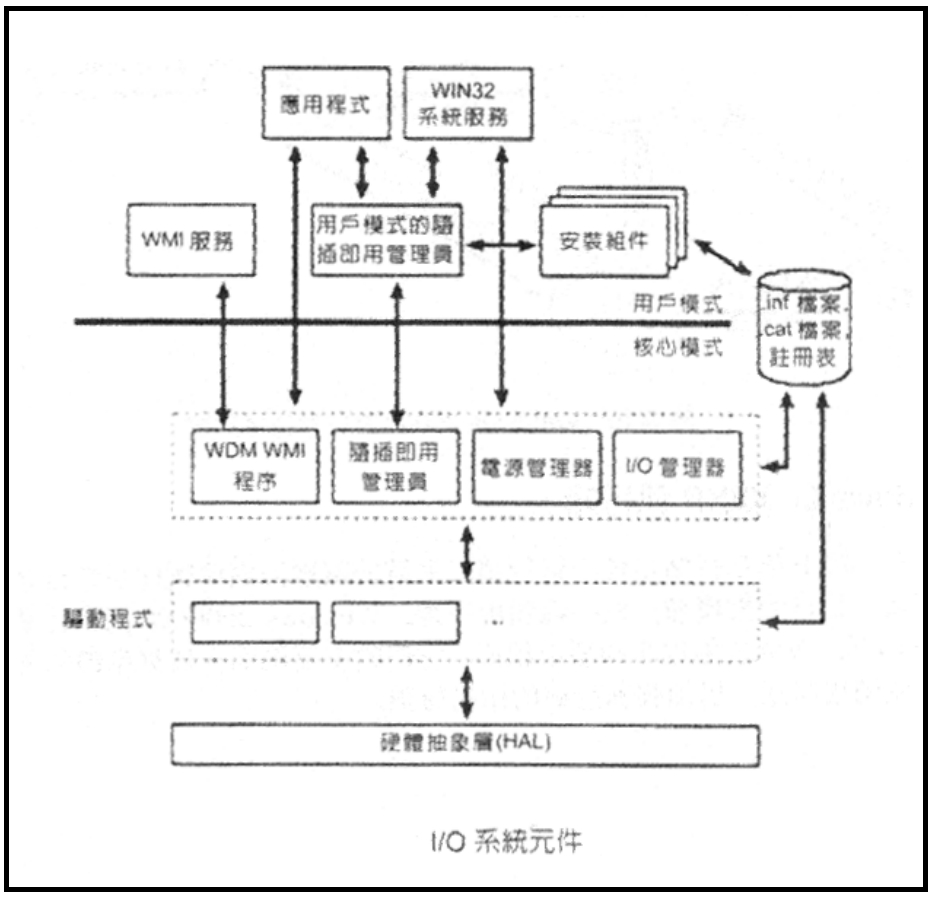
2.7 環境子系統

Windows 2000 支援三種環境子系統：Win32、POSIX 和 OS/2，在這三個子系統中，Win32 子系統比較特殊，因為 Windows 2000 需要它才能執行。其他的兩個子系統只有在需要的時候才會被啟動，而 Win32 子系統則必須始終處於啟動的狀態。

環境子系統的作用主要是將基本的管理程式系統服務的某些子集提供給應用程式，每個子集都可以提供呼叫 Windows 2000 中本地服務的不同子集，函數的呼叫不能在子系統中混用。用戶應用城市不能直接呼叫 Windows 2000 系統服務，這種呼叫必須透過一個或多個子系統的動態鏈結程式庫 (Dynamic Link Library，DLL)作為媒介才可完成。

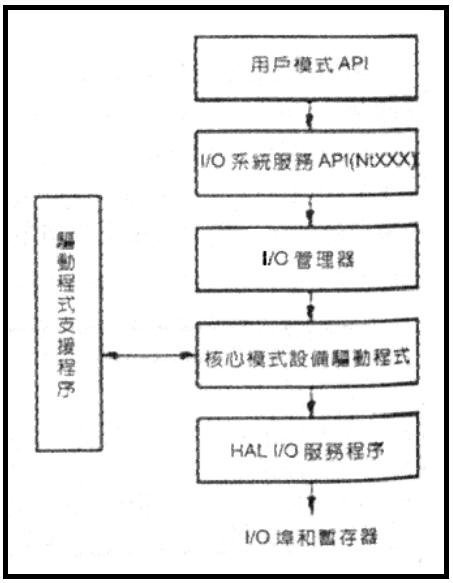
2.8 I/O 系統

Windows 2000 的 I/O 系統是管理程式的一個元件，它接受來自用戶程式或核心程式的 I/O 請求（request），並且以不同的形式將它們傳送到 I/O 設備。在用戶程式 I/O 函數和實際的 I/O 硬體之間有幾個分別獨立的系統元件，包括檔案系統驅動程式、篩檢程式驅動程式和低階設備驅動程式。



[作業系統原理-Windows 核心剖析]

大部份的 I/O 操作不會涉及到所有的 I/O 元件，典型的 I/O 操作通常只會涉及到 I/O 管理器、一個或多個設備驅動程式以及硬體抽象層（HAL）。下圖為一個典型的 I/O 請求流程。



[作業系統原理-Windows 核心剖析]

在 Windows 2000 中，所有的 I/O 操作都是透過虛擬檔案來執行，虛擬檔案的技術隱藏了 I/O 操作的細節，為應用程式提供了一個統一的介面。虛擬檔案用於 I/O 的所有來源或目標，它們都被當作檔案來處理，所有讀取寫入的資料都可以被視為直接讀寫到這些虛擬檔案的流。

第3章 檔案系統

3.1 簡介

Windows 2000 本身支援五種檔案系統，分別是FAT16與FAT32（ File Allocation Table ）、NTFS（ New Technology File System ）、CDFS（ Compact Disc File System ）、UDF（ Universal Disk Format ）。除了CDFS是標準的光碟格式和UDF是DVD光碟格式外，其餘三種皆屬於一般硬碟適用的格式。

3.2 FAT 16檔案系統

FAT (File Allocation Table，檔案配置表)是為了在硬碟中儲存檔案與檔案目錄的方式，FAT 16檔案系統最早是被開發來支援微軟的 DOS系統。顧名思義，FAT16的16代表 2^{16} ，相當於可以定址到65,535個叢集。叢集（clusters）為其儲存單位，在格式化時，會依照分割區的大小，決定叢集的大小。最小是2,048個位元組（Bytes)，相當於2KB，最大則是32,768個位元組，相當於32KB。

由於每個檔案都至少會用到一個叢集，FAT 16有65,535個檔案的限制，且極易產生所謂的內部碎裂（Internal Fragmentation)。譬如說，我們有一個大小為35KB的檔案，在叢集大小為32KB的系統中，這個檔案將會佔用2個叢集，一個完整的32K叢集，另一個則是以3KB的大小佔用了32KB的叢集，如此一來，第二個叢集會剩餘29KB的空間且無法被其他檔案使用造成浪費，此即所謂的內部碎裂。

受限於本身架構的關係（可定址的65,535個叢集乘以每個叢集最大32,768位元組等於2,147,450,880位元組，相當於2GB），FAT 16檔案系統只能格式化最大2GB的磁碟分割區。

3.3 FAT 32檔案系統

FAT 32檔案系統的原理與前一節介紹的FAT 16檔案系統大同小異，它是一個增強的檔案系統，可提高磁碟性能並增加可用的磁碟空間。FAT 32不支援512MB以下的磁碟分割區，但是它可以支援到2TB（2,000GB），由於使用較小的叢集，因此能提高15%的磁碟空間利用率。

下表為FAT 16與FAT 32在不同大小的磁碟分割區中叢集的大小比較。

分割區大小	FAT16 叢集大小	FAT32 叢集大小
32 MB	2 KB	不支援
128 MB	2 KB	不支援
256 MB	4 KB	不支援
512 MB	8 KB	4 KB
1 GB	16 KB	4 KB
2 GB	32 KB	4 KB
3 GB ~ 7 GB	不支援	4 KB
8 GB ~ 16 GB	不支援	8 KB
16 GB ~ 32 GB	不支援	16 KB
32 GB以上	不支援	32 KB

3.4 NTFS檔案系統

NTFS是一個64位元的檔案系統，理論上，檔案與磁碟的分割大小可以達到16 exabytes（1 exabyte = 2^{64} bytes），由於需要額外的負載，最小的磁碟分割建議為50MB。NTFS支援檔案壓縮，你可以逐檔案、目錄，甚至整個子目錄來進行壓縮，可以壓縮到50%，有相當不錯的效果。

至於碎列現象，NTFS下的檔案通常不會有碎裂的現象，當檔案資料決定寫入磁碟中時，系統會為你找一塊連續的儲存空間。只有在磁碟空間將用盡或檔案修改而增大時才會發生碎裂的現象。

NTFS本身還支援Hot Fixing的功能，當發現某個磁區 (Sector)壞掉時，系統會將該磁區標記起來並跳過這壞掉的磁區。

NTFS與FAT相比，是更為安全的新型檔案系統。使用 NTFS 可讓您針對資料設定更明確的安全性使用權限，也可使用Windows 2000 的新功能「加密檔案系統」(Encrypting File System，EFS) 對特定資料夾和檔案進行加密，以增進安全性並保護機密資料，關於NTFS的存取權限我們將在後面的章節介紹。

3.5 NTFS 的檔案壓縮

先前我們談到 NTFS 支援壓縮的功能，NTFS 檔案系統可以自動的壓縮和解壓縮檔案與資料夾，可以對個別的檔案、資料夾甚至是整個磁碟作壓縮的動作。考慮到效能的因素，NTFS 並不支援叢集大小超過 4KB 的分割區。雖然壓縮功能對於磁碟空間的使用上是一大助益，但對於某些需要大量寫入寫出的機器來說，壓縮功能只會造成系統負荷過重、效率低落的情況。

3.5.1 壓縮屬性

NTFS 檔案系統中的每一個檔案、資料夾甚至是分割區都具有壓縮屬性，只要具有壓縮屬性，即代表了該檔案、資料夾或磁碟機已經被壓縮了。

壓縮過後的檔案以使用者的角度來看與原來的檔案無異，只差在容量的大小不同，當需要讀取或寫入壓縮過的檔案時，系統會先將該檔案解壓縮然後執行讀取或寫入的動作，完畢後再將其壓縮起來，對系統來說只是多了一道程序而已，而對使用者來說只可能會感覺到執行速度稍微變慢。

3.5.2 壓縮方式

至於要怎麼壓縮，NTFS 提供了兩種方式：

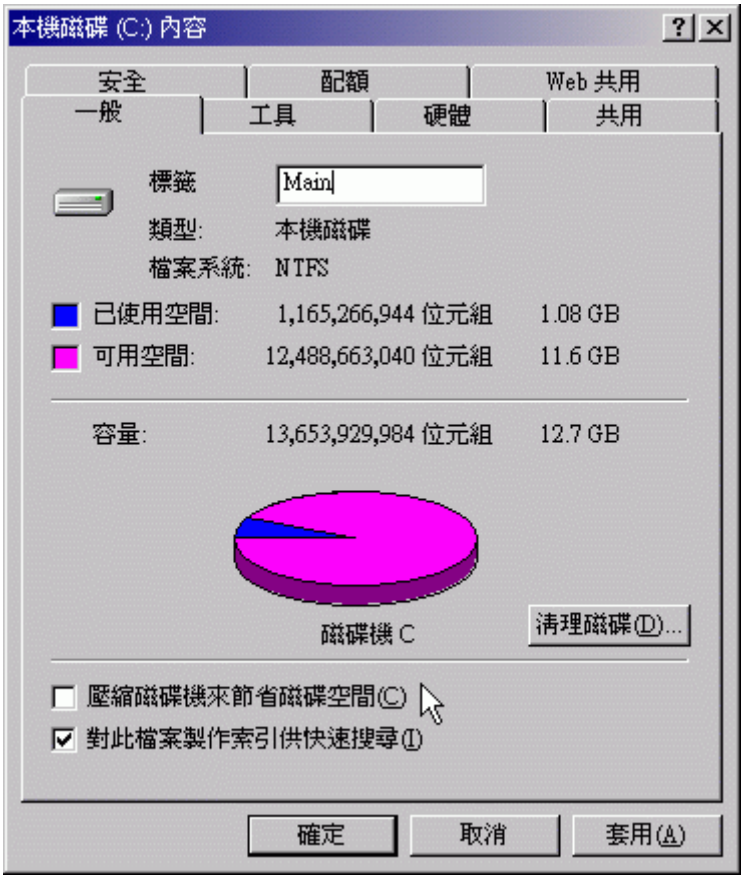
- Windows 2000 Explorer
- Compact 指令

■ Windows 2000 Explorer：

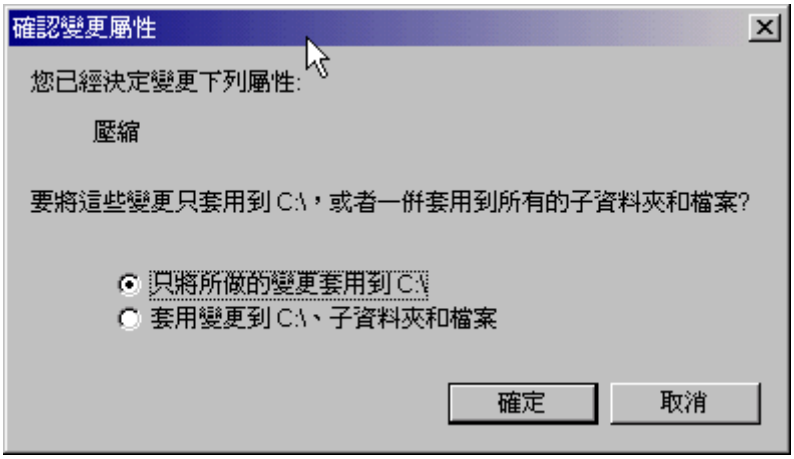
1. 磁碟壓縮

步驟一：選擇「我的電腦」→「磁碟機」→「內容」

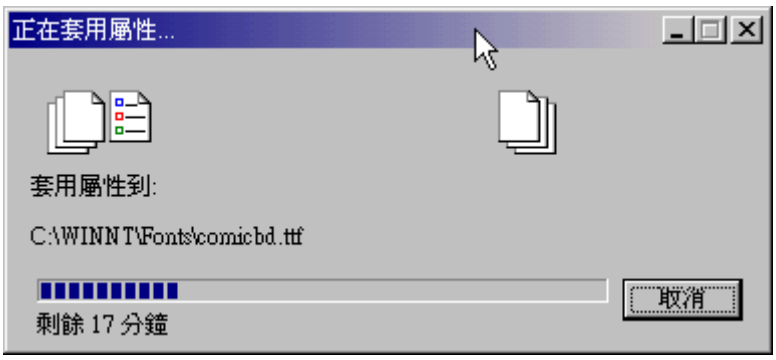
步驟二：在「一般」分頁的下方可以看到「壓縮磁碟機來節省磁碟空間」的核取方塊（Check Box）。



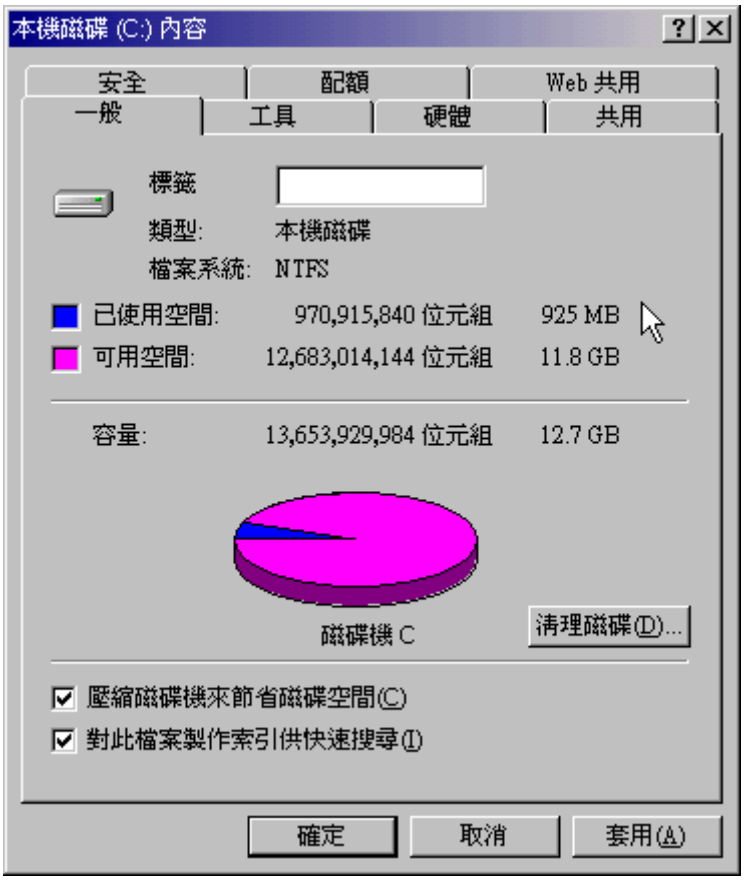
步驟三：勾選「壓縮磁碟機來節省磁碟空間」的核取方塊後，系統會出現「確認變更屬性」的對話窗，在這裡可以選擇只壓縮磁碟機根目錄下的所有檔案，或者是將壓縮的屬性套用到根目錄下的所有檔案及資料夾。



步驟四：選擇「確定」後，系統會根據你的選擇來套用壓縮的屬性。



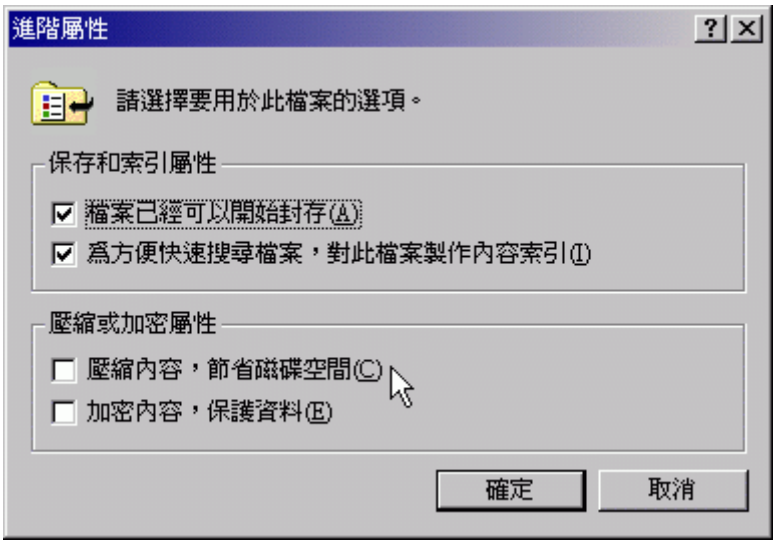
步驟五：在壓縮過後我們可以發現，磁碟機使用的空間已經由原來的 1.08 GB 壓縮成 925 MB。



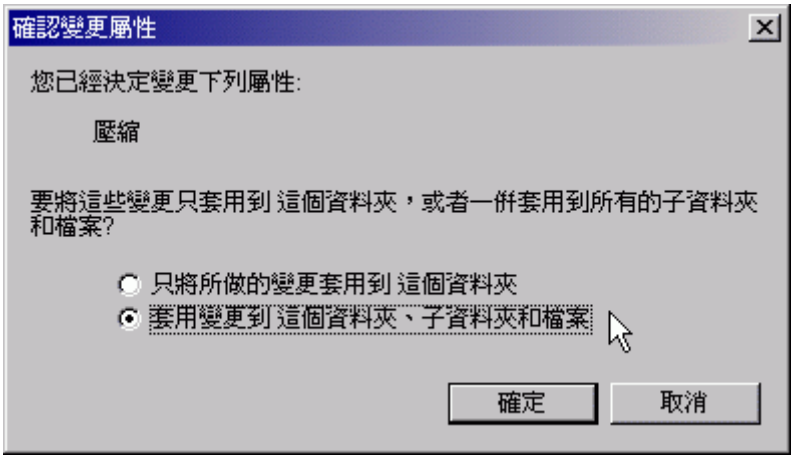
2. 資料夾壓縮

步驟一：資料夾的壓縮與磁碟壓縮步驟相當類似，選擇「我的電腦」→「磁碟機」→「內容」，在「一般」分頁的右下方點選「進階」按鈕。

步驟二：在「進階屬性」對話窗中，勾選「壓縮內容，節省磁碟空間」核取方塊。



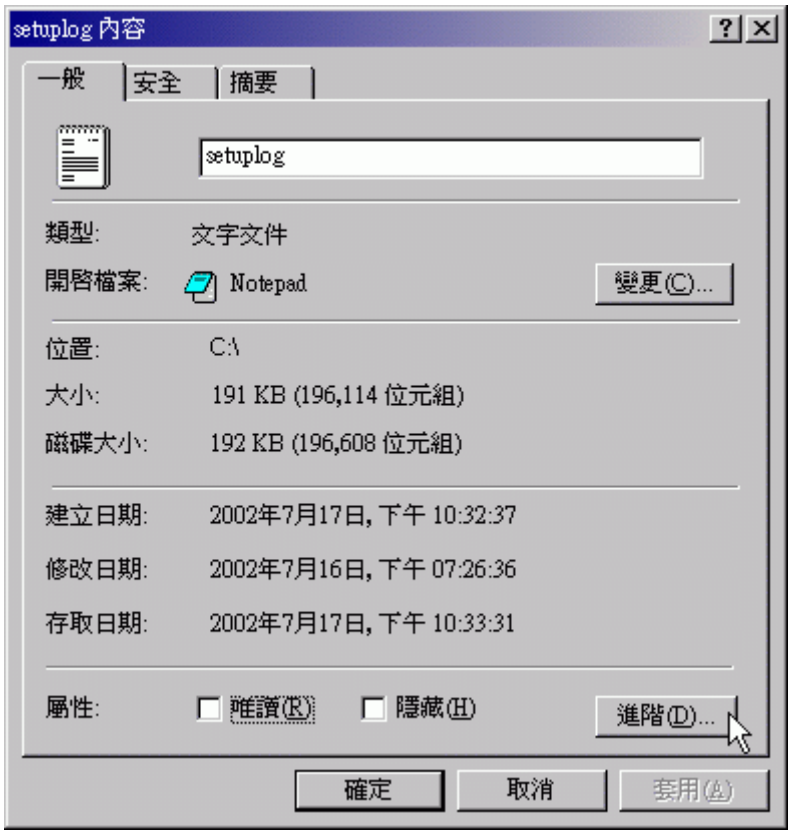
步驟三：「確認變更屬性」對話窗與壓縮磁碟相同



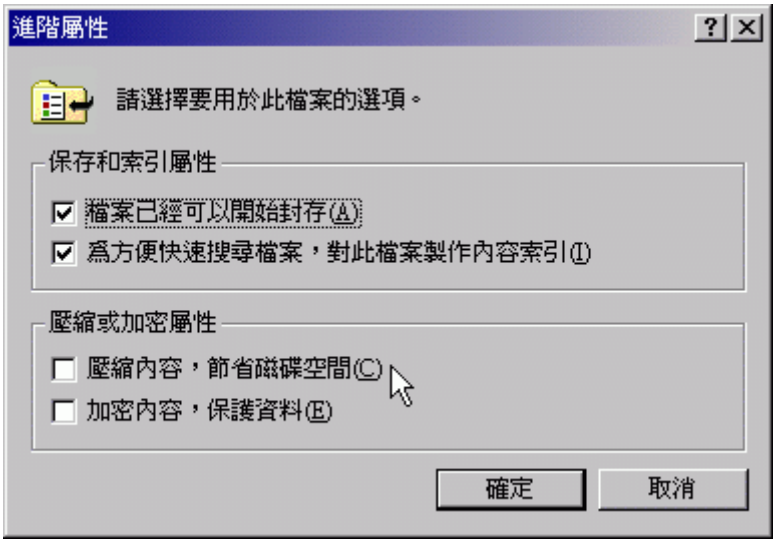
步驟四：套用完壓縮屬性後，我們可以發現資料夾的大小並未改變，這是為了讓使用者知道原始資料夾的大小，而資料夾所佔的磁碟大小則由原先的 19 MB 壓縮成 16.9 MB。

3. 檔案壓縮

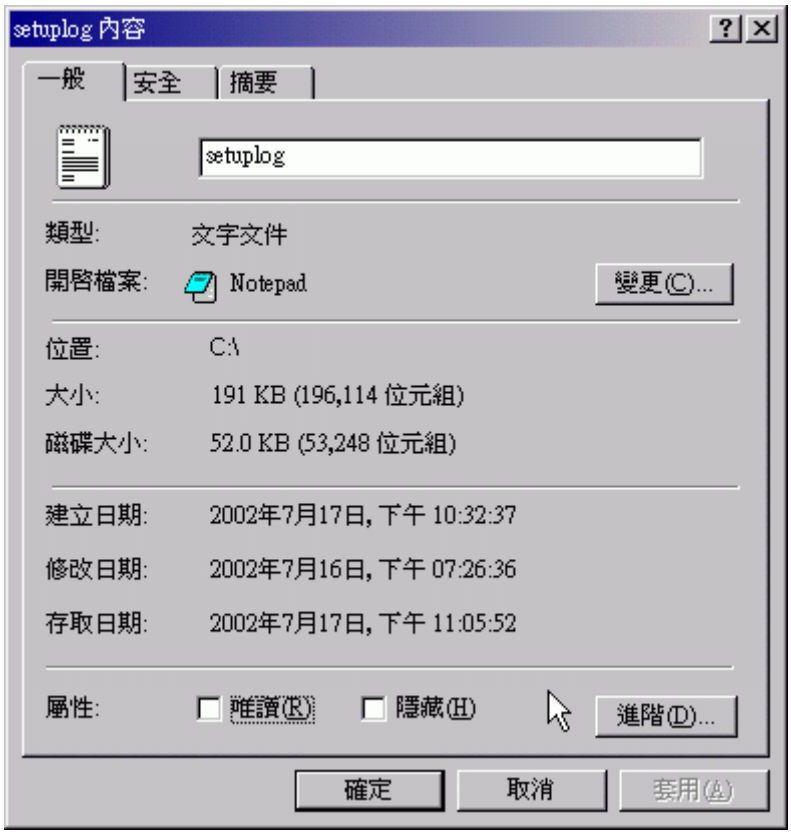
步驟一：選擇想要壓縮的檔案，在檔案「內容」的「一般」分頁右下方點選「進階」按鈕。



步驟二：在「進階屬性」對話窗中，勾選「壓縮內容，節省磁碟空間」核取方塊。



步驟三：套用完壓縮屬性後，我們可以發現一樣可以發現檔案的大小並未改變，而所佔的磁碟大小則由原先的 191 KB 壓縮成 52 KB。



步驟四：如果想要取消壓縮，只要以相同的步驟，取消壓縮核取方塊的勾選即可。

■ Compact 指令：

Compact 主要功能為顯示或改變 NTFS 磁碟分割上的檔案壓縮。

Compact 的參數如下：

COMPACT [/C | /U] [/S[:dir]] [/A] [/I] [/F] [/Q] [filename [...]]

參數	作用
/C	壓縮指定的檔案。 將標示目錄使以後新增的檔案將被壓縮。
/U	解壓縮指定的檔案。 將標示目錄使以後新增的檔案將不被解壓縮。
/S	對所給目錄及其所有子目錄下的檔案執行指定的作業。 "dir"預設為目前的目錄。
/A	顯示有隱藏或系統屬性的檔案。 這些檔案在預設中是不會顯示的。
/I	即使發生錯誤，仍繼續執行指定的操作。依預設，遇到錯誤時，COMPACT 將停止。
/F	強迫壓縮所有指定的檔案，即使是已壓縮過的檔案。依預設，將略過已壓縮的檔案。
/Q	只報告最基本的資訊。
filename	指定一格式、檔案、或目錄。

若不帶參數，則是顯示目前目錄及其含有所有檔案的壓縮狀態。您可以使用多個檔名或通配字元，在多個參數彼此間必須有空格。

首先選擇「開始」→「程式集」→「附屬應用程式」→「命令提示字元」(或由「開始」→「執行」，直接執行 cmd.exe)，開啟 DOS 指令視窗。

由於磁碟壓縮、資料夾壓縮和檔案壓縮的方式一樣，只差在參數的不同，所以對於磁碟壓縮及資料壓縮將只列出執行時需要下的指令和參數。

1. 磁碟壓縮

壓縮　　C:\>COMPACT /C /S
解壓縮　C:\>COMPACT /U /S

2. 資料夾壓縮

壓縮　　C:\>COMPACT /C /S:指定的資料夾名稱
解壓縮　C:\>COMPACT /U /S:指定的資料夾名稱

3. 檔案壓縮

在壓縮之前，先執行 COMPACT /A，列出目前磁碟目錄包含隱藏檔案與系統檔案的壓縮狀態。我們可以看出目前目錄共有 22 個檔案及資料夾且都尚未被壓縮。

執行 COMPACT /C SETUPLOG.TXT 壓縮根目錄下的 SETUPLOG.TXT 檔案，檔案由原先的 196,114 Bytes 壓縮到 53,248 Bytes，壓縮比率為 3.7 比 1。

再檢視磁碟目錄的狀態，這次我們不顯示隱藏檔案與系統檔案。磁碟目錄中由於 SETUPLOG.TXT 已經被壓縮，所以剩下 21 個未壓縮的檔案和資料夾。

接下來是解壓縮的方式，執行指令與壓縮相同，但要將參數由原來的壓縮(/C)改為解壓縮（/U)。

第4章 磁碟管理

4.1 簡介

一個磁碟機，必須先經過磁碟分割、格式化，我們才能將資料儲存在磁碟上。Windows 2000 提供了相當方便的磁碟分割工具，不僅可以直接在安裝時分割及格式化磁碟機，也可以在安裝完後，利用一個圖形化的工具 Disk Management，管理本機和遠端磁碟。

Windows 2000 提供兩種磁碟區的儲存類型：一種是與舊作業系統相容的基本儲存，另一種是能更有效利用磁碟空間的動態儲存，使用基本儲存的磁碟稱為基本磁碟 (Basic Disk)，而使用動態儲存的磁碟我們稱為動態磁碟 (Dynamic Disk)，動態磁碟又可分為簡單磁碟區 (Simple volumes)、跨越磁碟區 (Snapped volumes)、鏡像磁碟區 (Mirror volumes)、等量磁碟區 (Striped volumes)、RAID-5 磁碟區，這些磁碟區類型有部份與大家熟知的磁碟陣列 (Redundent Array of Independent Disks，RAID)有點類似，將會在後面一一介紹。

4.2 基本磁碟

Windows 2000 中，磁碟的儲存類型預設是基本儲存，基本儲存是 Windows 2000 為了與舊系統相容的一種儲存類型，而使用基本儲存的磁碟我們又稱為基本磁碟。基本磁碟可以包含磁碟分割和含有邏輯磁碟區的延伸磁碟分割。

4.2.1 基本磁碟的磁碟分割

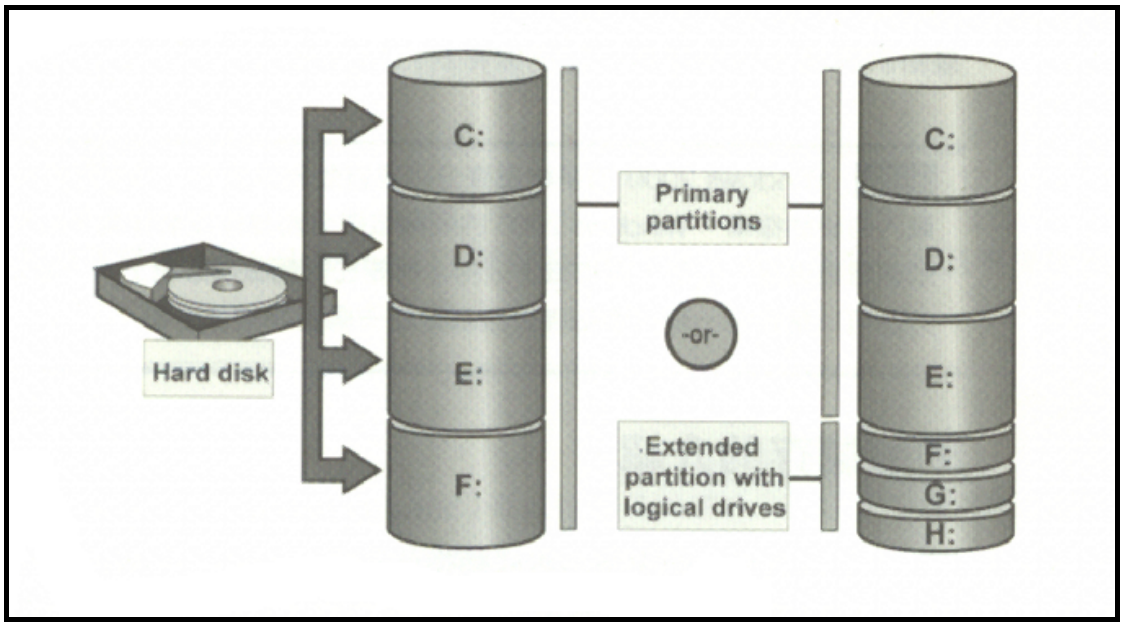
基本磁碟包括了兩種磁碟分割：主要 (Primary) 磁碟分割和延伸 (Extends) 磁碟分割。一個基本磁碟最多只能有四個分割區，而四個分割區中最多也只能有一個延伸磁碟分割，換句話說，一個基本磁碟的分割區可以是四個或四個以下的主要磁碟分割，也可以是三個或三個以下的主要分割和一個延伸磁碟分割。

4.2.2 主要磁碟分割：

大部分的作業系統，都需要一個主要磁碟分割來啟動電腦，電腦會先從唯一被指定為啟動 (active)的主要磁碟分割中，尋找開機檔以便啟動作業系統。

4.2.3 延伸磁碟分割：

延伸磁碟分割提供了解決分割區數量限制的方法，我們可以在延伸磁碟分割中分割出許多邏輯磁碟區，這些邏輯磁碟區使用起來就像主要磁碟分割一樣，檔案的管理也相對的方便。一般都是在分割完主要磁碟分割後才能分割延伸磁碟分割，所以我們要儘量的將剩餘的磁碟空間加入延伸磁碟分割，達到磁碟空間最有效的使用率。



4.3 動態磁碟

動態磁碟是使用 Windows 2000 動態儲存類型的磁碟，是由基本磁碟轉換而來的，動態磁碟也可以轉換回基本磁碟，不過必須先刪除動態磁碟上的所有磁碟區。動態磁碟可以包含動態磁碟區，也可以是一顆或多顆的實體硬碟的部分、或是好幾個部分。一個動態磁碟包括了簡單磁碟區、跨越磁碟區、鏡像磁碟區、等量磁碟區、RAID-5 磁碟區等五種類型。

4.3.1 動態磁碟的磁碟分割

動態磁碟機的磁碟分割可分為四區，包括有簡單磁碟區、鏡像磁碟區、跨越磁碟區、等量磁碟區，介紹如下。

4.3.2 簡單磁碟區 (Simple Volumes)：

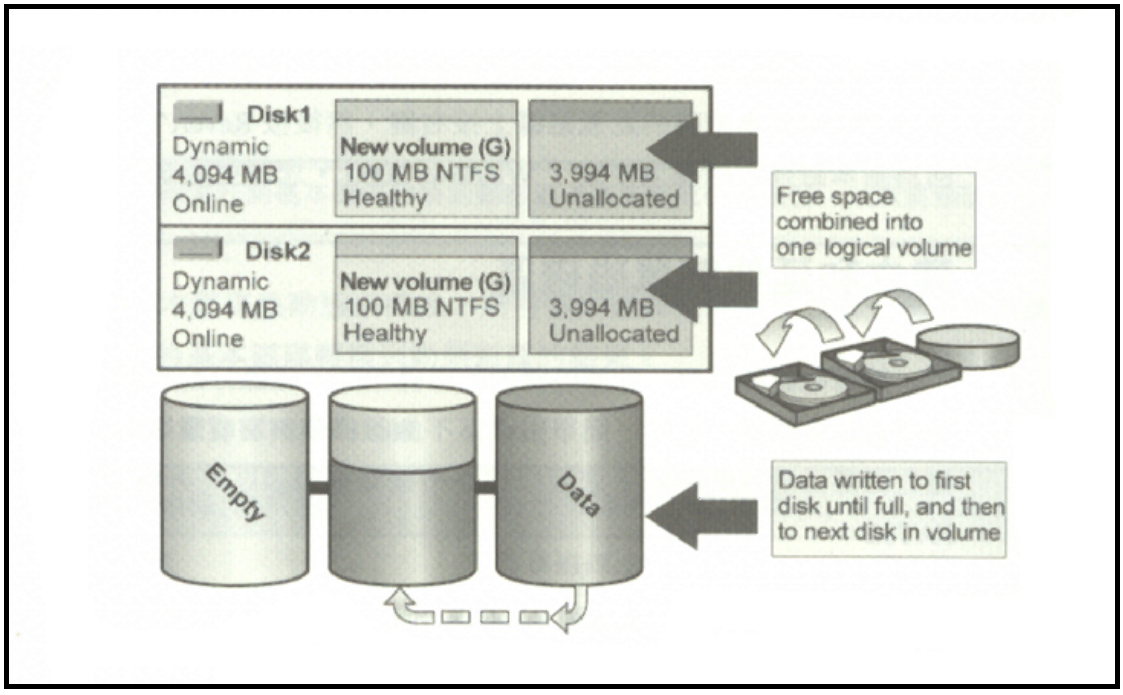
簡單磁碟區是在單一磁碟上的空間，本身不提供容錯，但可以利用鏡像磁碟區來作容錯。

4.3.3 鏡像磁碟區 (Mirror Volumes)：

鏡像磁碟區由兩個大小、內容相同但分屬於兩個不同的磁碟的簡單磁碟區所構成，當有一個磁碟毀損，即可利用另一個磁碟上的鏡像，達到容錯的效果。(RAID-1)

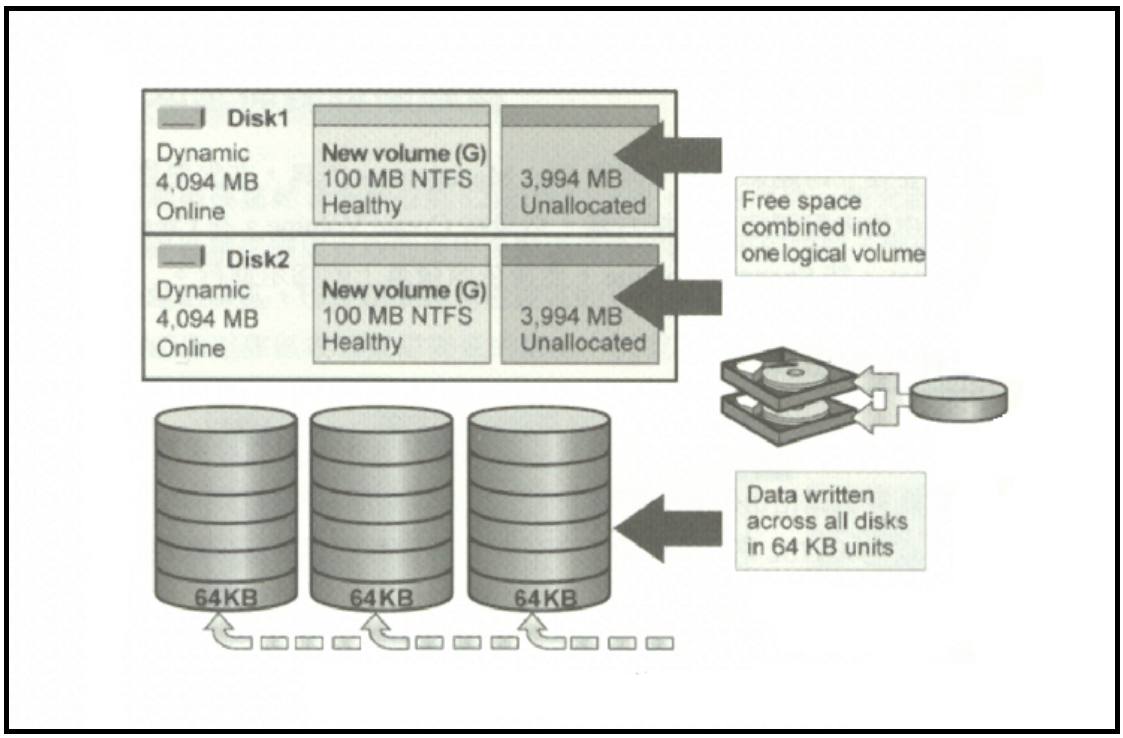
4.3.4 跨越磁碟區 (Snapped Volumes)：

跨越磁碟區可以包含兩顆或以上(最多 32 顆)的磁碟，而且不論磁碟的空間是否為連續空間。系統會先將資料寫入跨越磁碟區中第一個磁碟，直到第一個磁碟空間用盡，在寫入第二個…依此類推，也因此，倘若跨越磁碟區中有某個磁碟毀損，整個磁碟區中的資料將無法保存。



4.3.5 等量磁碟區 (Striped Volumes)：

等量磁碟區跟跨越磁碟區一樣可以包含兩顆或以上(最多 32 顆)的磁碟，不過每一個磁碟上要用來建立等量磁碟區的磁碟區大小都必須相同，系統會將資料以等比例的方式，同時間寫入(或讀出)各磁碟，這樣的方式與 RAID-0 相似，加速了硬體的 I/O 時間。相同的，倘若等量磁碟區中有某個磁碟毀損，整個磁碟區中的資料將無法保存。(RAID-0)

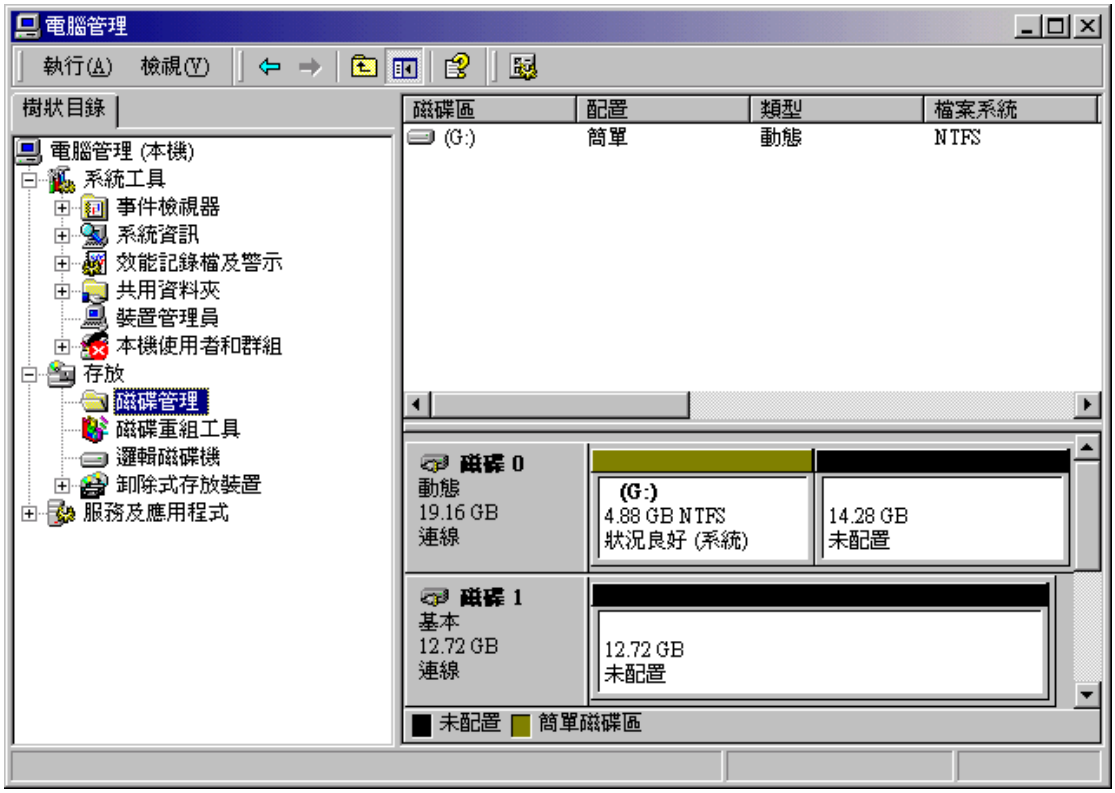


4.3.6 RAID-5 磁碟區：

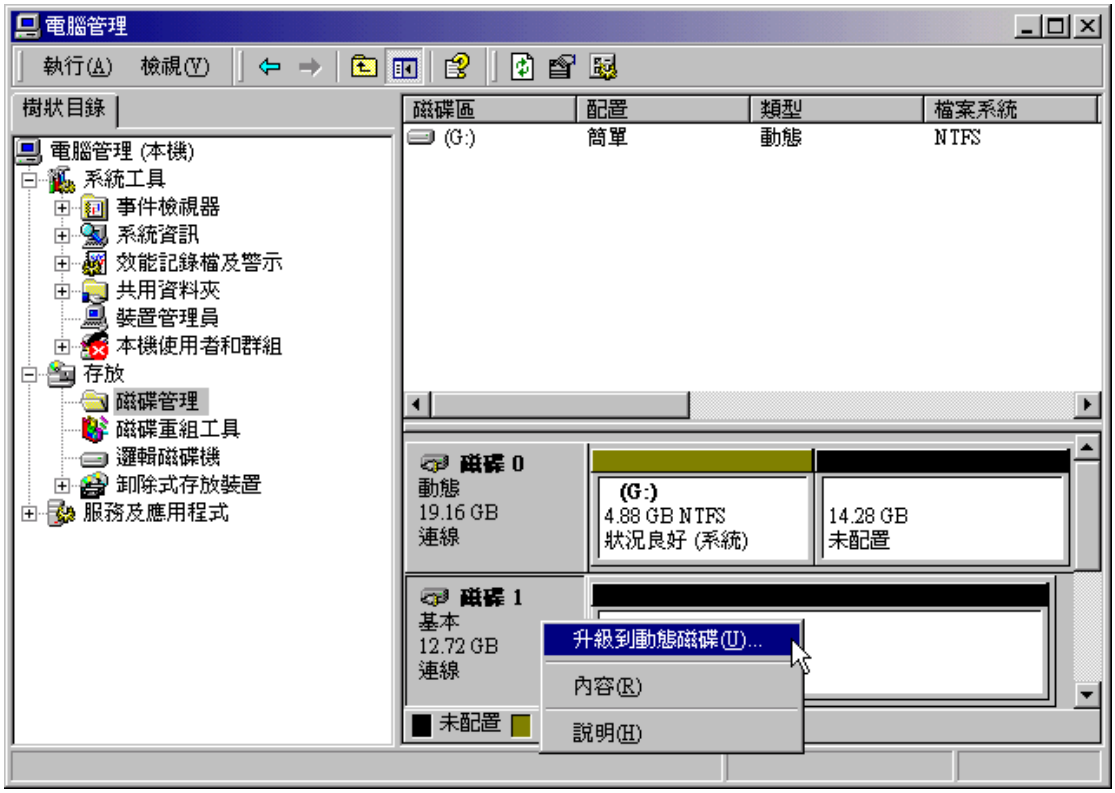
RAID-5 磁碟區是同時兼具容錯能力的等量磁碟區，利用同位檢查技術，在每次資料寫入前，先計算出同位檢查資訊。而同位檢查資訊將隨著資料分別散存在磁碟區所屬的各磁碟碟內。RAID-5 磁碟區是擁有資料及同位檢查等量，且跨越三個以上實體磁碟的可容錯磁碟區。同位檢查資訊可以在硬體失敗之後用來重建資料。如果部份實體磁碟失敗，則 Windows 會以其餘的資料及同位檢查資訊來重新建立位於失敗部份的資料。您只能在動態磁碟上建立 RAID-5 磁碟區，而且無法鏡像或延伸 RAID-5 磁碟區。

4.4 安裝

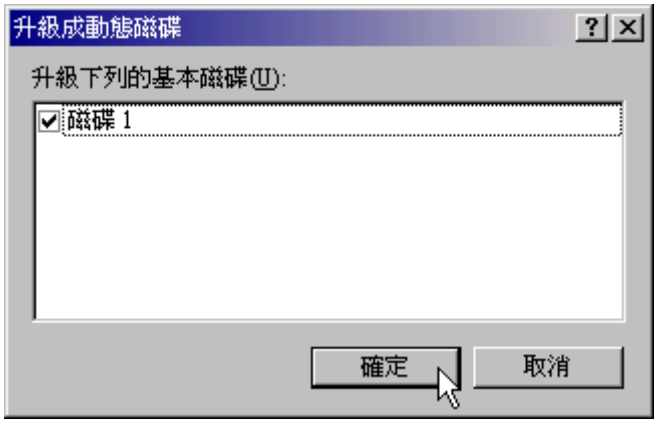
步驟一：開啟「系統管理工具」→「電腦管理」。



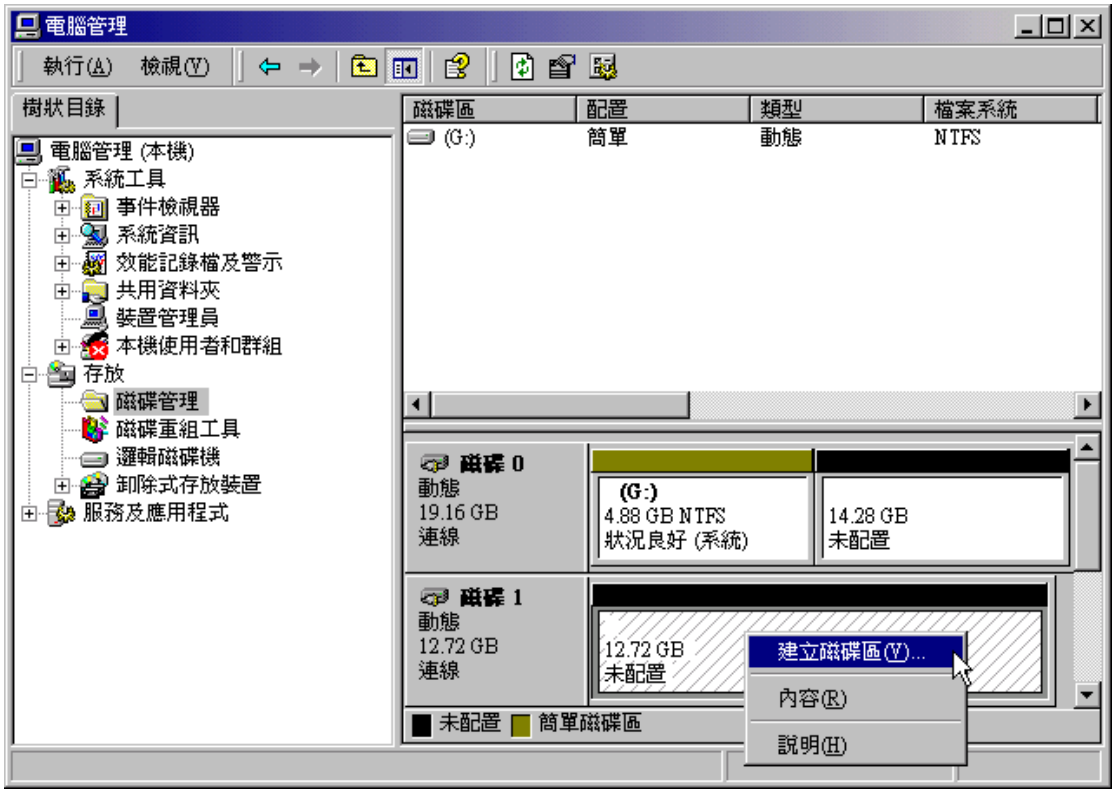
步驟二：在基本磁碟上按滑鼠右鍵，並點選升級到動態磁碟。



步驟三：選擇系統中要升級為動態磁碟的基本磁碟。

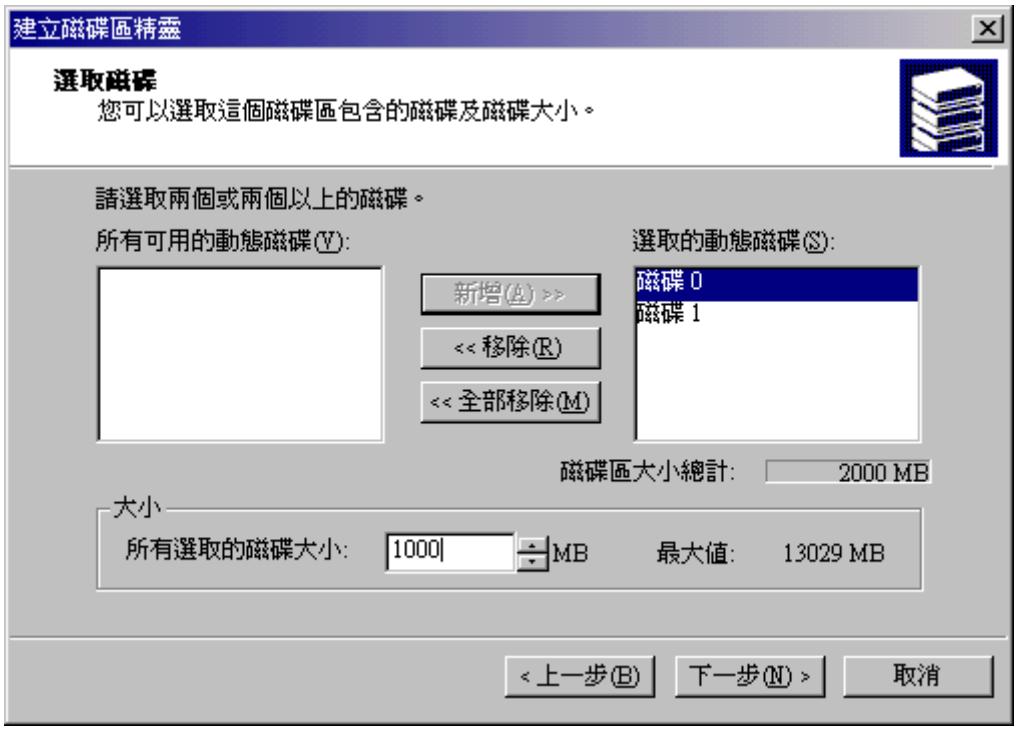


步驟四：在動態磁碟上按滑鼠右鍵，並點選建立磁碟區，在「歡迎使用建立磁碟區精靈」畫面點選「下一步」。

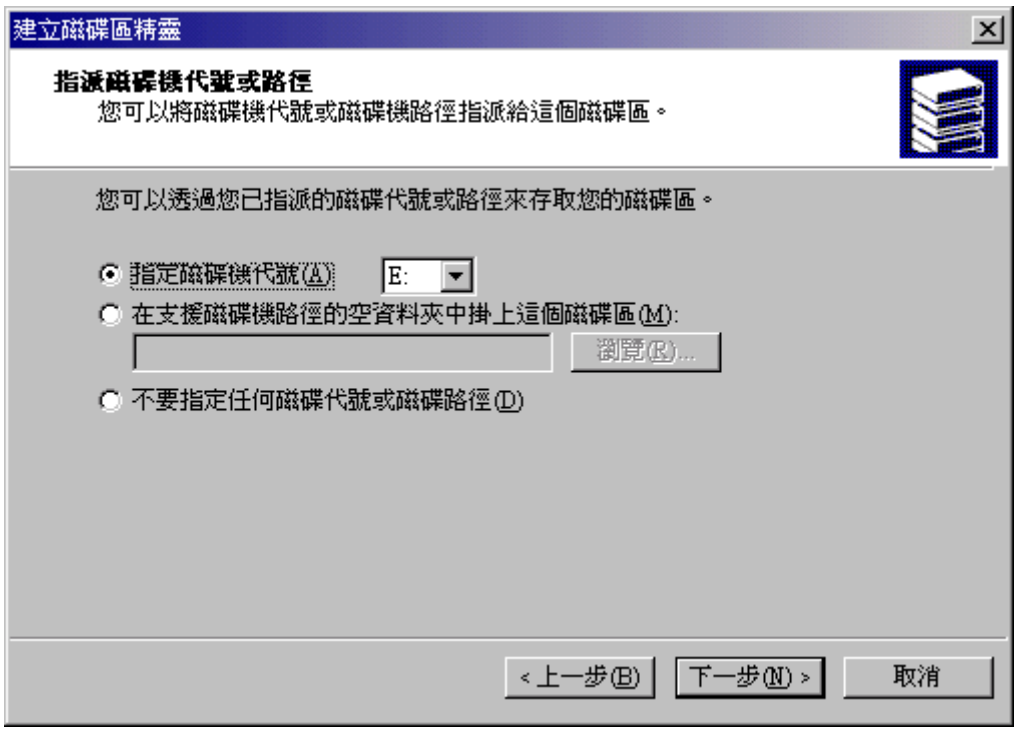

電腦管理
執行(A)
檢視(V)
樹狀目錄
電腦管理 (本機)
系統工具
事件檢視器
系統資訊
效能記錄檔及警示
共用資料夾
裝置管理員
本機使用者和群組
存放
磁碟管理
磁碟重組工具
邏輯磁碟機
卸除式存放裝置
服務及應用程式
磁碟區
配置
類型
檔案系統
(G:)
簡單
動態
NTFS
磁碟 0
動態
19.16 GB
連線
(G:)
4.88 GB NTFS
狀況良好 (系統)
14.28 GB
未配置
磁碟 1
動態
12.72 GB
連線
12.72 GB
未配置
建立磁碟區(V)...
內容(R)
說明(H)
未配置
簡單磁碟區

步驟五：選取要建立的磁碟區類型。(RAID-5 磁碟區因為系統磁碟數量未達三個，故無法選取。)

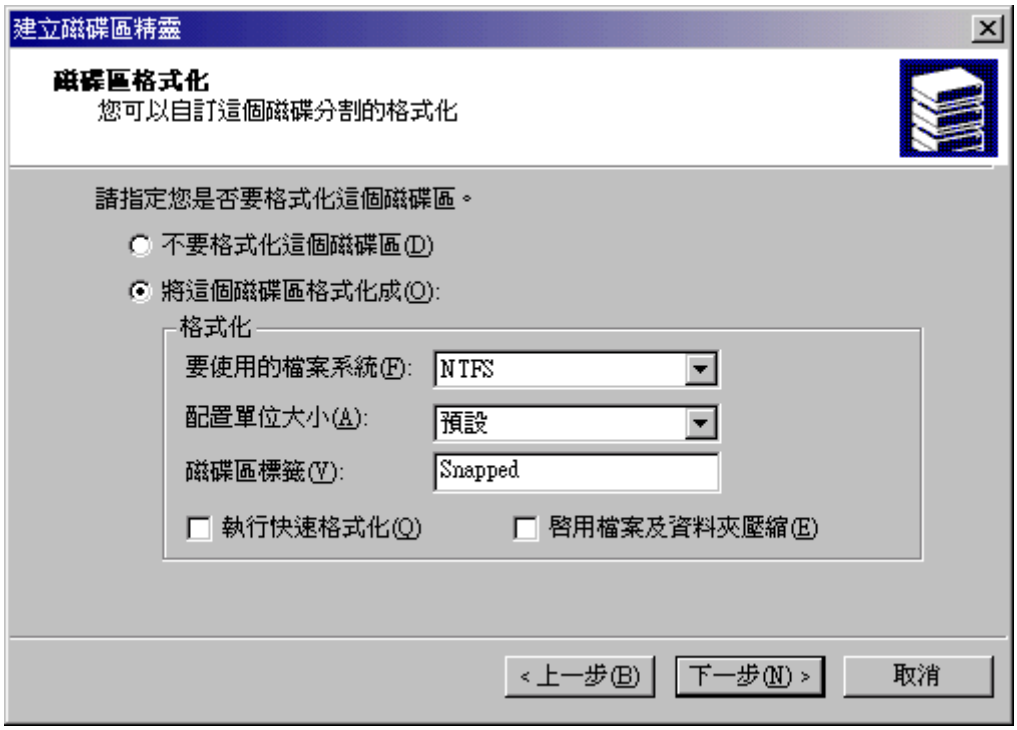
步驟六：選擇要建立的磁碟區大小及磁碟區所屬磁碟。



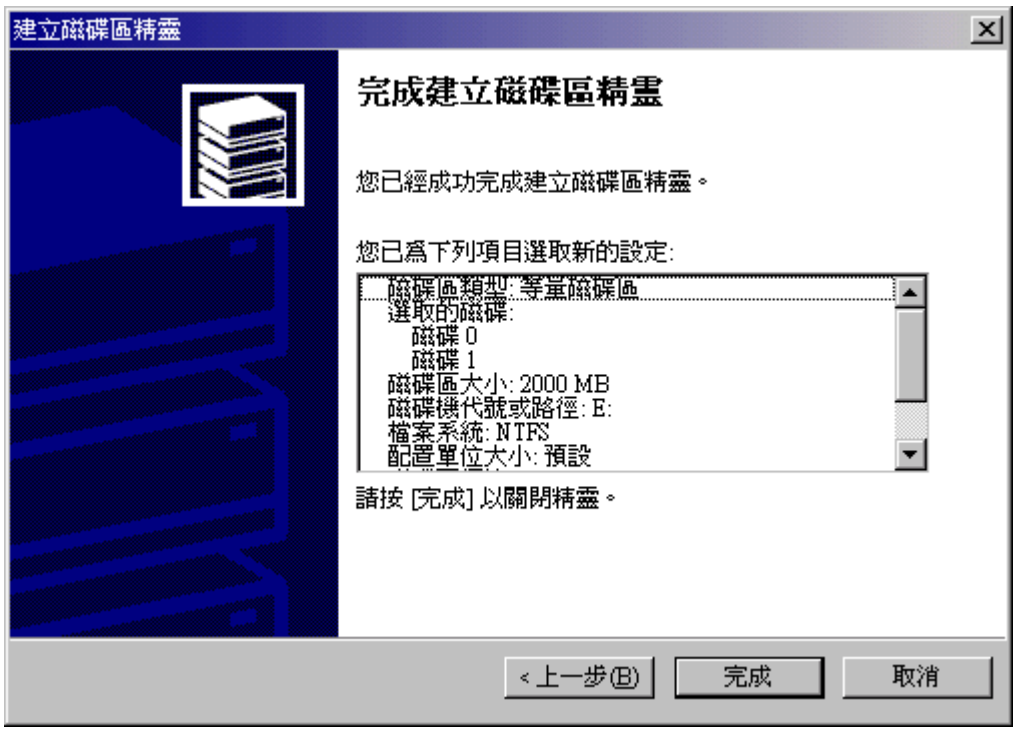
步驟七：指定磁碟區代號。



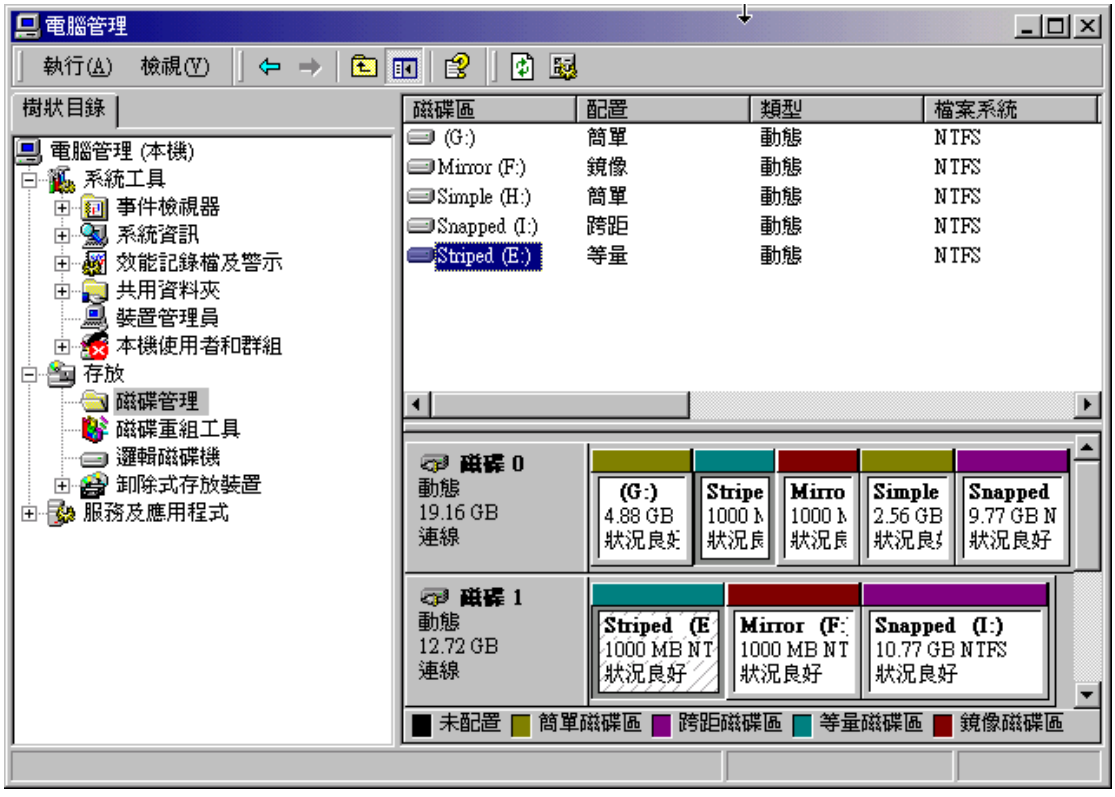
步驟八：選擇是否在建立完磁碟區後立即格式化及格式化屬性。



步驟九：列出各項設定的摘要，點選完成後即完成建立磁碟區的工作。



步驟十：建立完磁碟區後，各種磁碟區類型會以不同的顏色表示。



第5章 Active Directory

5.1 簡介

Active Directory 是 Windows 2000 針對目錄服務所提出的一項即為重要的服務元件，所謂的目錄服務就是指將許多分散在網路上資源(如：應用程式、檔案、印表機、管理員與使用者等)的相關資訊紀錄起來，如同電話簿儲存電話用戶的相關資訊，增加彼此間的互動，更能掌握網路資源的完整性與隱私性，對系統管理者來說，內部網路架構、系統整體管理、使用者的控管方面，都不再是繁碎的瑣事。Active Directory 支援多種定義完善的通訊協定及格式，提供功能強大、具有彈性且易於使用的 API。Active Directory 是安全的、可分布、可磁碟分割及可複製的。無論多大的安裝，它都可正常地工作，從具備數百個物件的單一伺服器到有百萬個物件的數千台伺服器都一樣。

5.2 Active Directory 特點

Active Directory 的特點為，易於管理、安全性的加強及延伸交互運作的能力等三項，介紹如下。

5.2.1 易於管理

分散式系統的管理，通常都會有一些浪費時間或是多餘的程序，例如新增應用軟體或有新同事的加入，系統管理員就必須為這些新的元件另外建立適當的目錄，或是修改其他紀錄；Active Directory 提供單一可以紀錄使用者、群組、網路資源等資訊的空間，減少了管理人員的負擔。

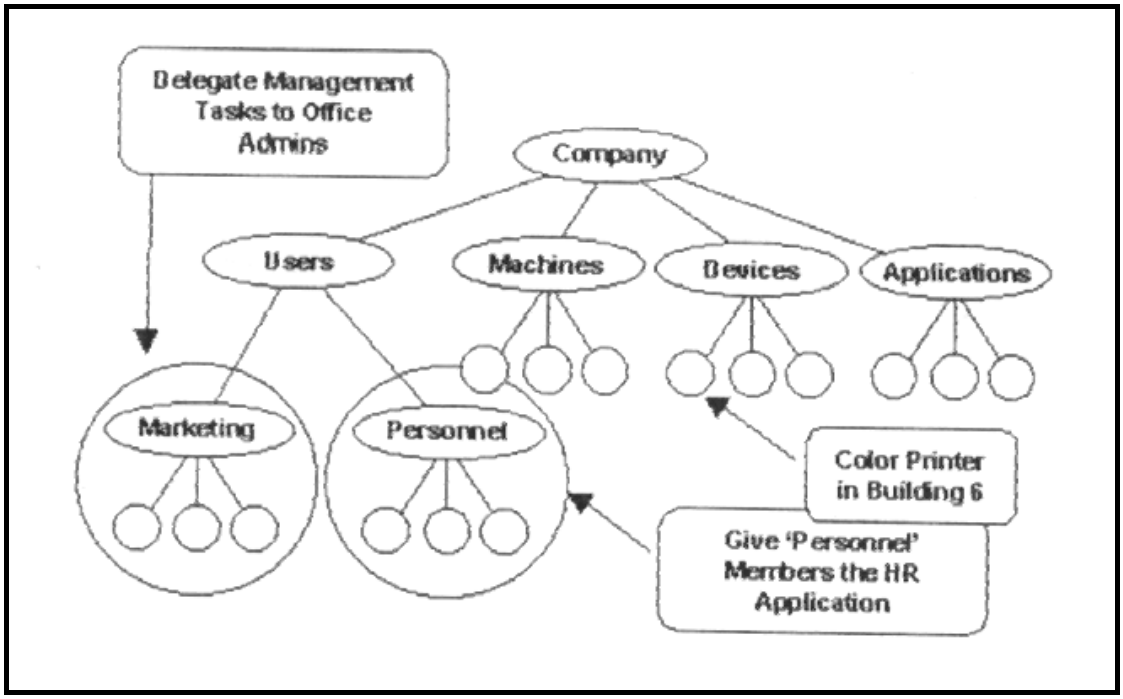
- **減少多餘的管理工作**
  將系統中，使用者帳號、用戶端、伺服器及應用程式一起管理，不僅管理起來方便，也具備目錄的同步性。

- **使用者桌面管理**
  能將軟體自動依使用者的權限，指定到其所使用電腦的桌面，完成適當的設定，使軟體安裝更加的方便。

- **資源更有效的利用**

系統管理者能將各管理功能，授權給組織內部各個不同部門階層，減少管理者的負擔。

Active Directory 利用將使用者及網路資源的組織結構化，讓系統管理員能由單一點切入，以類似容器的觀念，直接對網路上多個群組或物件進行管理。



[Windows 2000 Server 網域建制與安全維護]

Active Directory 能讓管理者對個別的使用者或特定的群組，授與不同的管理權限跟工作，使得系統的管理更具彈性。例如：授權給某個部門的主管，允許它擁有重設使用者密碼的權限，而更具特權性的功能，如建立使用者帳號，則保留給更高階層的管理員。另外，Active Directory 也可以照使用者的身分，自動的安裝指定的應用程式，將軟體分配到該使用者登入的電腦中。

對於使用者來說，因為目錄服務會紀錄網路資源物件的屬性，所以 Active Directory 可以讓使用者直接使用這些資源。例如：印表機的使用，目錄服務會安裝好該印表機的驅動程式，使用者只要利用 Windows 的「尋找」，找到該台印表機，就可以直接使用。

5.2.2 安全性的加強

在分散式要求共同運作的網路環境裡，安全性的考量特別受到重視，管理起來也特別容易出錯，Active Directory 改採用集中式的管理，針對不同的使用

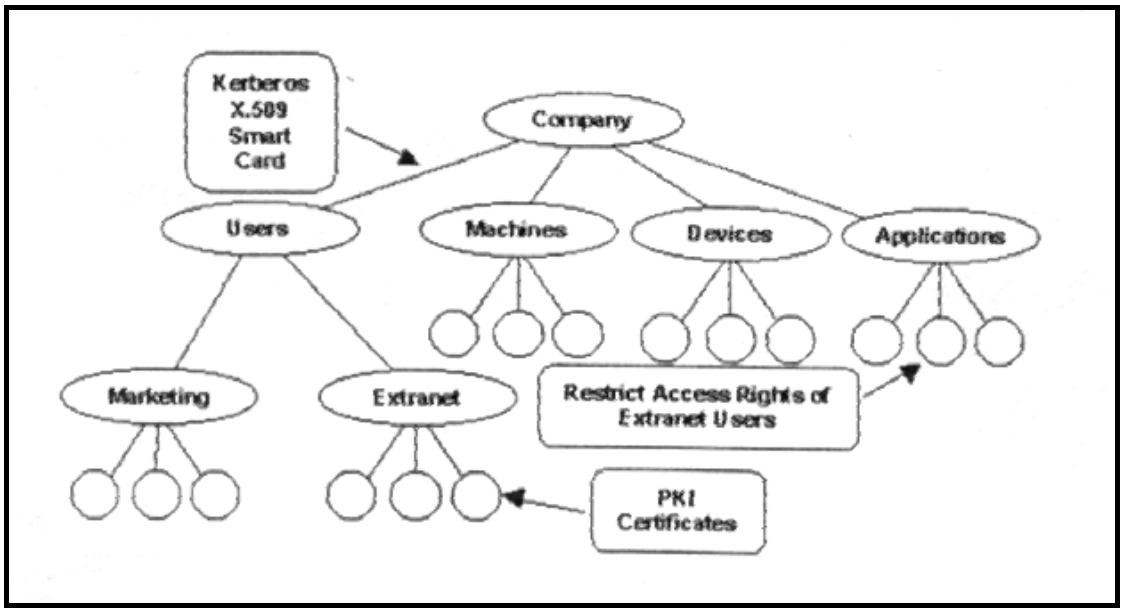
者，採用多種不同的安全措施，如 Kerberos、X.509 及智慧卡（smart card），對於公司內部的使用者、遠端撥接的使用者、外部進行電子商務的客戶，提供不同類型的存取控制。

- **整合的安全服務**
  改進密碼的安全性與管理，讓使用者在登入網路資源時具備透通性(Transparent)。

- **確保桌面功能**
  針對不同身分的使用者，限制桌面的設定並防止用戶端特定的操作，如軟體安裝或登錄檔的編輯。

- **加快電子商務的部署**
  內建的網路安全標準協定與認證機制，如：Kerberos、公開金鑰架構(Public Key Infrastructure，PKI)和在 Secure Sockets Layer (SSL)上執行的 LDAP (Lightweight Directory Access Protocol)，讓企業在電子商務的部署上更佳的便利、快速。

- **精確的網路安全控制**
  針對目錄物件及其內部的資料，設定存取控制的權限。



[Windows 2000 Server 網域建制與安全維護]

Windows 2000 Server 架構透過 Active Directory 與進階安全控制的整合，使得資料的防護有更新的層級。Active Directory 在使用者認證的管理及網路資源的存取控制上，提供多種登入 Windows 2000 的認證方法，而且在登入後，

資源的存取也是單一的認證方式。支援的網路安全標準協定與認證機制，能讓企業安全的將所選取的目錄資訊提供給防火牆外的使用者及電子商務的客戶，大大的增進安全上的防護。

5.2.3 延伸交互運作的能力

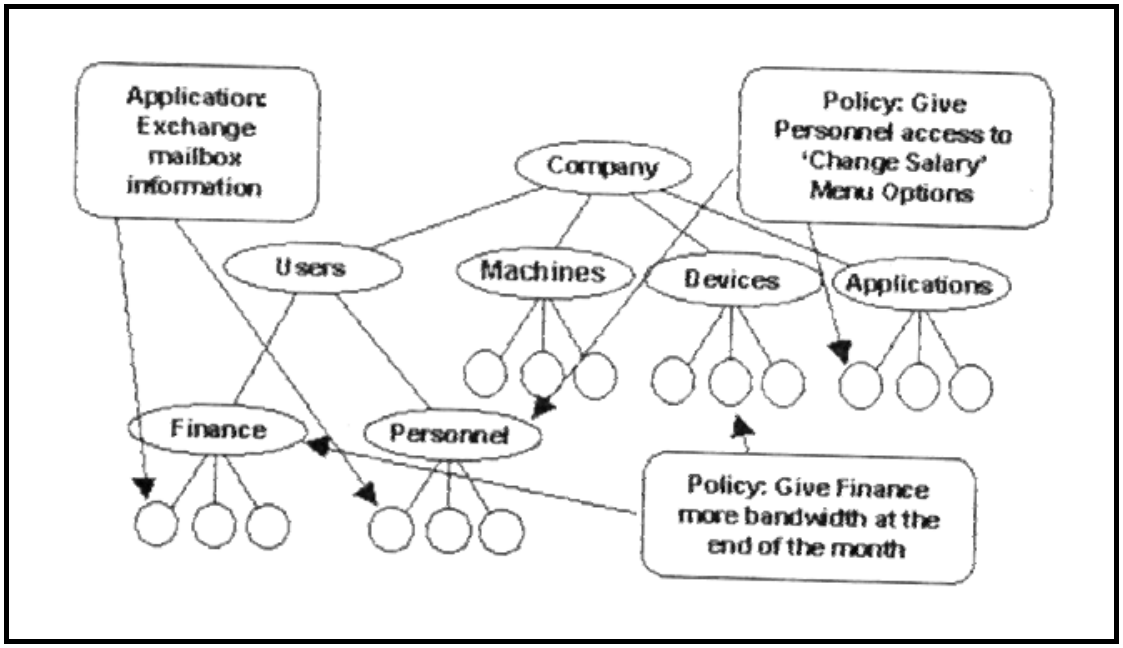
Active Directory 提供了一個標準化的介面與開放式的同步化機制，讓各式各樣的應用程式及設備整合在一起，透過交互運作的網路系統將功能發揮出來。例如：電子商務應用程式、防火牆、電子郵件…等，都具備這樣的特性。

- **標準化的介面**
  藉由標準化的介面，對於現存已投資的軟硬體功能，能夠確保其在往後的新應用程式與系統架構中，保持彈性。

- **應用程式的合併管理**
  開放式的介面連結以及同步化的機制，系統管理人員能合併管理多種應用程式的目錄。

- **支援目錄的網路功能**
  能夠利用目錄讓系統管理人員根據使用者的身分，指定不同的網路服務品質 (Quality of Service，QoS)與頻寬。

- **可發展支援目錄的應用程式**
  使用能完全延伸的目錄架構，程式開發人員可以在發展應用程式時，增加一些依照使用者需要來分發程式的功能。



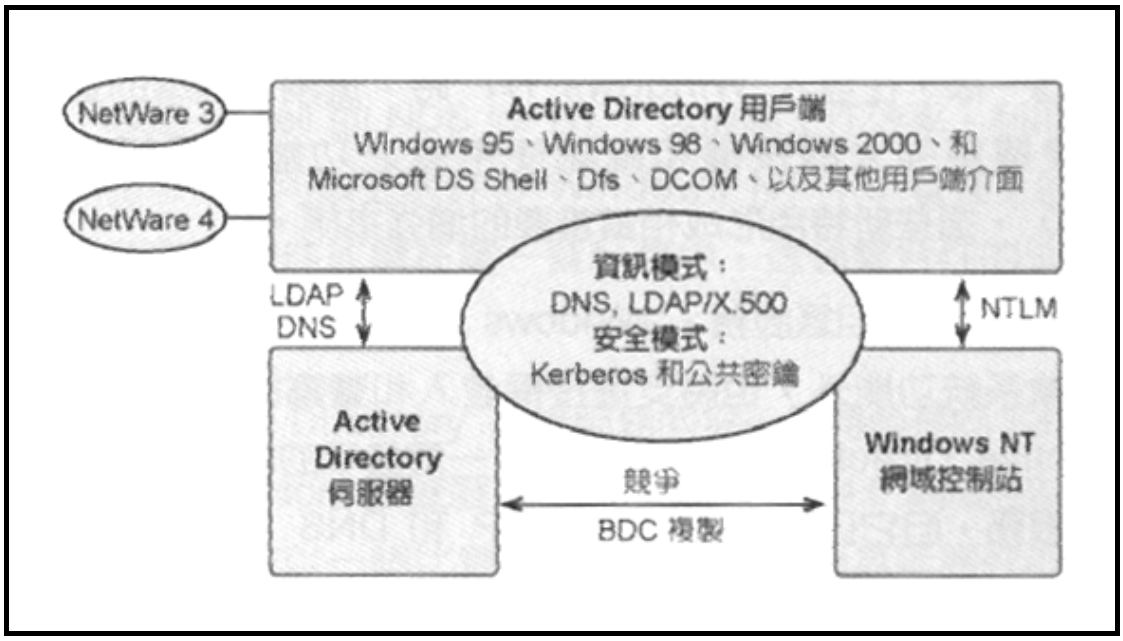
[Windows 2000 Server 網域建制與安全維護]

Active Directory 提供集中式的整合，讓目錄與管理的工作能加以合併，藉由目錄的標準介面，讓程式開發人員能整合目錄服務的功能與架構，來發展新的應用軟體。應用程式的開發人員能依照使用者的身分控制其權限，例如：一個允許使用目錄的應用程式，能參考使用者在其目錄中的資料，依照使用者的特性來提供特定項目及功能的選單。

另外，因為開放式的同步化機制，Active Directory 也能更進一步的延伸到 Windows 之外的環境去運作，將交互運作的功能應用到廣大範圍的應用程式與設備。

5.3 Active Directory 的架構概觀

Active Directory 目錄服務架構在 Windows 的用戶端與伺服端所擁有的元件是一樣的，支援了 LDAP/X.500 和 DNS 資訊模式，並同時支援此架構中的安全模式，如 Kerberos 和公開金鑰。Active Directory 目錄服務提供用戶端進入 Window NT LAN Manager (NTLM)，而 NetWare 3.X 與 NetWare 4.X 的使用者，在相同的伺服器層級下，可以向下跟 Windows NT 3.X 和 4.0 網路相容。

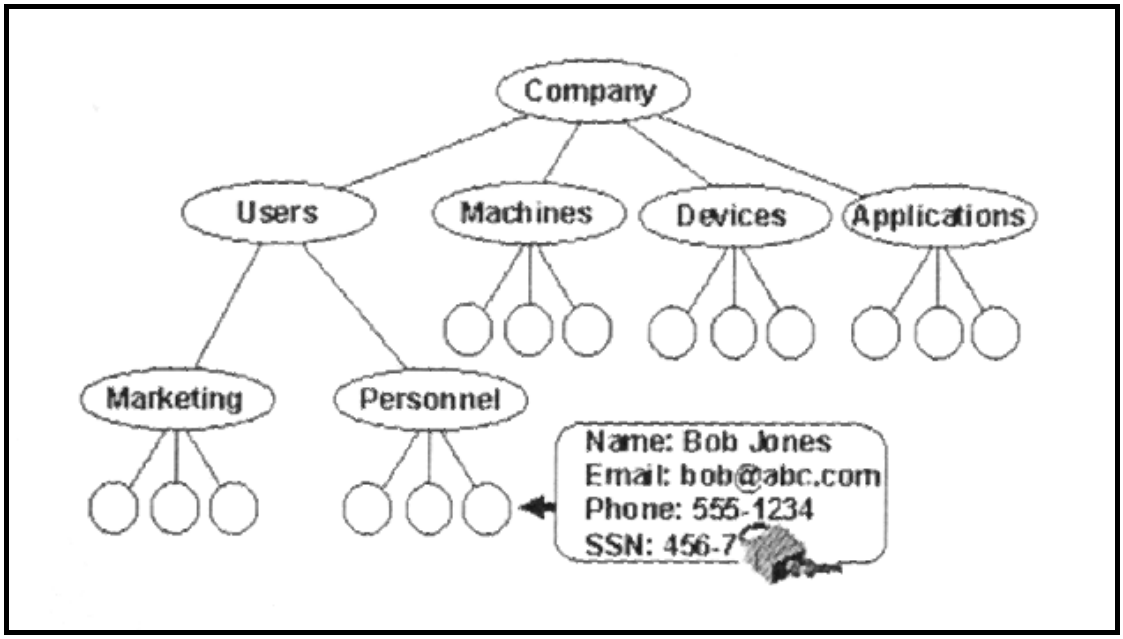


[認識 Active Directory Services 目錄服務]

5.4 Active Directory 的元件

正如前面所說，Active Directory 目錄服務儲存了網路元件的資訊。Active Directory 以它的名稱空間 (namespace)對應到網路上某個圓建的位置，讓使用者藉此找到特定的物件。Active Directory 提供了非常廣泛的網路物件名稱和其實際位置的解析，其中包括了使用者、系統及網路服務。

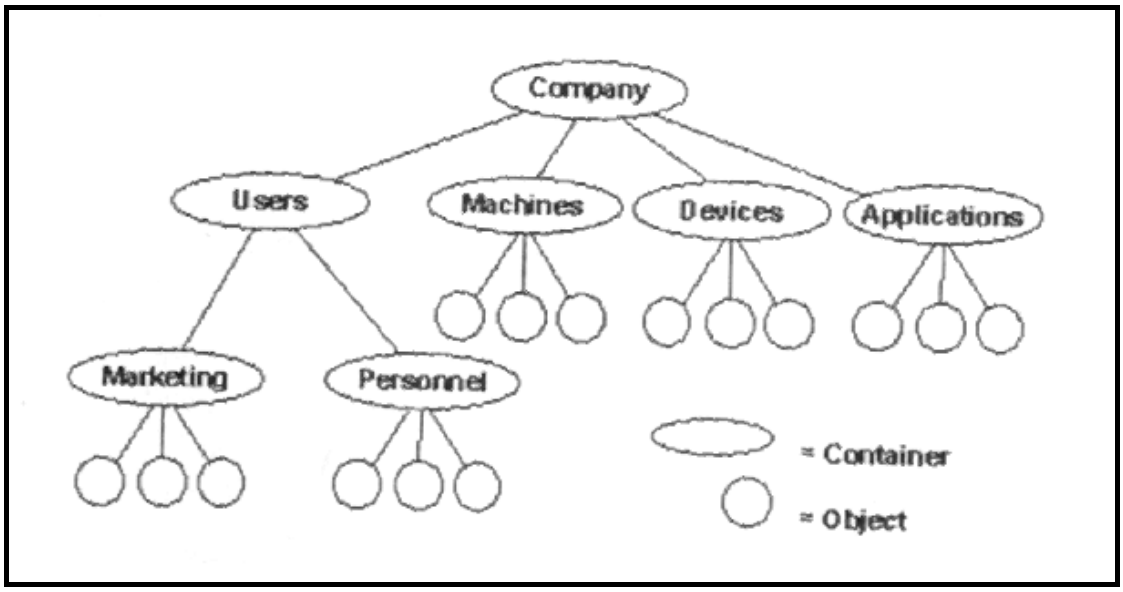
每一個 Active Directory 所追蹤的目標，都稱為物件 (object)，如使用者、系統、資源以及網路服務；屬性 (attribute)則是 Active Directory 系統對於物件的描述，如所有的「使用者」物件都是使用同一組屬性來儲存使用者名稱和其他相關資料，其他種類的物件則有其個別的屬性來負責描述。



[Windows 2000 Server 網域建制與安全維護]

我們將同一類型的物件所使用的一組屬性，總稱為架構 (schema)，它是所有物件類別的正式定義，擁有不同 schema 的物件群，即屬於不同的物件類別 (object class)。Schema 的實際儲存位置是在 Active Directory 中，因此系統管理員可以將屬性套用在物件類別上，讓他隨著 Active Directory 的複製動作散播到網域的每一個角落，而不須重新啟動任何一部網域控制器。

容器 (container)是一種用來組織 Active Directory 的特殊型態物件，並不代表任何的實體物件。容器物件中還可以擁有其他的容器。



[Windows 2000 Server 網域建制與安全維護]

目錄樹 (tree)是指 Active Directory 中的一組物件所組成的結構，而 forest 則是描述一組共用相同 schema、組態以及通用類別目錄 (Global Catalog) 且相互信任的目錄樹。

站台 (site)定義於 Active Directory 中的地理位置資訊，應用軟體可以透過站台，在網路中尋找實體位置最為接近的伺服器，以大幅降低廣域網路的交通流量。

5.5 通用類別目錄 (Global Catalog)

通用類別目錄 (Global Catalog，GC)是 Active Directory 所提供的一個尋找物件的服務，幫助擁有適當存取權的使用者尋找物件。它的機制遠勝於過去的「尋找電腦」功能，可以搜尋的對象包含了伺服器、印表機、使用者和應用軟體等 Active Directory 裡的任何物件。

因為 LDAP 名稱結構相當複雜，Global Catalog 的功能就顯得特別重要，Global Catalog 是存放在 Active Directory 伺服器的一份索引，記載了 Active Directory 所有的物件名稱及其使用者較常用來查詢的部份屬性。

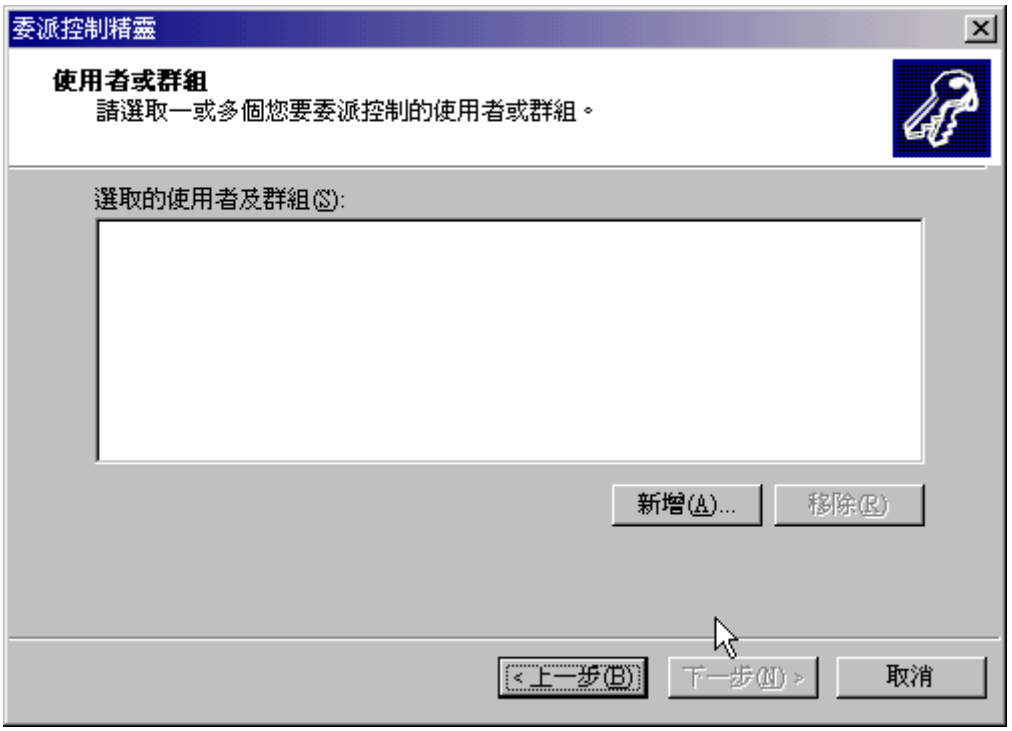
當物件被建立的時候，就會被指定一個獨有的號碼，叫做全球單一識別碼 (Globally Unique Identifier，GUID)，GUID 是一個 128 位元的識別碼，對使用者而言可能沒什麼特別的意義，但應用軟體可以將物件的 GUID 紀錄下來，然

後利用 Global Catalog 來進行追蹤。

5.6 委派管理

委派管理 (delegates administration)是 Windows 2000 Server 的新概念，系統管理員可以將某些特定的管理工作指派給其他使用者，而這個管理工作可能只牽涉到一小部份的系統和使用者。

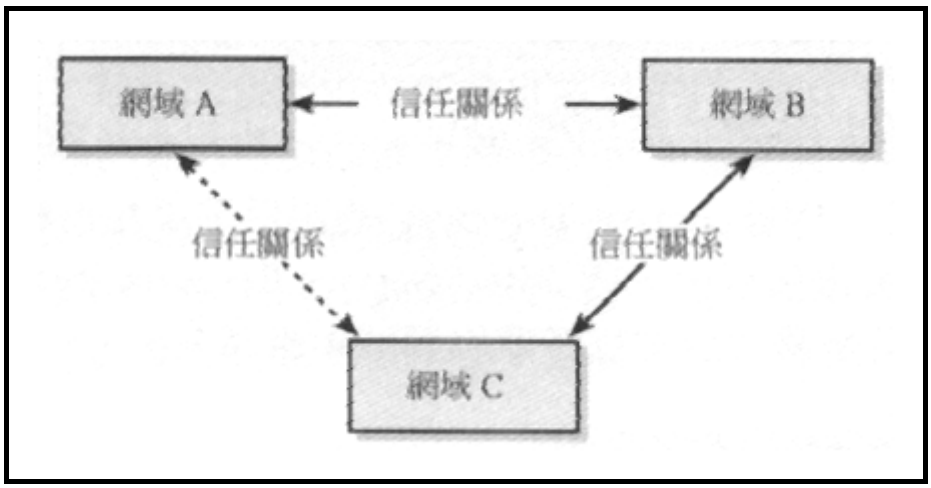
系統管理員可以透過「委派控制精靈」來設定存取控制清單 (Access Control List，ACL)，將存取權限指派給 Active Directory 當中的物件。要使用「委派控制精靈」，可以用滑鼠右鍵點選「Active Directory 使用者與電腦」裡的任何物件，然後選擇「委派控制」來啟動精靈。



「委派控制精靈」能夠讓系統管理員針對個別使用者和群組設定不同層次的物件存取權限和屬性，甚至可以將特定的屬性隱藏起來。例如，系統管理員可以指定只有部門主管能夠查閱其他使用者的住家電話號碼，而讓其他人員甚至不知道有這個屬性存在系統中。Active Directory 也支援繼承關係，任何新加入的物件都會繼承該容器的存取控制清單。

5.7 信任關係

雖然 Windows 2000 仍舊使用網域信任關係，但卻有著與 Windows NT 非常不一樣的機制，單向和雙向的信任關係再也沒有區別了；Active Directory 的信任關係都是雙向的，並且可以遷移。也就是說，如果網域 A 信任網域 B (B 網域也會信任 A 網域)、且網域 B 信任網域 C (C 網域也會信任 B 網域)，則網域 A 與網域 C 之間的信任關係就會自動成立。



[邁向 Windows 2000]

5.8 複製

Active Directory 服務必須在多部伺服器上同時運作，當有任何一部伺服器失效時，用戶端依舊能夠存取放置在其他伺服器上的資料副本，而資料庫的更新也將被發送到所有的 Active Directory 伺服器。雖然這種機制使得複製的程序複雜化，卻可以避免因為單一網域伺服器的失誤而中止了資料庫的更新，同時也降低了網域控制器的負擔。

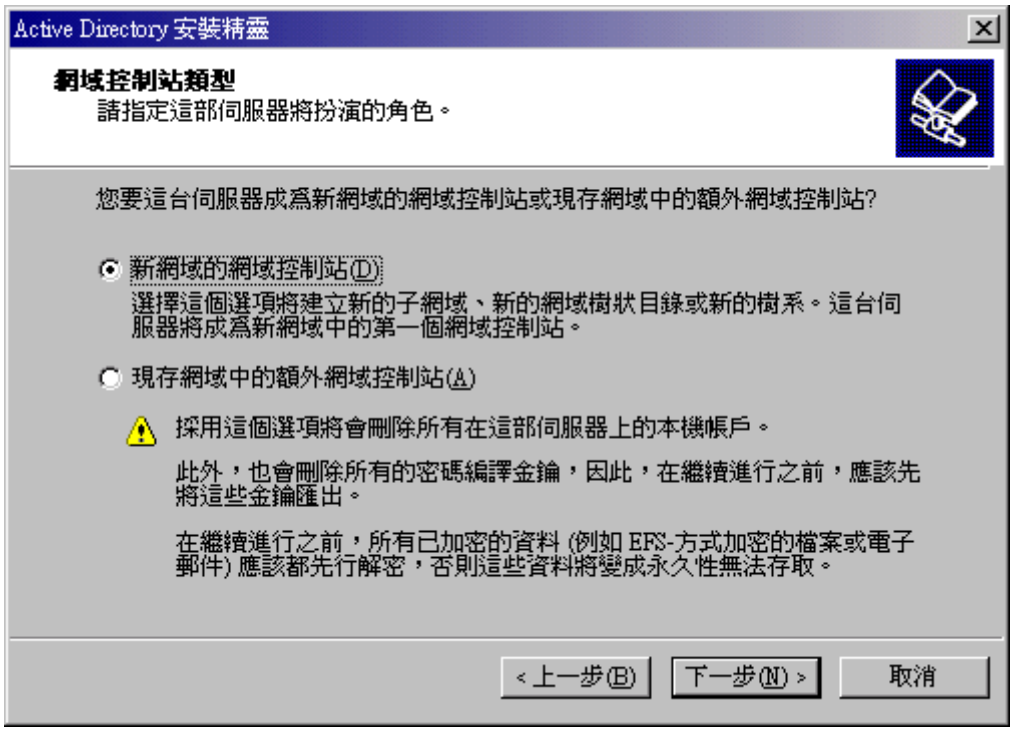
Active Directory 服務當中含有一個複製元件，簡化了系統管理員的作業，只要將網域控制器加入 Active Directory 網域，伺服器間就會自動的開始進行複製程序。

5.9 安裝

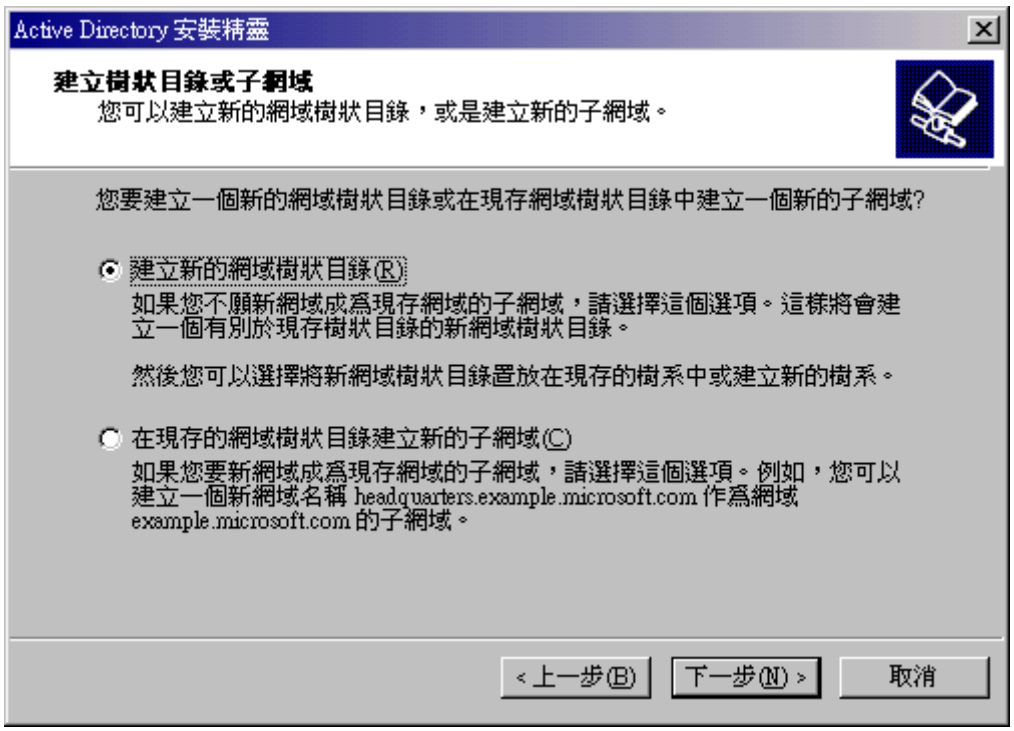
步驟一：點選「系統管理工具」→「設定您的伺服器」，在「Windows 2000 伺服器設定」畫面左邊點選「Active Directory」，即可開始進行安裝。

步驟二：點選下一步略過歡迎畫面。

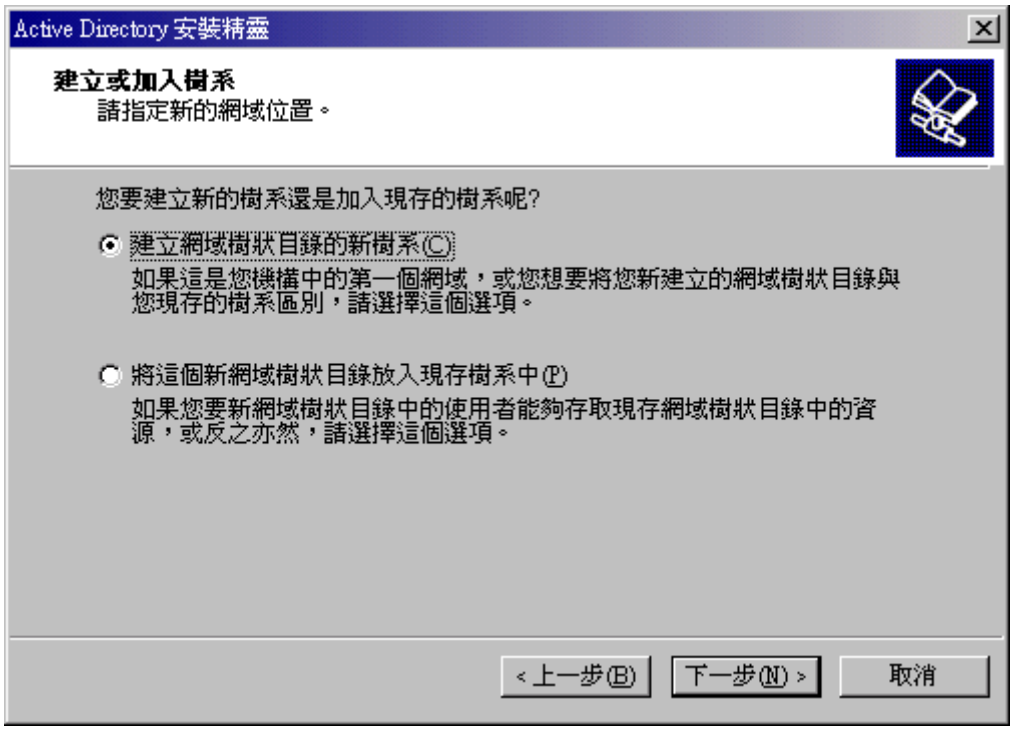
首先，先指定網域控制站的類型，若是網域已有 Active Directory 網域控制站，則選擇「現存網域中的額外網域控制站」。預設的選擇為「新網域的網域控制站」。



步驟三：選擇要建立新的樹狀目錄或子網域。

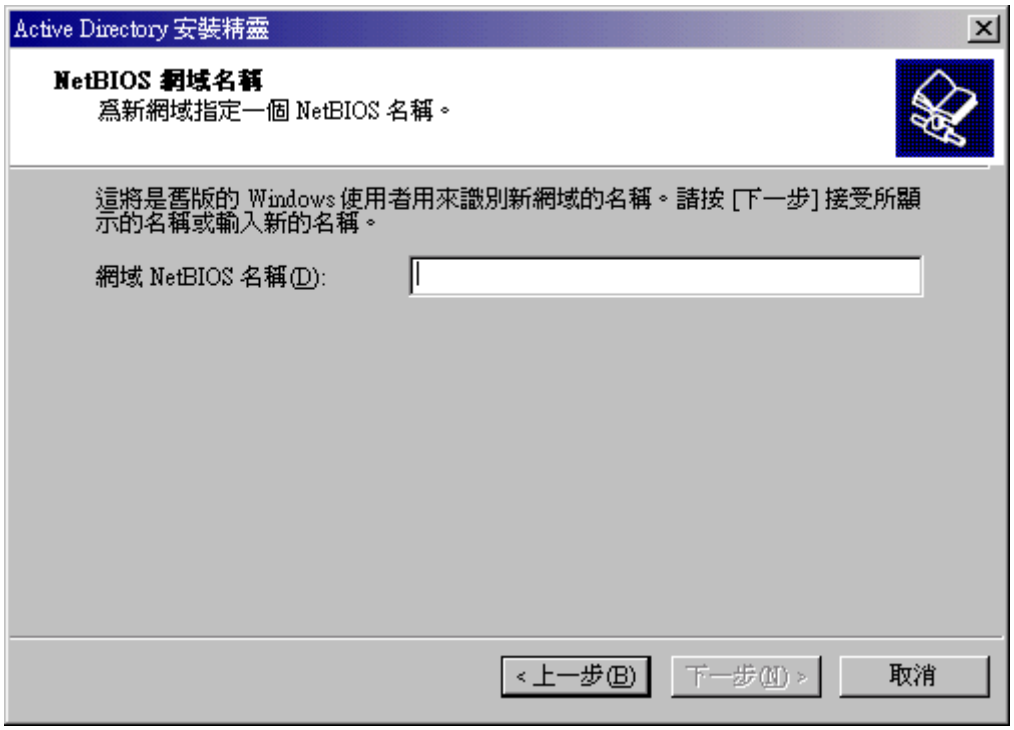


步驟四：選擇建立新樹系或是加入已經存在的樹系中。

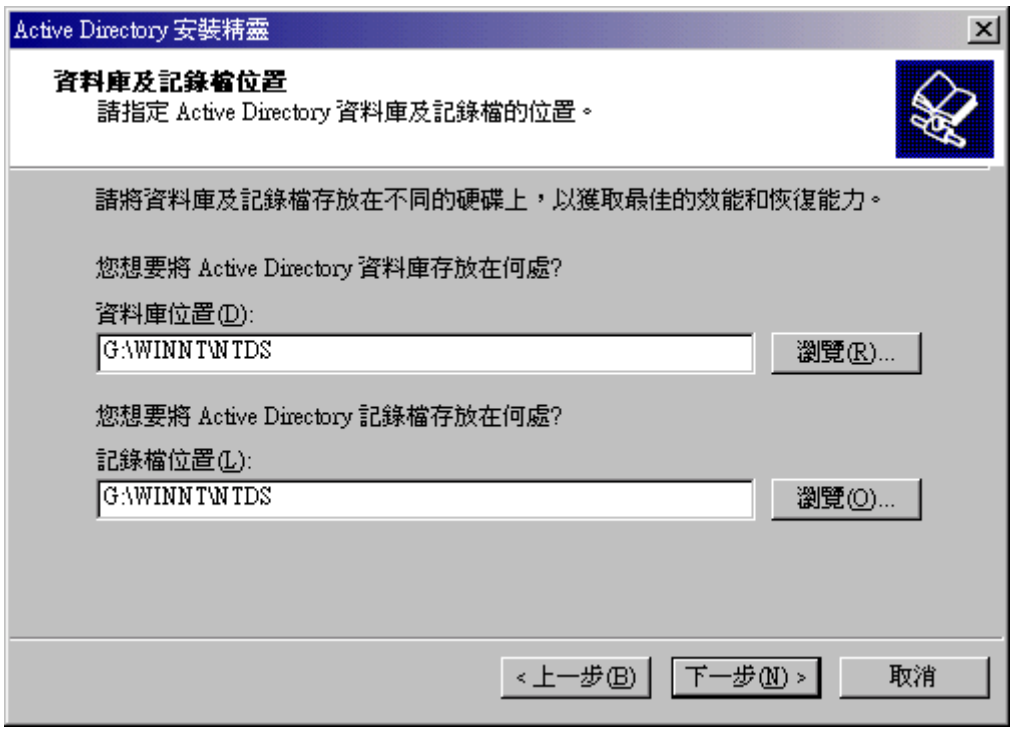


步驟五：因為在指定網域控制站的類型時選擇預設的「新網域的網域控制站」，所以要指定新網域的名稱。

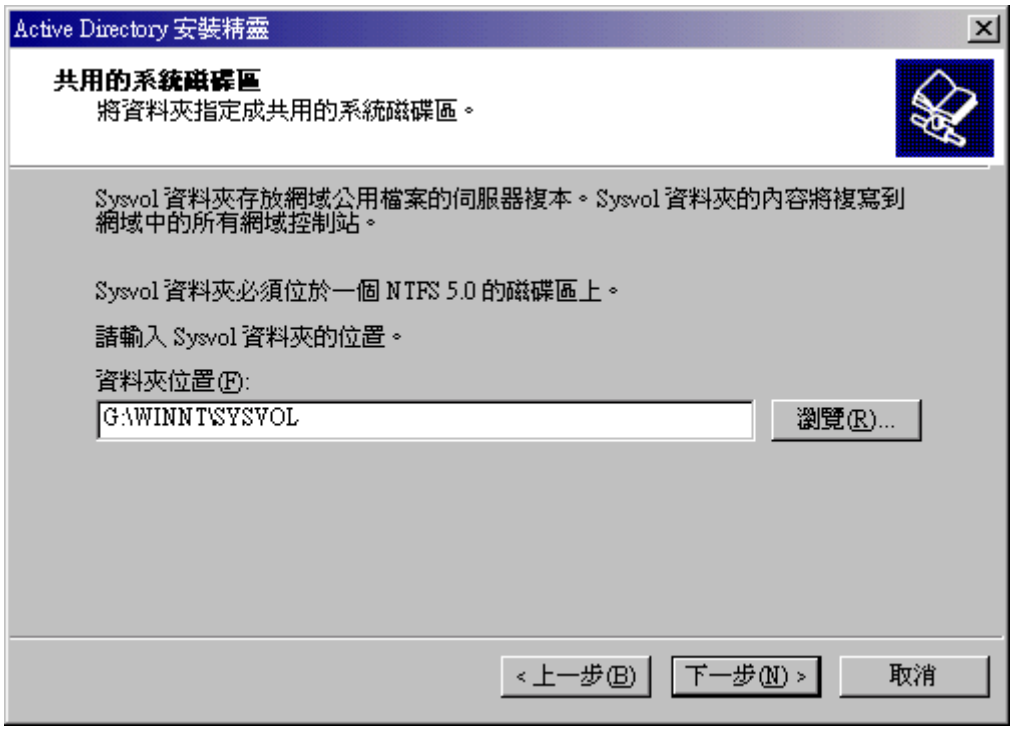
步驟六：同樣的，指定新網域的 NetBIOS 名稱。



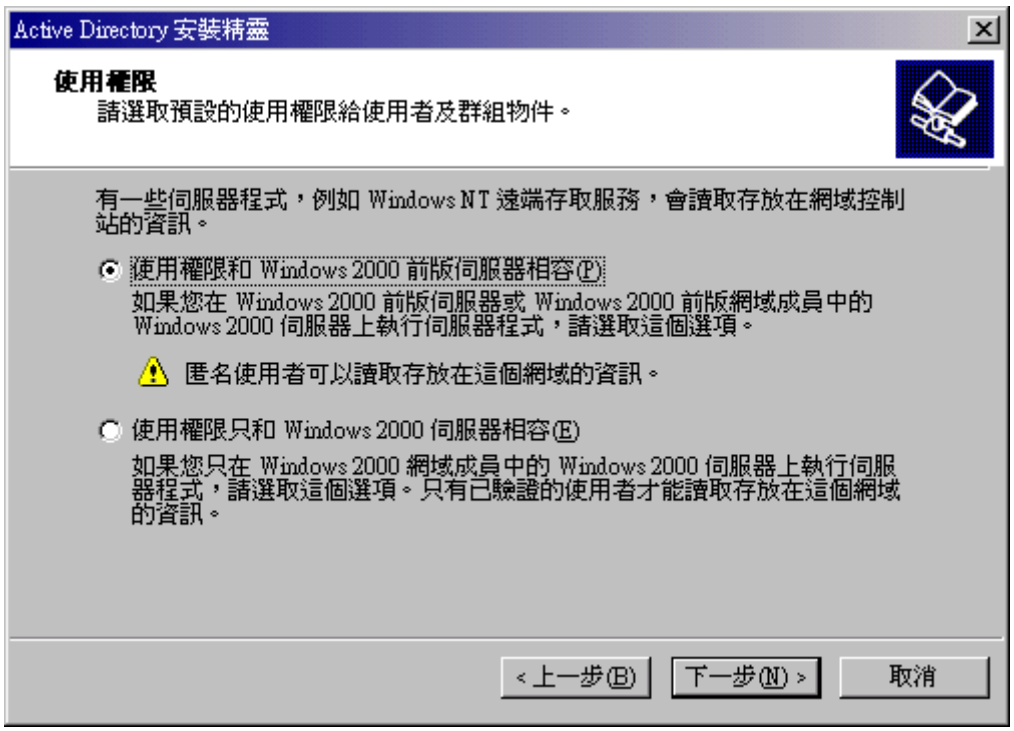
步驟七：指定 Active Directory 資料庫及紀錄檔存放的位置。



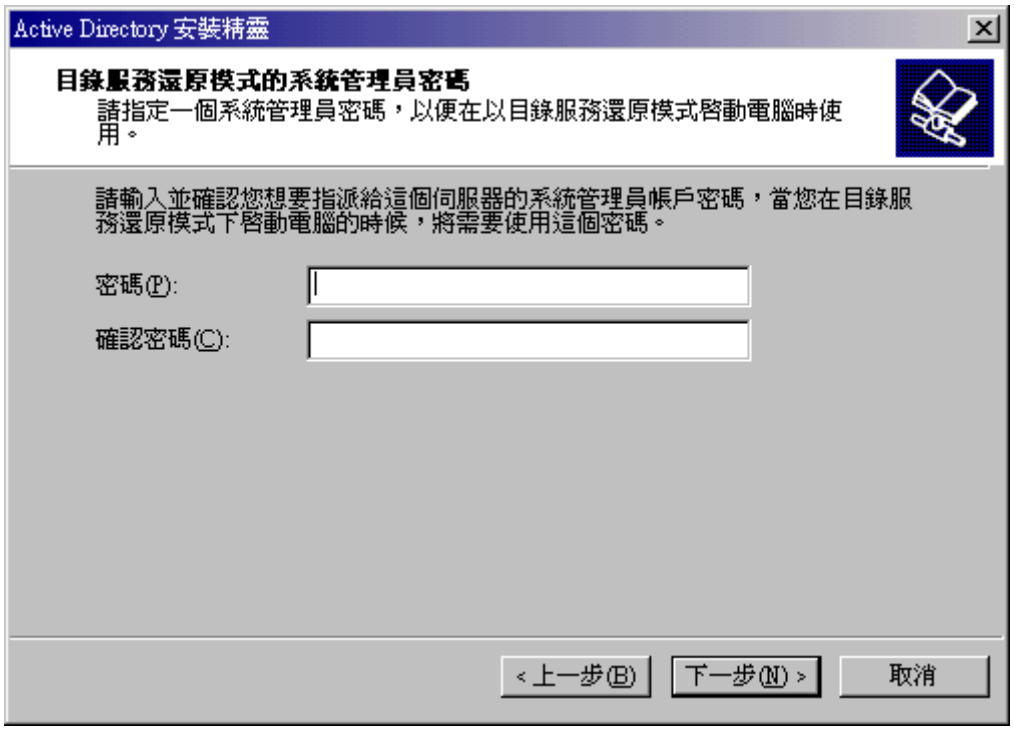
步驟八：指定 Active Directory 存放著網域公用檔案的伺服器副本的位置。



步驟九：選取預設的使用權限給使用者及群組物件。

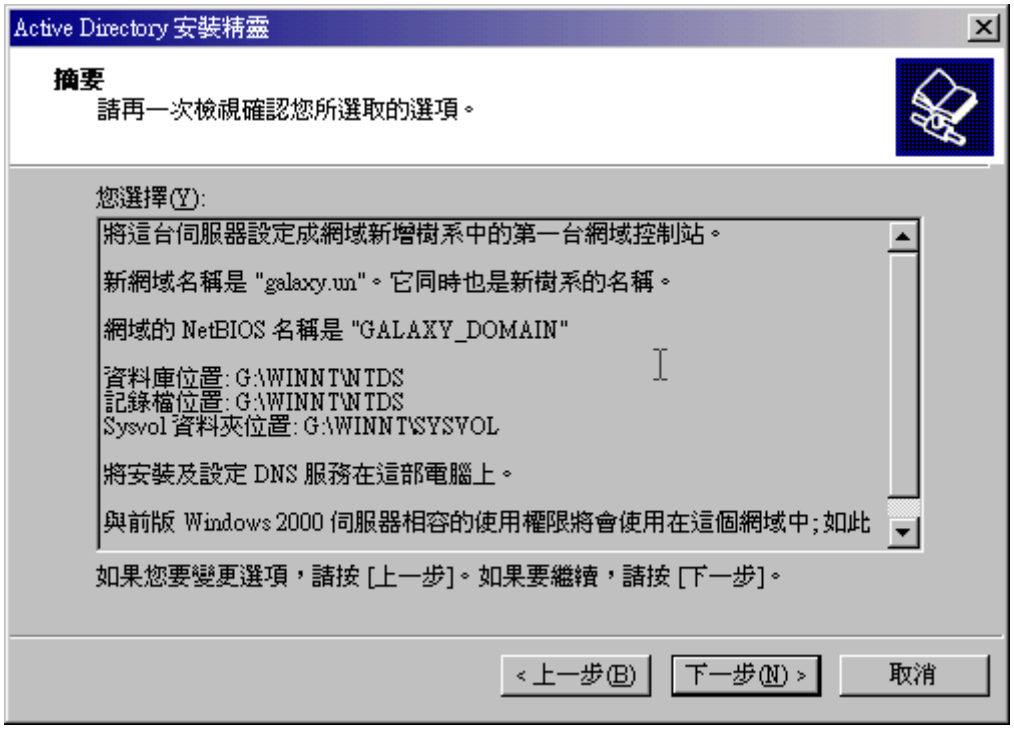


步驟十：設定以目錄還原模式啟動電腦時，系統管理員的密碼。



步驟十一：顯示前幾個步驟所作的選擇之摘要。

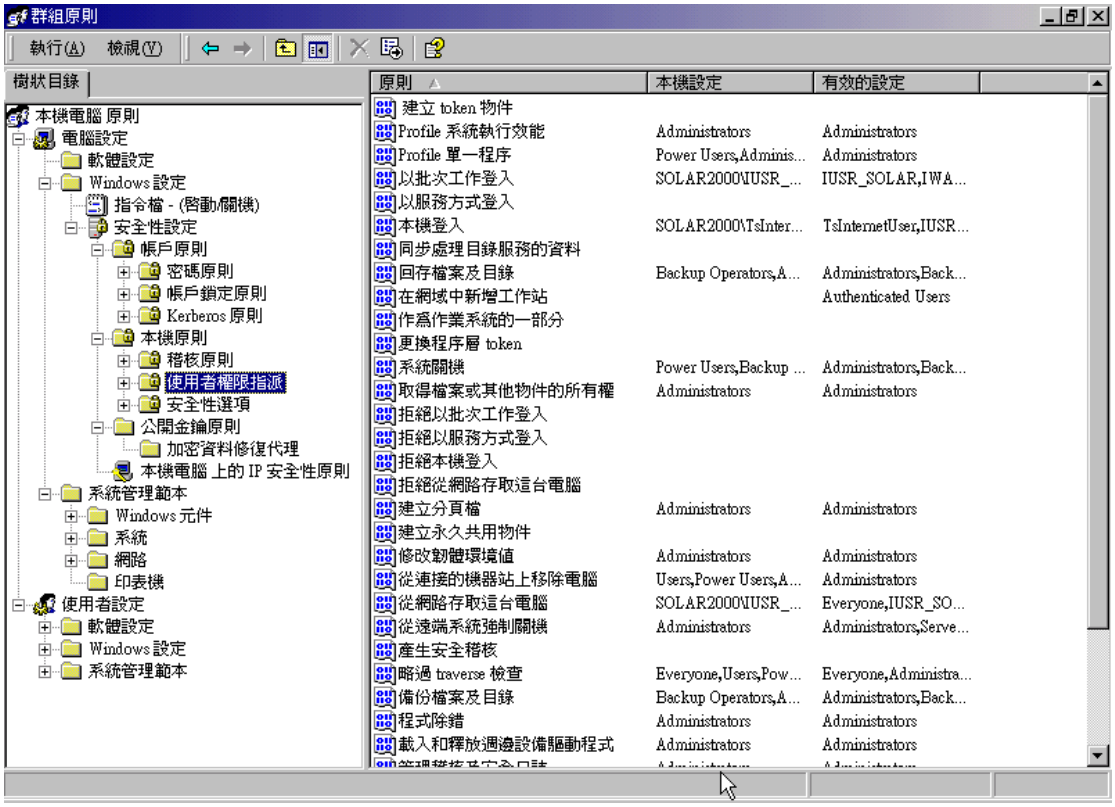
點選「下一步」後，即可完成 Active Directory 的安裝。



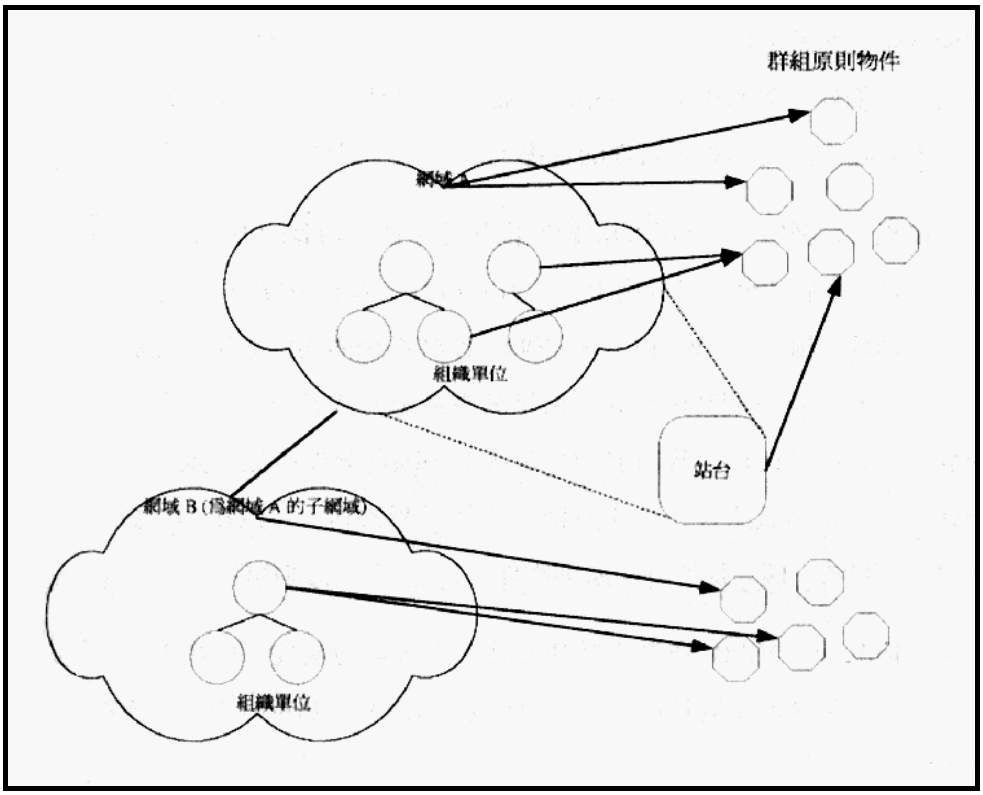
第6章 群組原則

6.1 簡介

群組原則 (Group Policy)是 Windows 2000 作業系統中，建立管理服務架構很重要的一種技術，系統中大半內建的集中式管理服務都是透過群組原則的方式來建立。其主要的目的就是將一些重要的管理原則設定在使用者群組或是電腦群組上，讓這些群組內的成員都套用相同的管理規則。



在 Windows 2000 作業系統中，群組原則是由群組原則物件 (Group Policy Object，GPO)所構成的，換句話說，群組原則的主要管理規則內容是紀錄在群組原則物件中。在 Active Directory 網域上的每個容器物件 (Container)都可以對應到許許多多不同的群組原則物件，管理人員可以在 Active Directory 網域上的網域 (Domain)、組織單位 (Organizational Unit，OU)或站台 (Site)上定義原則，而這些原則的管理規則就儲存在其對映的群組原則物件中。



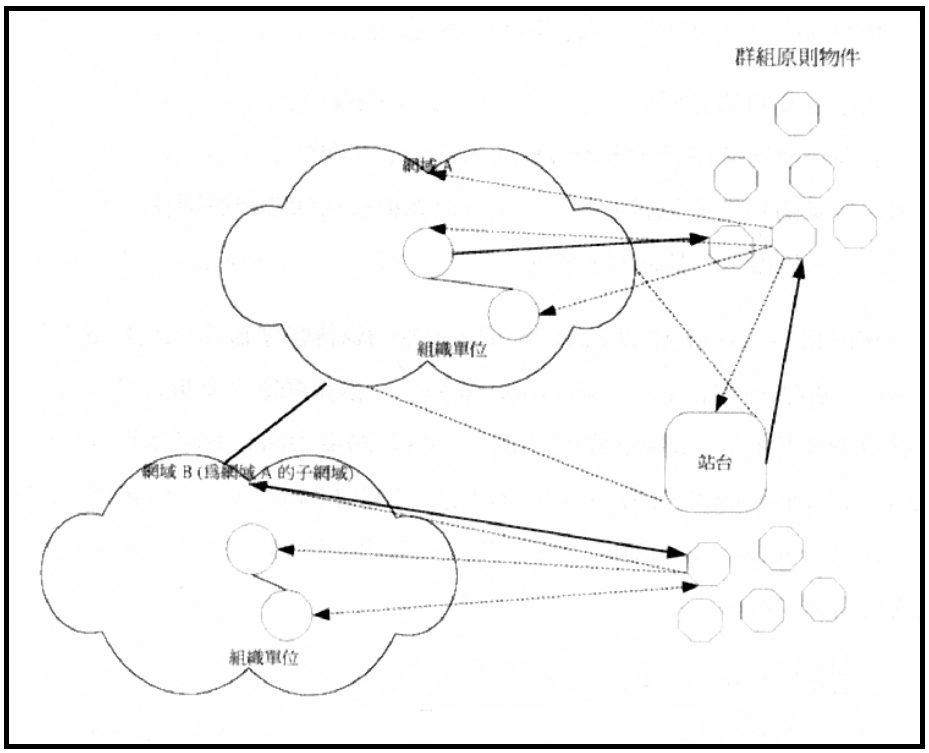
[Windows 2000 Server 網域建制與安全維護]

在每一個群組原則物件內都紀錄了一種群組原則的管理規則，每一個 Active Directory 的容器物件，可以對應到一個或多個的群組原則物件；而一個群組原則物件也可以被多個 Active Directory 的容器物件所對應。雖然 Active Directory 的網域樹系有信任關係的存在，但是群組原則物件不會透過此信任關係繼承到子網域裡。

6.2 群組原則的繼承性

群組原則的繼承性是由 Active Directory 目錄系統容區的集合與子集合的關係所建立起來的，而不是由 Active Directory 網域樹系的信任關係(網域樹系或網藝術上的上層網域與子網域)所建立。

以群組員則可以套用的 Active Directory 容區空間來說，Active Directory 目錄系統內站台的容區範圍最大，其次是 Active Directory 網域，然後才是 Active Directory 網域內的組織單位容區空間。一個 Active Directory 的站台可以包括一個或數個的 Active Directory 網域、一個網域可能又包含了一個或多個的 Active Directory 組織單位容區空間。



[Windows 2000 Server 網域建制與安全維護]

剛剛所提到的群組原則的繼承性就是由這三種 Active Directory 容區空間彼此的的集合關係所構成。上層容區空間一旦以某一個群組原則物件套用後，所有下層的子容區空間預設皆會繼承到該群組原則，除非子容區空間有設定相同的群組原則。例如，有一個群組原則物件的套用範圍是一 Active Directory 的站台，在這個 Active Directory 站台範圍內的網域或者是組織單位容區空間沒有設定相關的群組原則物件

第7章 網路管理一

身為一套作業系統，Windows 2000 Server 隨著 Internet 的趨勢，改進其在網路和 Internet 上的表現，協助企業組織建立更安全、更有效率的網路。除了支援標準的 TCP/IP 協定外，VPNs 讓企業內部網路的建置成本降低並保有相當的安全性；內建路由的功能讓伺服器能輕易的扮演路由器的角色，管理起來不再困難；QoS 標準讓網路作業更加順暢、可靠，加強了即時視訊與聲音的傳輸。

7.1 DNS (Domain Name System)

7.1.1 簡介

DNS 是一個架構在樹狀階層式結構上的一種分散式領域名稱服務，就像電話號碼一樣，每一部在 Internet 上的電腦都有一個唯一的位址，我們稱它作 IP address（IP 位址)。隨著網路的蓬勃發展，愈來愈多的數字化位址讓我們無法記住每一部電腦的 IP address，所以我們改用較熟悉的且有意義的語言來記，就是所謂的完整領域名稱 (FQDN)，一個完整領域名稱 (Fully Qualified Domain Name ,FQDN)是由主機名稱與領域名稱所組成。

每當我們要以完整領域名稱連結網路上的主機時，透過 DNS 將完整領域名稱解析成 IP address，我們就可以順利的悠遊 Internet 了。

7.1.2 原理

- **根領域 (Root domain)**

根領域是 DNS 樹狀階層結構中的最高層，以句點 ”.” 表示，紀錄著所有頂層領域 (Top-Level Domain，TLD)DNS server 的資料，透過這些 DNS Server，我們就可以從 Root DNS Server 一層一層的往下找，直到找到實際負責該區域的 DNS Server。

- **頂層領域 (Top-Level domain)**

頂層領域是根據組織類別或地理區域來劃分，下列為頂層領域的的例子：

頂層領域名稱	說明
com	民間企業
edu	教育學術單位

org	非商業組織
tw	台灣的國家領域名稱
cn	中國大陸的國家領域名稱
biz	商業或民間企業　New
museum	博物館　New

- **第二層領域** (Second-Level domains)

第二層領域包含了兩部份：頂層領域名稱和獨立的下一層領域名稱，可供個人或團體登錄。

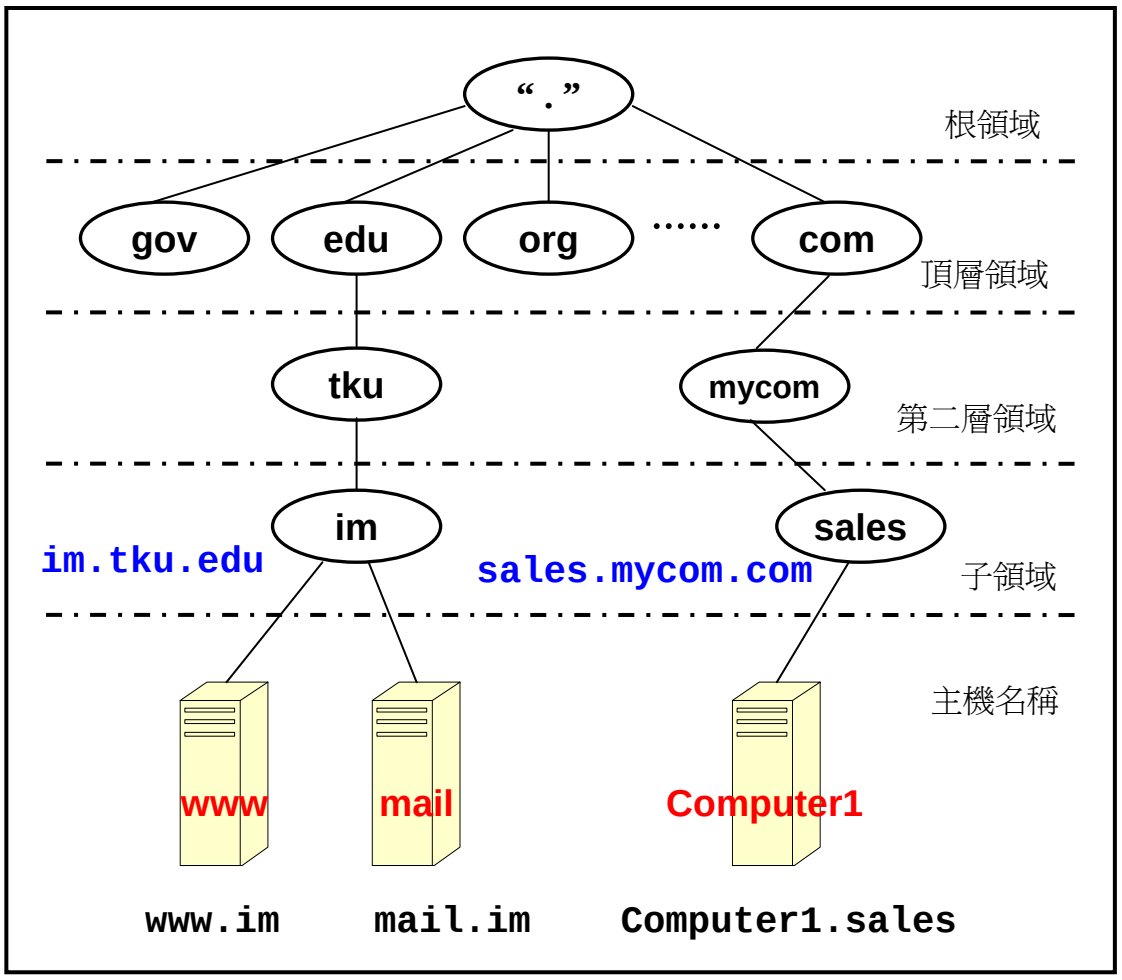
領域名稱	說明
ibm.com	國際商業機器
tku.edu	淡江大學
w3.org	World Wide Web 組織
president.gov.tw	台灣的總統府

- **子領域** (Subdomain)

企業或團體依內部的部門、群組或地區分別建立其子領域，以利管理，如：sales.mycom.com。

- **主機名稱** (Host Names)

主機名稱就是在電腦在網路上的名稱。



我們以下圖為例來說明 DNS 服務的流程：

步驟一：Client 端向其 DNS 伺服器查詢 www.im.tku.edu.tw 的 IP 位址。

步驟二：Client 端所屬的 DNS 伺服器若無法在其資料庫中找到跟 www.im.tku.edu.tw 相對應的 IP 位址，Client 則轉而詢問 Root DNS 伺服器。

步驟三：Root DNS 伺服器根據登記的資料，請 Client 端詢問管理 tw 這個區域的 DNS Server。

步驟四：tw 伺服器再根據登記的資料，查出 www.im.tku.edu.tw 這個名稱登記在負責 edu 區域的 DNS 伺服器，所以 Client 端詢問 edu 的 DNS 伺服器。

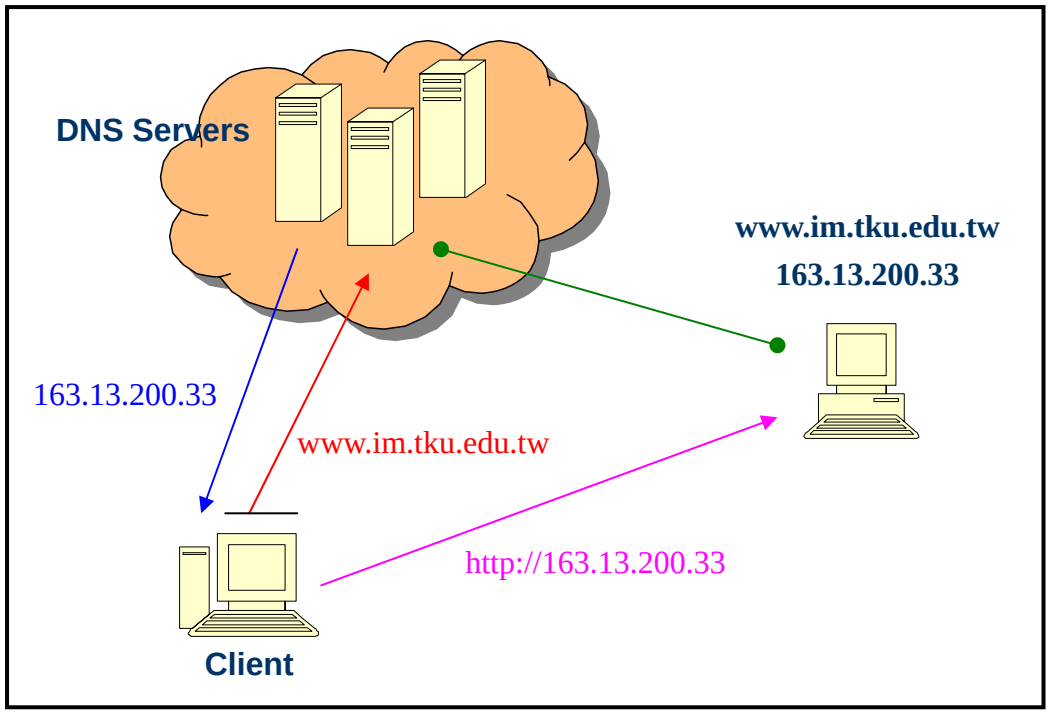
步驟五：相同的，Client 再轉而詢問 tku 的 DNS 伺服器。

步驟六：在 tku 的 DNS 伺服器上查出相對應的 IP 位址。

步驟七：將 IP 位址傳回給 Client 端。

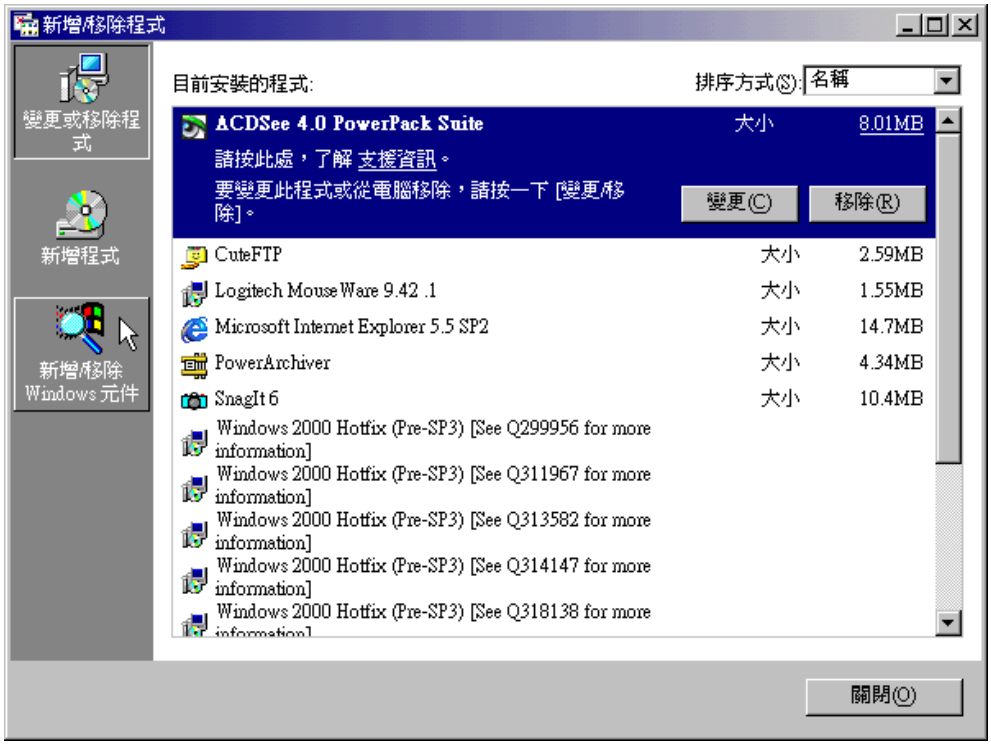
步驟八：Client 端以對 163.13.200.33 這個 IP 位址發出需求。

這個過程看來繁複，但實際上所花的時間非常的短。

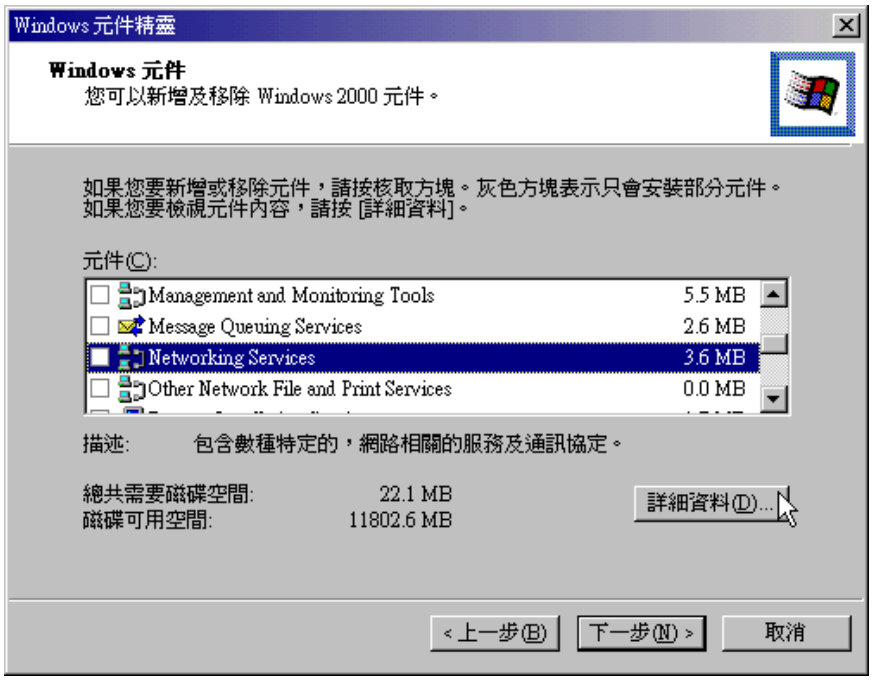


7.1.3 安裝 DNS 伺服器

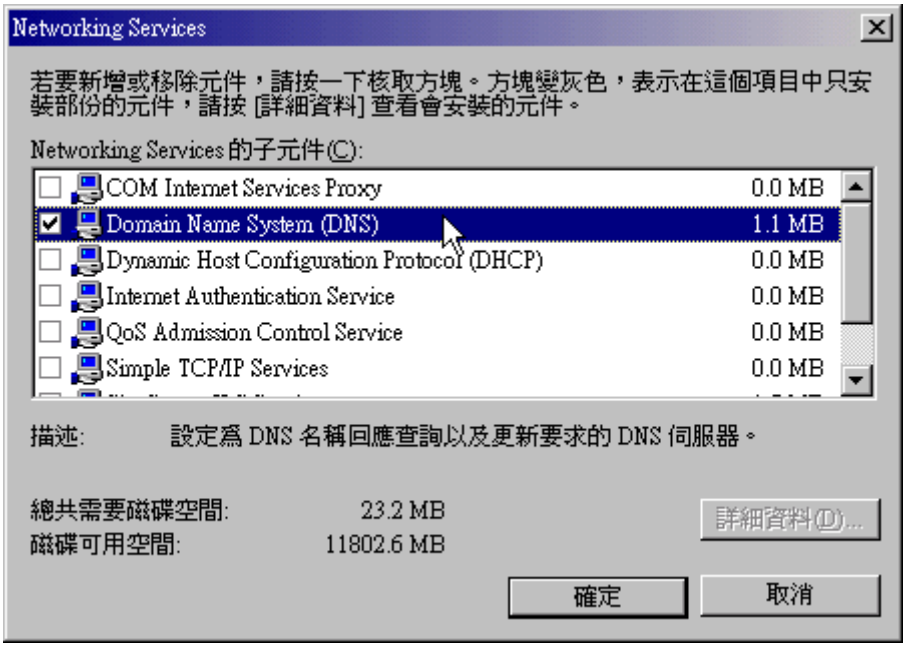
步驟一：首先由「控制台」選擇「新增/移除程式」，在新增/移除程式對話窗左邊點選「新增/移除 Windows 元件」。



步驟二：在 Windows 元件精靈對話窗中，選擇「Networking Services」，然後再點選「詳細資料」按鈕。



步驟三：勾選「Domain Name System (DNS)」，再點選「確定」。

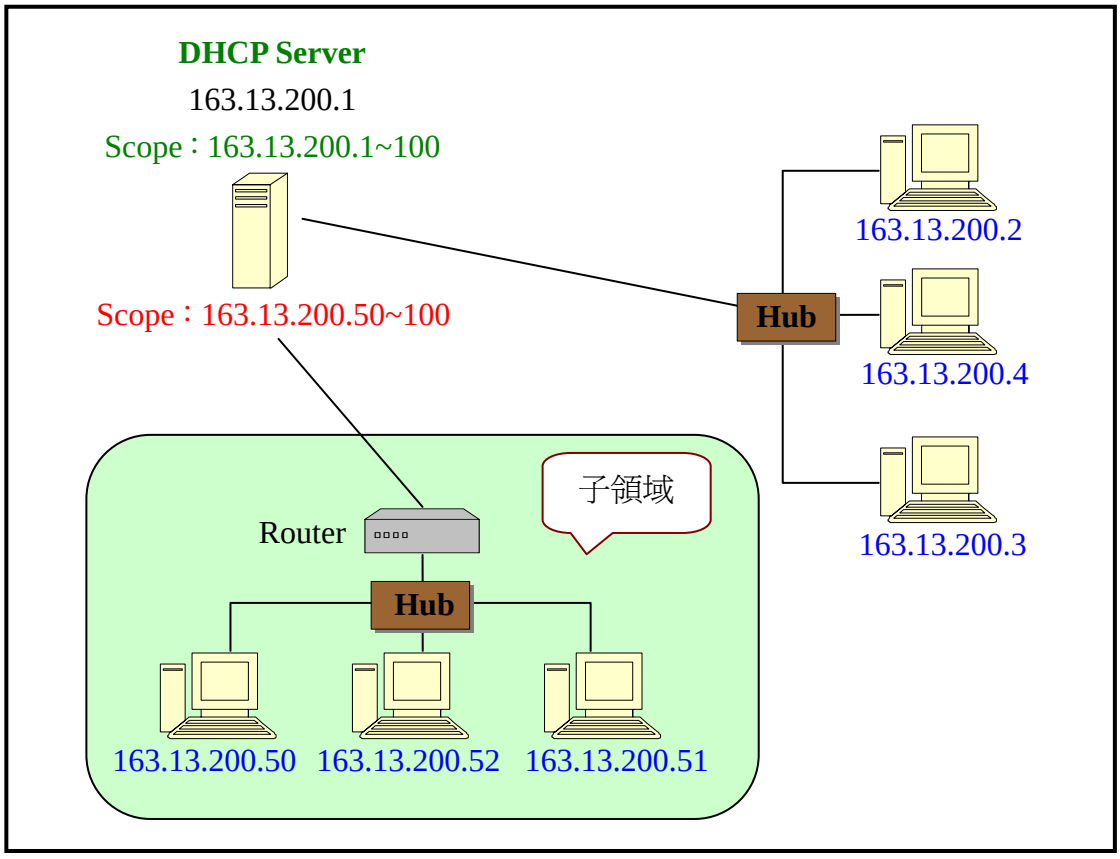


步驟四：回到 Windows 元件精靈對話窗中，點選「下一步」，系統便會開始進行安裝 DNS 的動作。

7.2 DHCP (Dynamic Host Confirguation Protocol)

7.2.1 簡介

在一個 TCP/IP 的網路中，每一台電腦都需要一個 IP 位址，為了簡化管理者的工作，制定了 DHCP 這個開放性的標準協定。管理者透過一台 DHCP 伺服器，給定 IP 範圍、參數及規則，即可輕鬆的管理整個 TCP/IP 網路的 IP 位址，相較於一個一個的輸入，不僅有較高的效率，也減少了因為輸入錯誤而所引起的問題。



用戶端開機後，會透過廣播 (Broadcast)向 DHCP Server 要求租用 (Lease)一個 IP 位址，DHCP Server 收到後會傳回一個尚未租用的 IP 位址以及其他相關資訊 (如：子網路遮罩、DNS 伺服器位址、預設閘道位址等)給用戶端。由於是透過廣播，在子網路中就必須透過 DHCP 中繼代理者 (DHCP Relay Agent)，DHCP Server 才有辦法接收到用戶端所傳送出來的廣播封包，而 DHCP 中繼代理者的角色一般都是由支援(Bootstrap Protocol，BOOTP)的 Router 來扮演。

7.2.2 安裝 DHCP Server

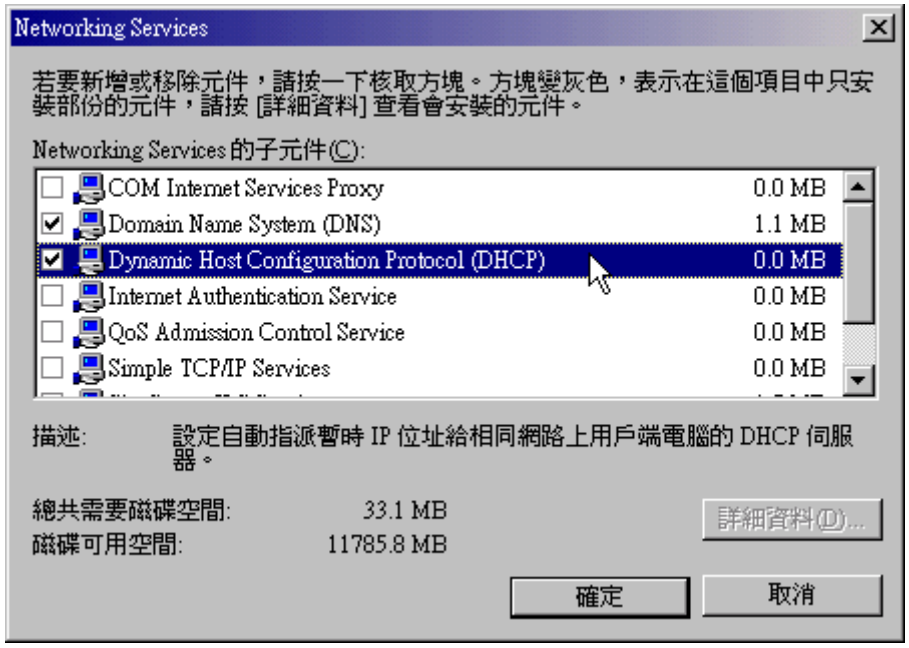
與 DNS Server 安裝類似。

步驟一：由「控制台」選擇「新增/移除程式」，在新增/移除程式對話窗左邊點選「新增/移除 Windows 元件」。

步驟二：在 Windows 元件精靈對話窗中，選擇「Networking Services」，然後再點選「詳細資料」按鈕。

步驟三：勾選「Dynamic Host Configuration Protocol (DHCP)」，再點選「確定」。

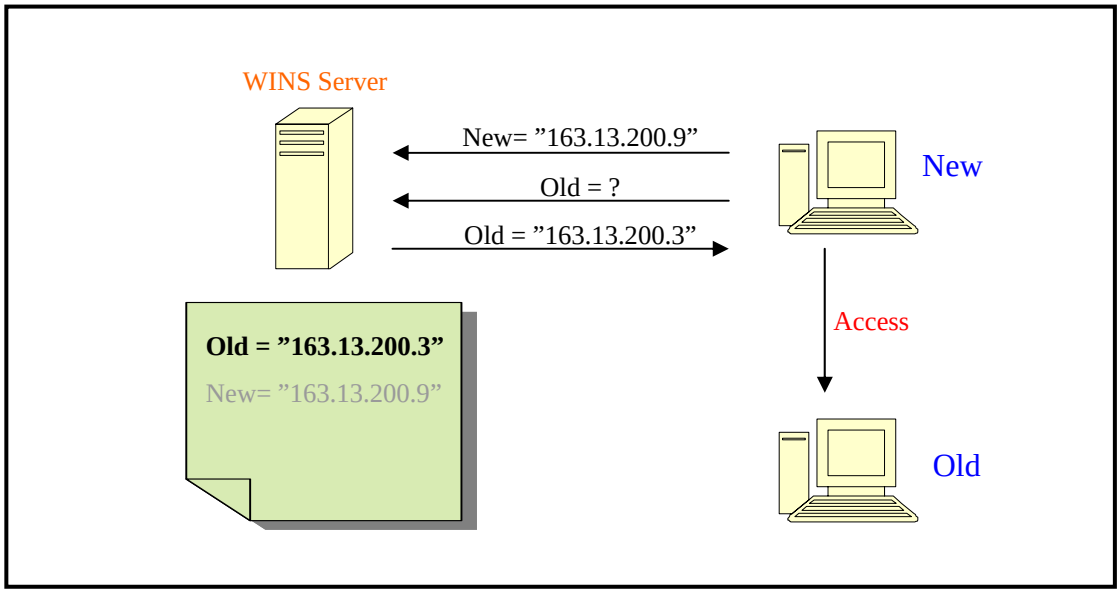
步驟四：回到 Windows 元件精靈對話窗中，點選「下一步」，系統便會開始進行安裝 DHCP 的動作。



7.3 WINS (Windows Internet Name Service)

7.3.1 簡介

WINS Server 維護著一個動態的資料庫，該資料庫儲存著網路上所有電腦的 NetBIOS 名稱，用戶端藉由其 IP 位址，向 WINS Server 註冊。當在以 NetBIOS 名稱為基礎的網路中要進行網路連結時，透過 WINS Server 將 NetBIOS 名稱解析為 IP 位址，可以減少網路廣播流量，同時也跨越了子網路間連結的限制。



用戶端開機時，先向 WINS Server 註冊，由於 WINS Server 資料庫中的每筆紀錄都有一個保存期限 (Time To Live，TTL)，所以用戶端會定時更新在 WINS Server 中的紀錄。

當要進行網路連結時，”New” 端先以 NetBIOS 名稱 ”Old” 向 WINS Server 查詢 ”Old” 的 IP 位址，再透過 WINS Server 回傳的 IP 位址連結 “Old” 端。

7.3.2 安裝 WINS Server

步驟一：由「控制台」選擇「新增/移除程式」，在新增/移除程式對話窗左邊點選「新增/移除 Windows 元件」。

步驟二：在 Windows 元件精靈對話窗中，選擇「Networking Services」，然後再點選「詳細資料」按鈕。

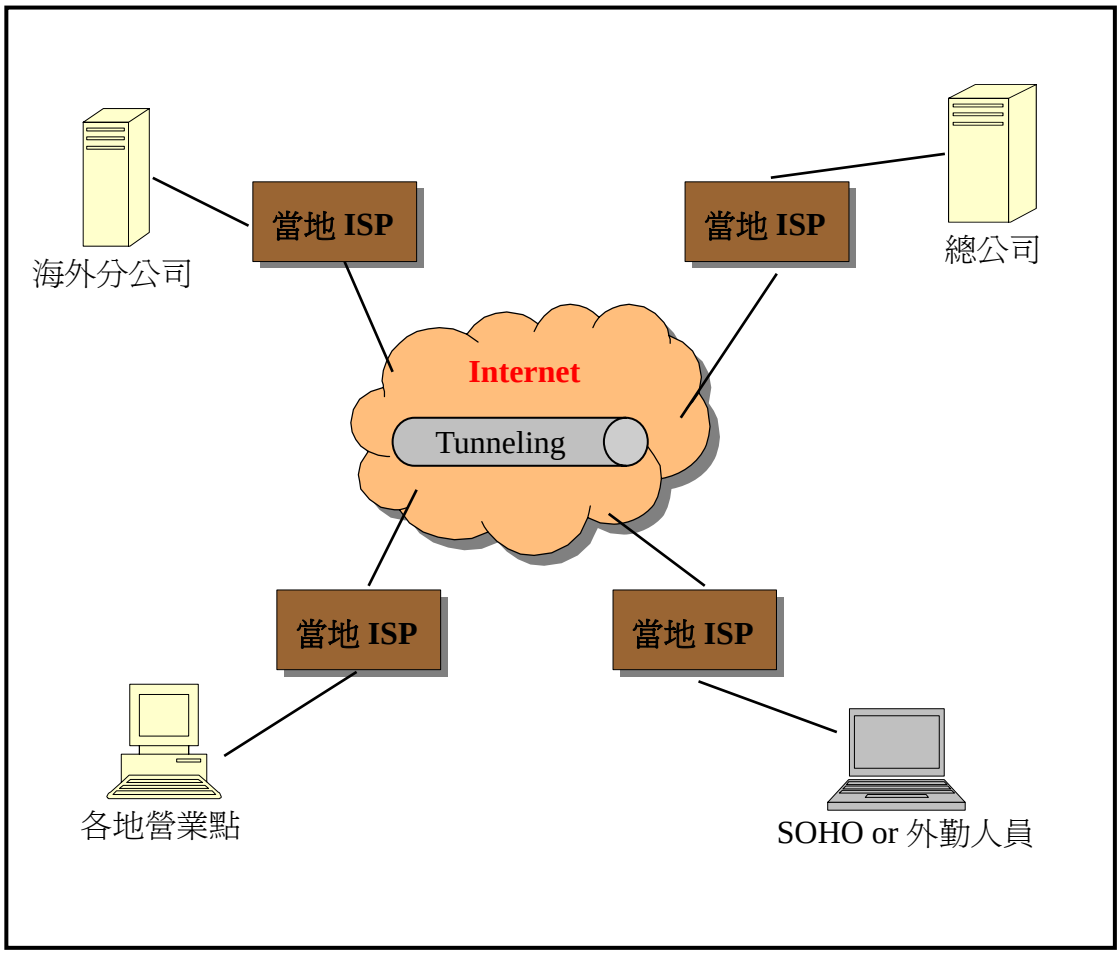
步驟三：勾選「Windows Internet Name System (WINS)」，再點選「確定」。

步驟四：回到 Windows 元件精靈對話窗中，點選「下一步」，系統便會開始進行安裝 WINS 的動作。

7.4 VPN (Virtual Private Network)

7.4.1 簡介

VPN 虛擬私人網路是近年來相當熱門的技術，利用 Internet IP 的技術，以公眾網路代替專線，在 Internet 上建立加密通道 (Tunneling)來架構企業網路，降低了硬體及連線的成本。



7.4.2 通道 (Tunneling)技術

通道技術指的是利用一種網路協定來傳輸另一種網路協定，其傳遞的資料可以是不同通訊協定的封包，通道通訊協定會將該種類封包再行封裝以便傳輸，而接收端收到封包後再去除掉傳送前再行封裝時加上的標頭，成為原來種類的通訊協定封包，轉送到目的地。目前較被廣泛使用的通道技術有三個：點對點通道協

定（Point-to-Point Tunneling Protocol，PPTP）、第二層通道通訊協定（Layer Tunneling Protocol，L2TP）、IP 安全（Internet Protocol Security，IPSec）。

■ **點對點通道協定（Point-to-Point Tunneling Protocol，PPTP）**

PPTP 是微軟和其他公司所發展的多重協定通道技術，它以能讓兩台電腦用撥接的方式建立連線的 PPP（Point to Point Protocol)通訊協定為基礎，因此 PPTP 借用了 PPP 的認證及交談機制。

PPTP 可以建立隧道或將 IP、IPX 或 NetBEUI 協定封裝在 PPP 封包內，所以允許用戶端運行依賴特定網路協定的應用程式。亦即，即使用戶端使用了不同的通訊協定依舊可以連接到內部網路中以 IPX/SPX 為通訊協定的 NetWare 伺服器。

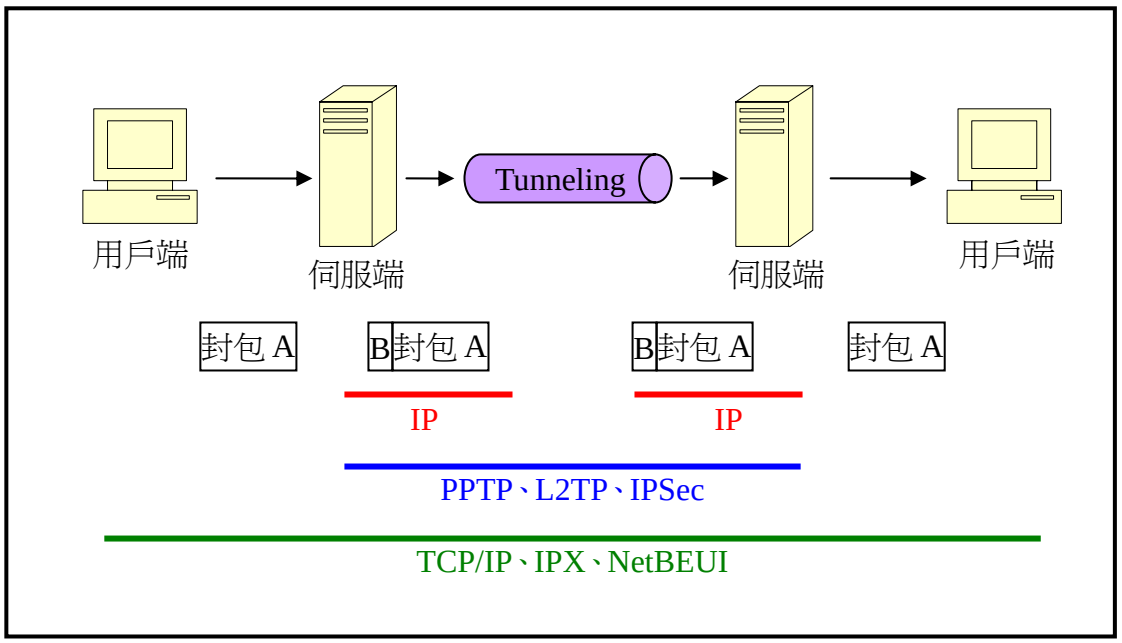
■ **第二層通道通訊協定（Layer TunnelingProtocol，L2TP）**

L2TP 是由 Microsoft、Cisco、Ascend、IBM 和 3Com 所共同制定的通訊協定，演化自 PPTP 及 Cisco 的 Layer-2 Forwarding（L2F），與 PPTP 不同的是，L2TP 使用新的 IP 安全（IPSec）機制來進行身份驗證和資料加密。

L2TP 支援一項特別的技術 MPPP（Multilink Point to Point Protocol，多重連結點對點通訊協定），可讓系統同時透過兩個不同的 ISP，傳送資料到也使用 L2TP MPPP 的遠端系統，該系統會再自動的重組回原來的資料，可惜大部分的 ISP 部不支援。

■ **IP 安全（Internet Protocol Security，IPSec）**

IPSec 屬於第三層通道技術，它不需負責通道的建立與維護，不像 PPTP、L2TP 需要通道的兩個端點必須同意建立通道並協商其各種配置變量。IPSec 僅支援 IP 網路的傳輸，傳送前以安全的方式封裝資料再對 IP 封包加密，然後再次封裝加上 IP 標頭進行傳輸。



7.4.3 運作模式

■ PPTP Tunneling：

1. Client-initiated：經由客戶端撥接至 ISP，再由客戶端啟動 PPTP 的 Session 和欲建立的另一端 PPTP Server 建立通道。

2. ISP-initiated：客戶端和 ISP 的 Access Server 建立 PPP 連線後，經由 ISP 的 Access Server 和目的端 PPTP Server 建立通道，客戶端不需安裝 PPTP Client 完全由 ISP-PPTP-enable。

3. LAN-to-LAN tunneling：在 PPTP Server 兩端建立 PPTP tunneling，如在 Windows 2000 Server 上加裝 Microsoft 的 Routing and Remote Access Service (R&RAS)。

■ L2TP Tunning：

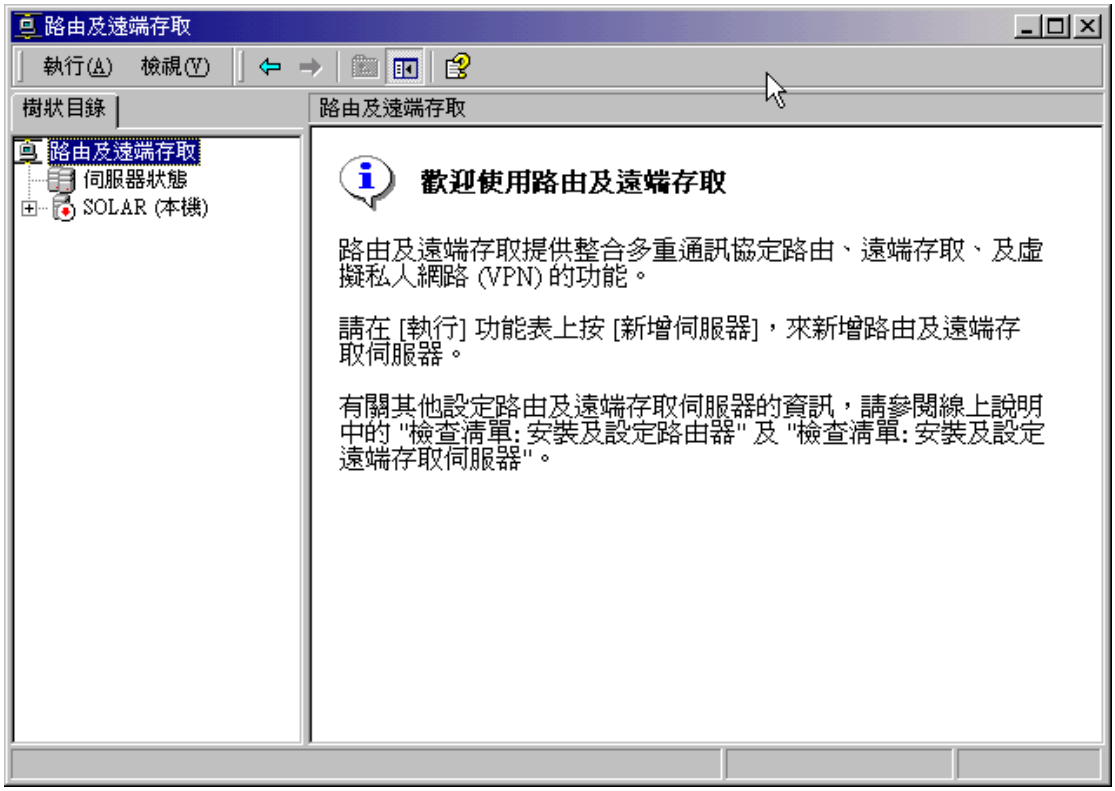
1. 遠端使用者先和 ISP 端建立普通的 PPP 連線。

2. 遠端使用者透過 ISP 和遠端 L2TP Server 進行協調。

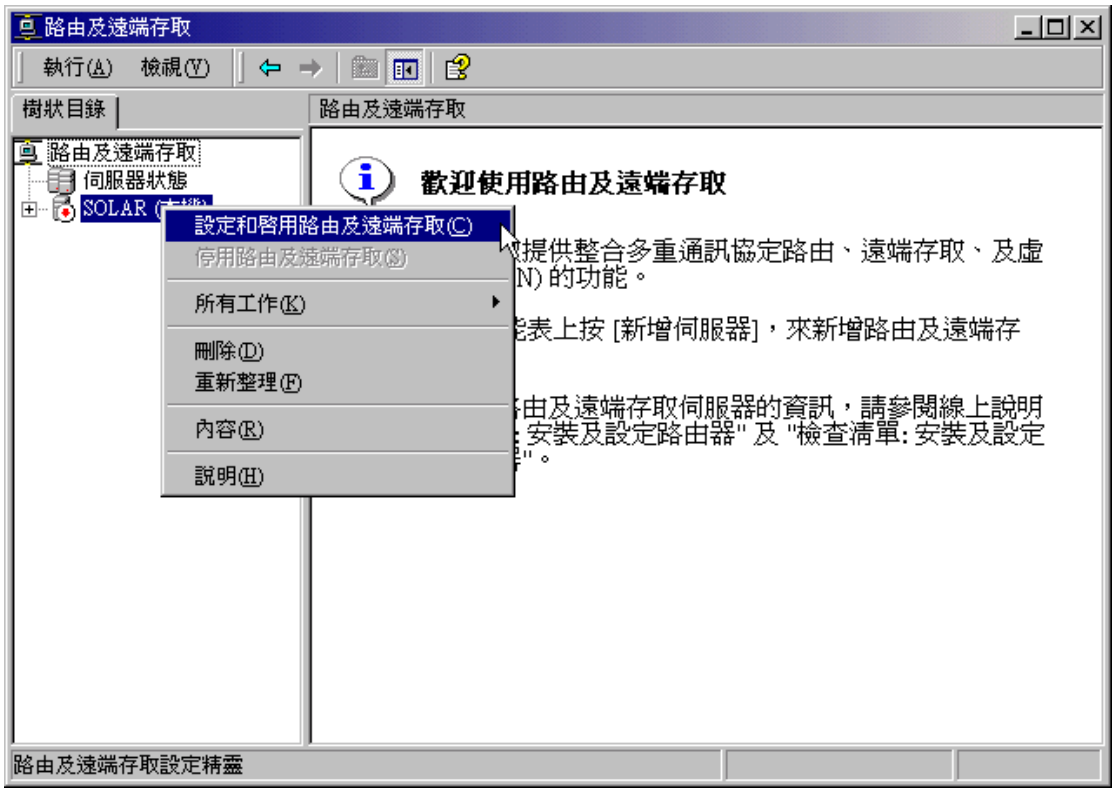
3. 遠端使用者透過 ISP 和遠端 L2TP Server 建立通道。

7.4.4 安裝 VPN

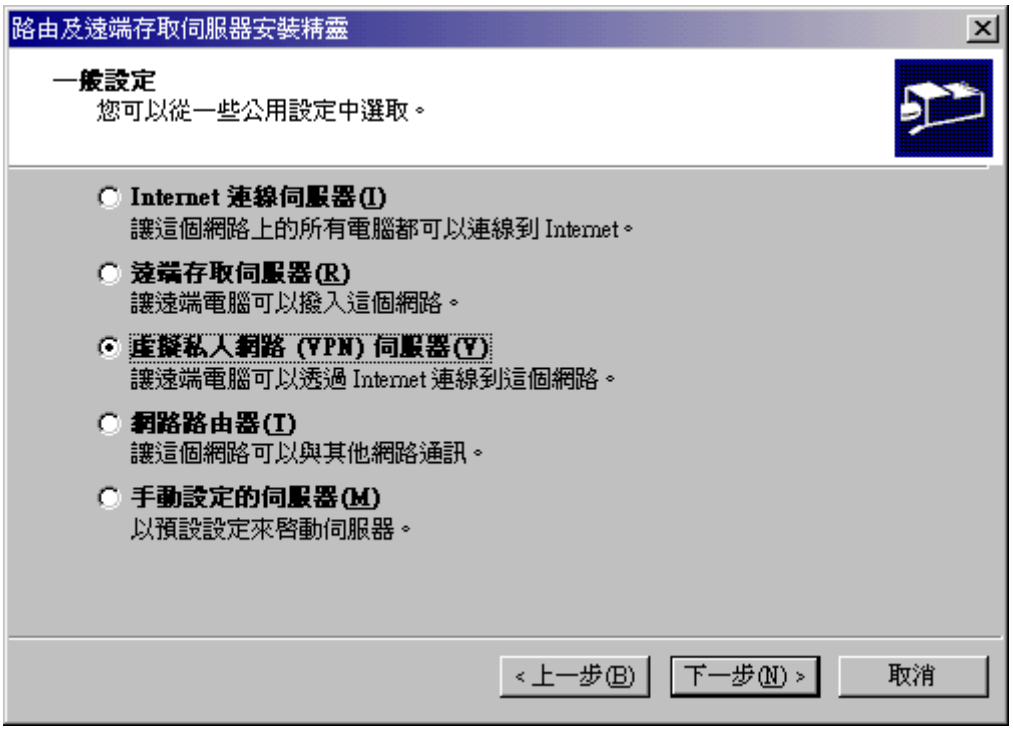
步驟一：選擇「系統管理工具」→「路由及遠端存取」後可以看見，本機的路由及遠端存取服務尚未啟動。



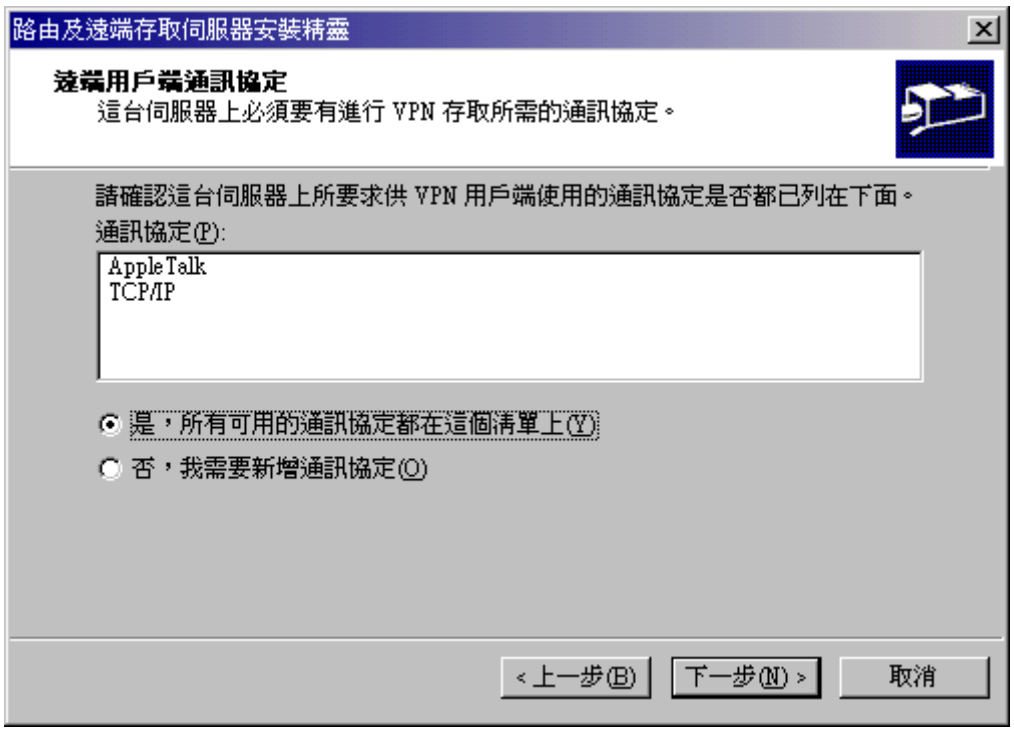
步驟二：在本機伺服器上按滑鼠右鍵，點選「設定和啟用路由及遠端存取」，接著點選「下一步」略過歡迎畫面。



步驟三：選擇要安裝的虛擬私人網路(VPN)伺服器。

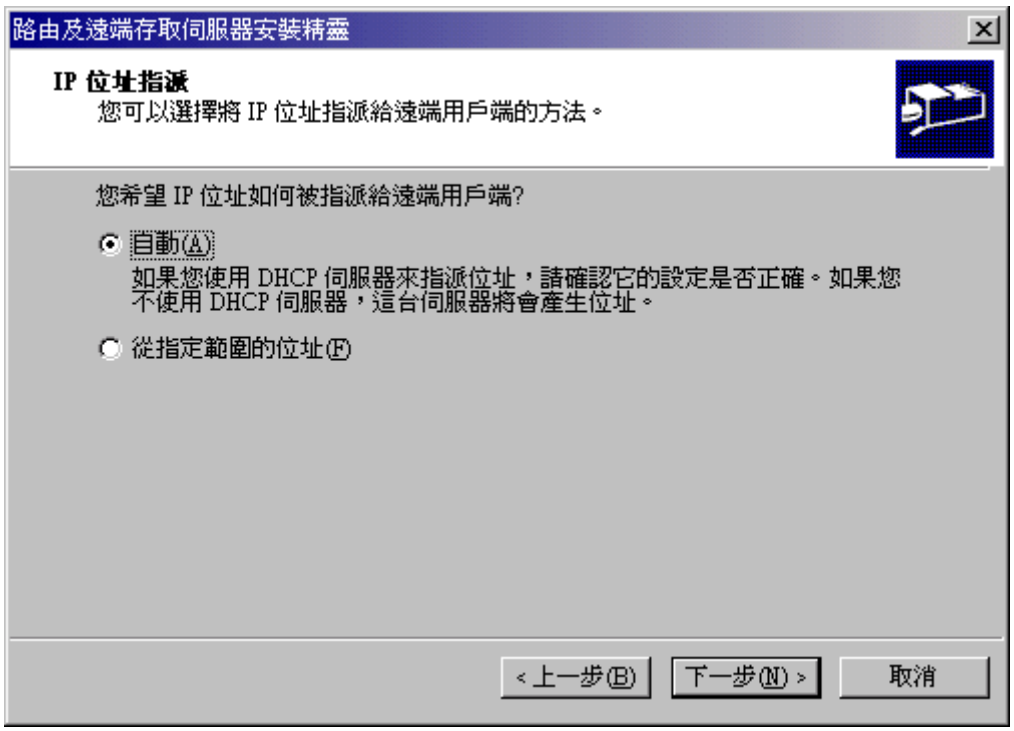


步驟四：確認供用戶端使用之通訊協定。

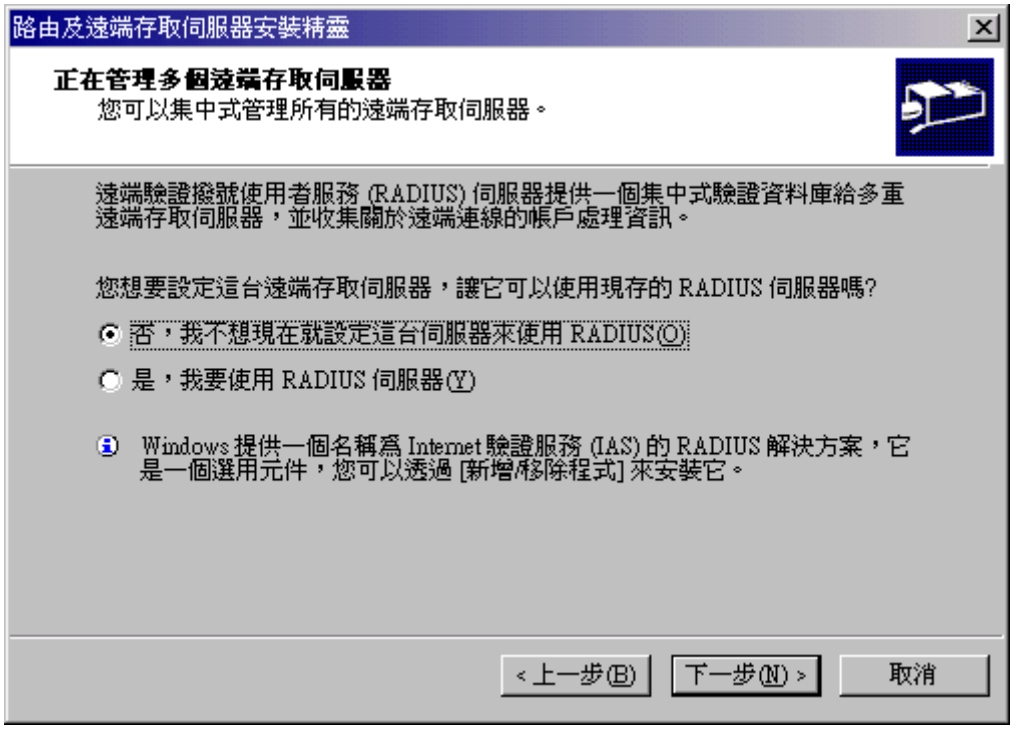


步驟五：選取 Internet 連線。

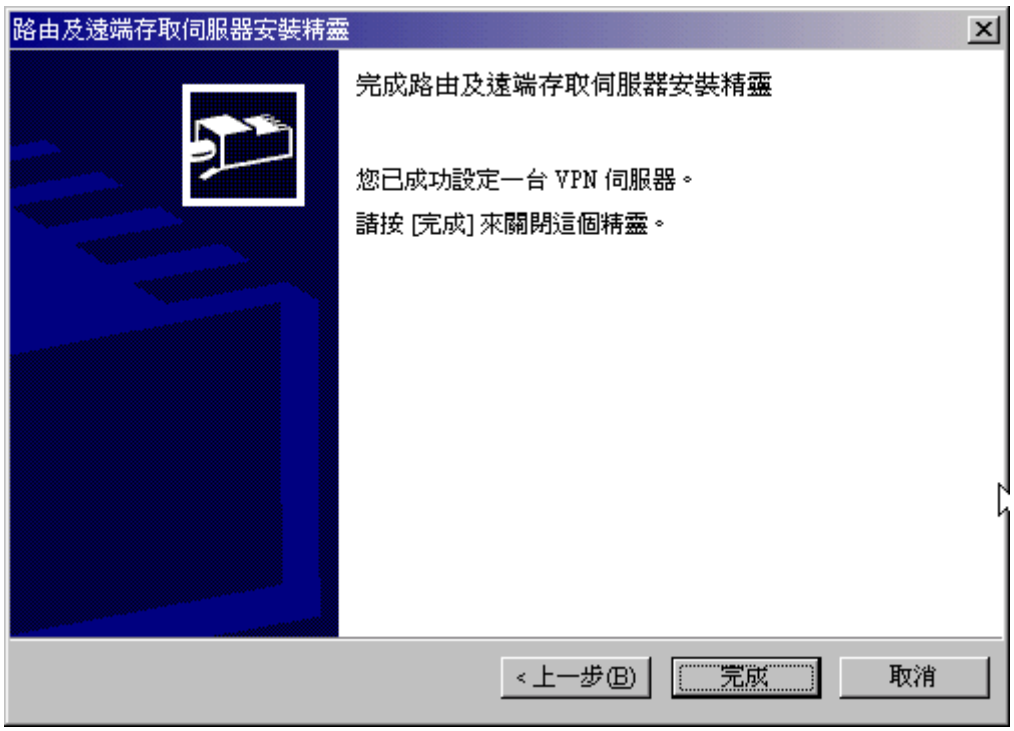
步驟六：選取用戶端 IP 的指派方式。



步驟七：是否使用 RADIUS 伺服器。



步驟八：點選「完成」，即開始起始 VPN 伺服器。



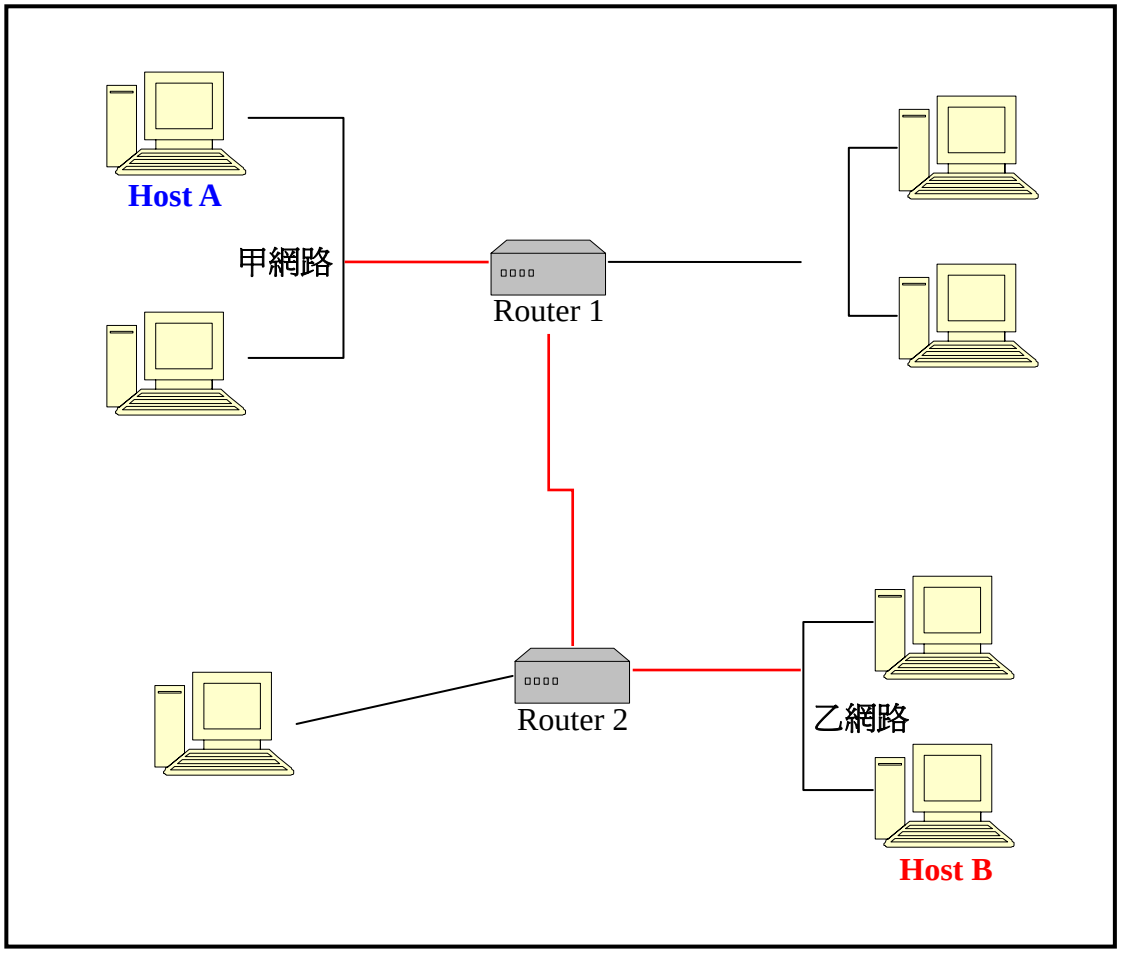
當以路由及遠端存取服務啟動虛擬私人網路伺服器後，由於安裝精靈在連接 Internet 的網路介面上設定了篩選器，只允許 PPTP 和 L2TP 的封包通過，因此一般的使用會無法連線到 Internet，需要透過修改篩選器的設定，Internet 的連線才會恢復正常。

7.5 IP 路由

7.5.1 IP 路由 (IP Routing)

IP 路由指的是將 IP 封包轉送到目的地的機制。為了降低大量廣播封包對頻寬的影響，整個網路上存在著許多的子網路，透過子網路的設計，廣播封包只能在其所屬的子網路中傳遞。若要跨越不同的子網路，則必須透過路由的技術，將封包由甲子網路轉送至乙子網路的目的地。一般網路間的連接，我們可以直接透過硬體的路由器 (Router)，不過在這我們將介紹的是以軟體透過伺服器的機制，來達到 IP 路由的相同效果。

7.5.2 運作方式



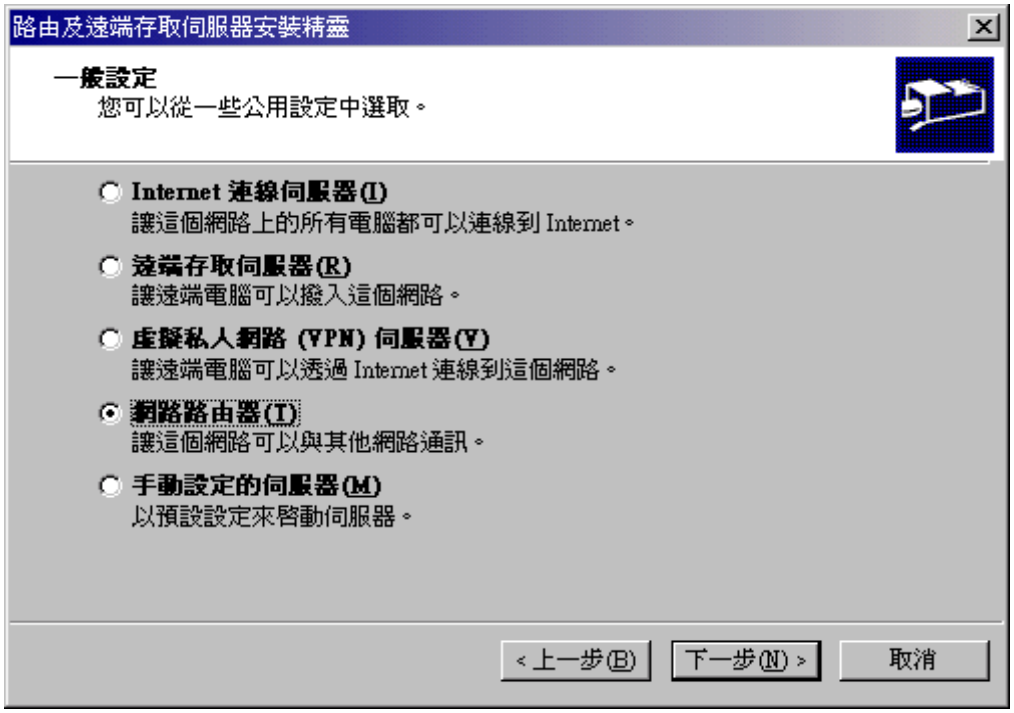
Host A 要將封包傳送到 Host B 時，首先在甲網路中，經由 Router 1 中的路由表 (Routing Table)判斷要將封包轉送到 Router 2，Router 2 再查詢自己的路由表，將封包轉送到乙網路中，Host B 即可收到封包。在整個路由的路徑中，通常會經過許多的路由器轉送，若封包的目的地不在路由器所連接的網路

上，則路由器會再將封包轉送給其他相鄰的路由器，如此循環，直到封包正確的到達目的地。

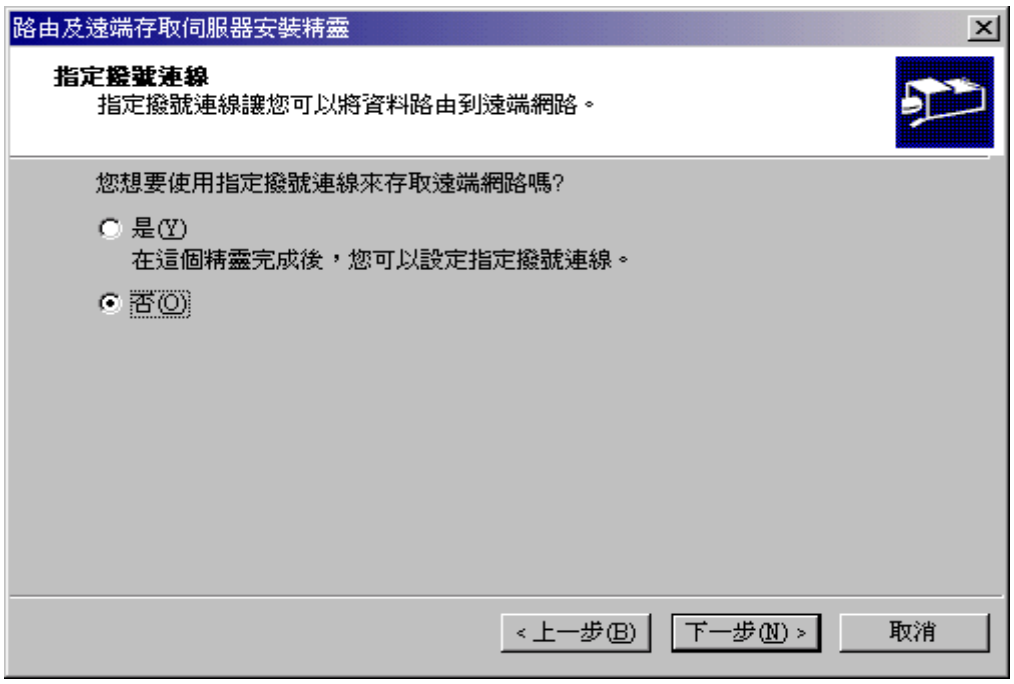
7.5.3 啟動 IP 路由

步驟一：選擇「系統管理工具」→「路由及遠端存取」後可以看見，本機的路由及遠端存取服務尚未啟動。在本機伺服器上按滑鼠右鍵，點選「設定和啟用路由及遠端存取」，接著點選「下一步」略過歡迎畫面。

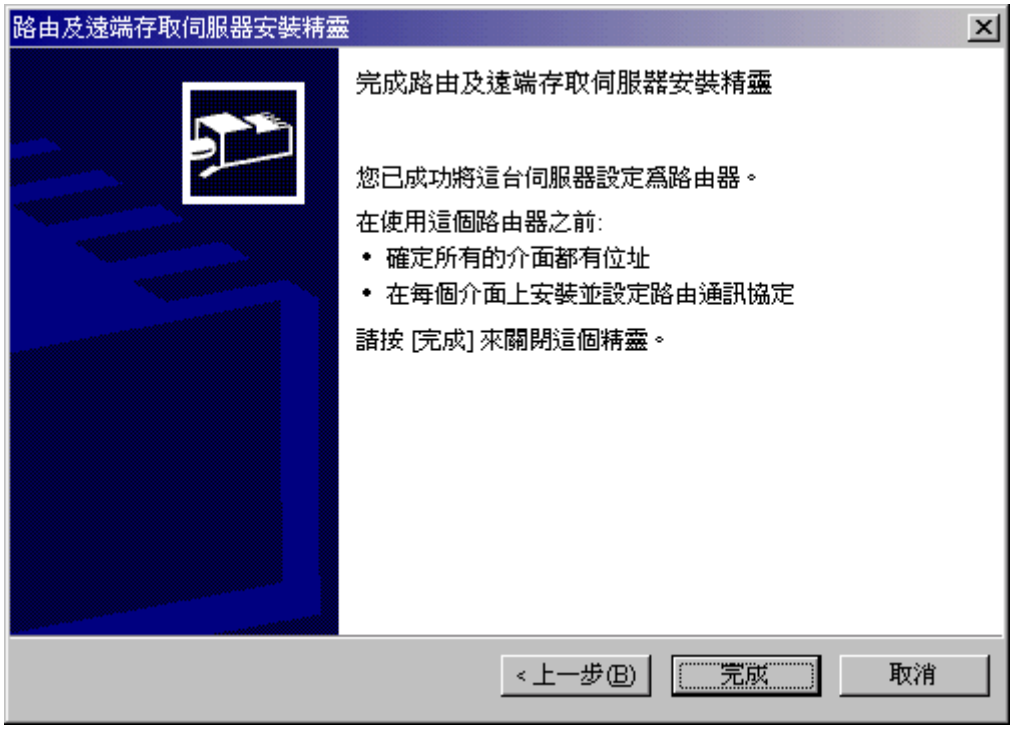
步驟二：選擇要安裝的網路路由器。



步驟三：確認供用戶端使用之通訊協定完後，選擇是否在完成後設定指定撥號連線，因為一般為固接式網路所以選否。



步驟四：點選「完成」，即開始啟始 IP 路由。



第8章 網路管理二

8.1 網路通訊協定與服務

8.1.1 NetBIOS 與 NetBEUI

NetBIOS(Network Basic Input Output System)最早是由 IBM 所發展，是一種讓程式可以在區域網路 (LAN)上使用的應用程式程式介面 (API)，NetBIOS 為所有程式提供一組一致的指令，用於要求需要管理名稱的低等服務、進行工作階段、以及在網路上的節點間傳送資料包。微軟將它用在會議層 (Session Layer)做檔案及列印的分享、名稱解析等等…，但 Client 也要安裝 NetBIOS 才能使用 Server 所提供的服務及分享。

NetBEUI (NetBIOS Extension User Interface)是 IBM 為小型區域網路所開發的通訊協定，在早期 TCP/IP 尚未普及化前，區域網路上的通訊協定除了 IPX/SPX 之外就是 NetBEUI 了，而且在不使用 TCP/IP 的小型網路上，NetBEUI 速度也是很快的。

8.1.2 IPX/SPX

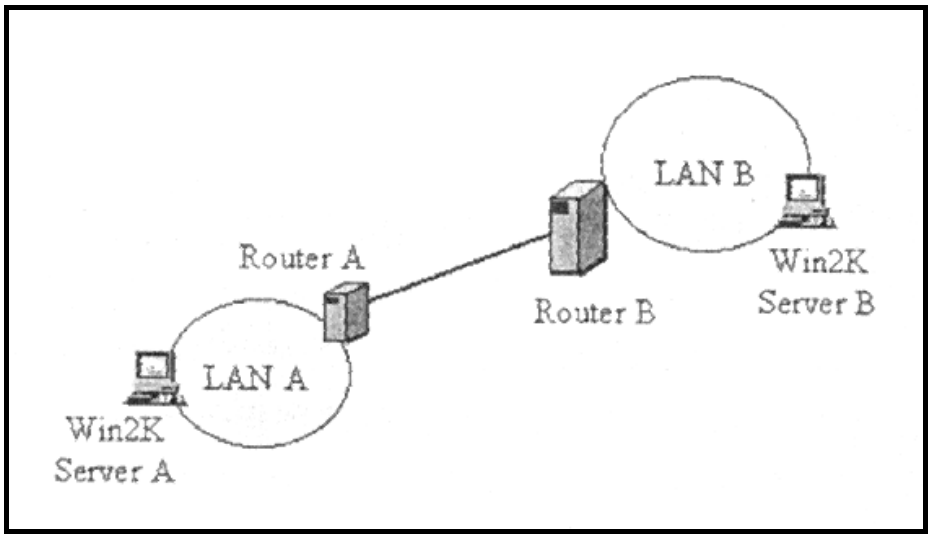
網際網路封包交換/連續封包交換 (Internet Packet Exchange/IPX/SPX)，為 Novell Netware 系統所使用的無連結 (Connectionless)網路通訊協定。該協定主要是定義了網路資料封包傳輸協定，IPX 與 SPX 兩者之間的唯一不同點就是，當你要傳一堆資料封包到另外一台電腦而使用 IPX 通訊協定時，對方電腦收到封包的順序會跟你傳封包的順序一模一樣；但如果是 SPX 通訊協定的話，這點就不能保證了，IPX 在 NETWARE 的角色類似 IP 在 TCP/IP 的作用。在 Microsoft Windows 作業環境中提供了 NWLink 來支援 IPX/SPX，以便 Windows 使用者可與 Novell Netware 伺服器溝通。

8.1.3 QoS (Quality of Service)

Quality of Service 是一項新的網路技術，Windows 2000 開始將此服務納

入作業系統中。它的目的是為了保持某個服務在網路上可以使用的頻寬，也就是說限制某個服務在網路上可以使用的頻寬。例如說，公司聯外的網路頻寬有128Kbps，而這個頻寬卻提供了公司大大小小的連線需求，可以限制公司最重要的電子交易系統保有 64Kbps 的頻寬。如此一來，當該系統需要頻寬時，最少能有 64Kbps 的頻寬保證，才不至於使系統出現問題。

要實行 QoS，最好每個網路設備都要支援 QoS，支援的越徹底，實行的效果也越好，下圖中，在 LAN A 中的某個 Windows 2000 Server A，想要執行 QoS 的功能，以維護在它電腦中某個應用程式服務連線到 LAN B 中的電腦 B 時，能保有適當的網路頻寬。若只在 LAN A 中實行 QoS，那麼在 LAN A 中保有了頻寬，但在 LAN B 中卻沒有。所以對 QoS 來說，要實行的徹底，不僅是端點的電腦要實行 QoS，就連中間的網路設備，如路由器、Switch 等，也都要實行 QoS，這樣才能完整的保有網路頻寬。



[Windows 2000 Server 網路系統與服務]

8.1.4 QoS 許可控制服務

QoS 許可控制服務 (Admission Control Service，ACS)是一個用來管理 QoS 使用情況的元件，用來管理網路區段上網路資源的使用，主要目的是為了確保接二連三建立的應用程式可以順利的進行通訊且 QoS 可以正常的運作。

QoS 許可控制服務簡化了許多網路管理上的問題，有下面幾個優點：

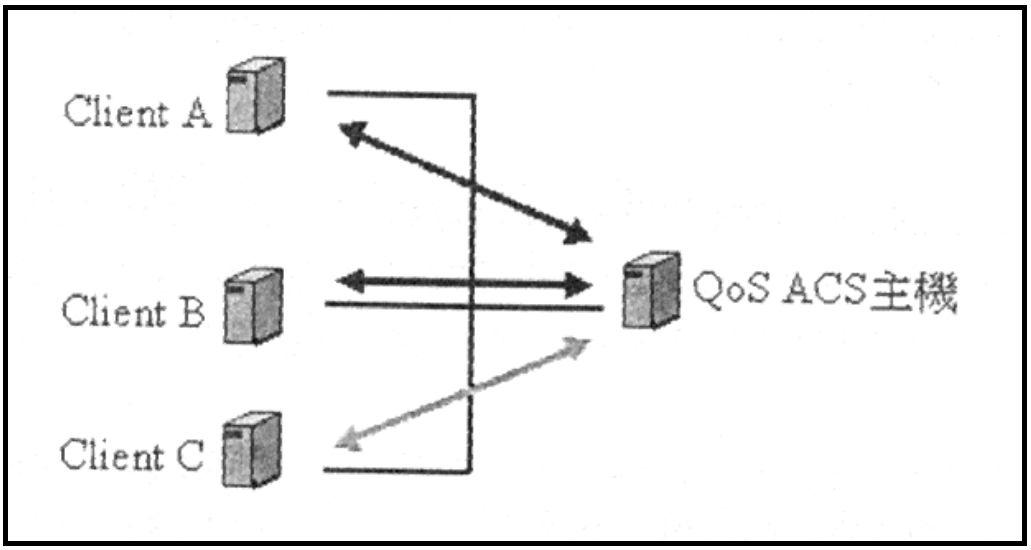
1. 透過 QoS 許可控制服務，針對每個使用者或是每個子網路來集中化子網路的頻寬原則。
2. 對使用者來說無須學習任何設定。

3. 能夠在低優先權與高優先權之間作分割網路資源的能力。
4. 具有低延遲保留的點對點網路服務。
5. 不受網路拓樸限制，可在 LAN、WAN、ATM、Ethernet 或是 Token Ring 的環境中運作。
6. 支援頻寬保留訊息的多點傳送 (Multicasting)功能。

8.1.5 QoS 許可控制服務的運作

為了能在網路中應用多媒體程式，或是執行一些會傳送重要資料的應用程式，網路的頻寬勢必是個關鍵。多媒體應用程式需要大量且固定的頻寬，如此一來，才能使的服務可以順順利力的進行，而不至於有延遲的現象發生。不僅如此，子網路管理服務還必須找出一個處理封包優先權的方式，且不能與原有網路上的交通流有所衝突。

當 QoS 許可控制服務伺服器收到頻寬處理的要求時，它會先檢查哪一個網路服務層級較適合，QoS 許可控制服務會檢查傳送端或是接收端，也有可能兩端都檢查。而要求頻寬處理的使用者，QoS 許可控制服務就會利用 Kerberos 協定來驗證使用者，成功的話，QoS 會從 Active Directory 中取得 QoS 許可控制服務原則，接著 QoS 許可控制服務伺服器會檢視原則，以查看使用者是否有足夠的權限可以做這樣的網路頻寬請求，最後，QoS 許可控制服務會依上述的結果來決定是否讓這個網路頻寬請求運作，若決定通過，則使用者所要求的 QoS 機制便會立即啟用。



[Windows 2000 Server 網路系統與服務]

上圖示一個基本的 QoS 許可控制服務模型，有 A、B、C 三個用戶端，而這個子網路只有 20MB 的網路頻寬，用戶端 A 為了要執行某個應用程式，向 QoS 許可控制服務伺服器要求了 10MB 的頻寬，經過驗證合格，因此 QoS 許可控制服務伺服器便核可了這一個需求。接著，用戶端 B 也為了要執行某個多媒體應用程式而向 QoS 許可控制服務伺服器要求了 10MB 的頻寬，驗證一樣合格，且剩餘頻寬也足夠，所以 QoS 許可控制服務伺服器也核可了用戶端 B 的要求。當用戶端 C 也要跟 QoS 許可控制服務伺服器要求 10MB 的頻寬，但 QoS 許可控制服務伺服器發現已經沒有剩餘的頻寬可供指派，便拒絕用戶端 C 的請求。此時用戶端 C 只有兩種選擇，一種就是在沒有 QoS 的機制下，用一般的傳輸方式傳送封包，另一種方式就是不傳送封包到網路上。

8.2 IIS (Internet Information Server)

8.2.1 簡介

IIS 是一套由微軟公司所推出的網際網路資訊伺服器，說它是資訊伺服器一點也不誇張，他本身提供了一般最常使用到的網路服務，它一向又與其自家的網路作業系統緊密的結合在一起，為的是要確保使用團體在建置 Internet 或是 Intranet 的安全性，由於嚴格的安全管制，使得管理 IIS 相當容易，並利用與 Microsoft Proxy Server、Certificate Server、Site Server、BackOfficeâ 以及其他應用程式緊密結合之便，提供一套內建整合性、極度安全的作業平台。因此無論是使用 Internet 或是 Intranet 都可以在不同的平台上迅速地建置網站。

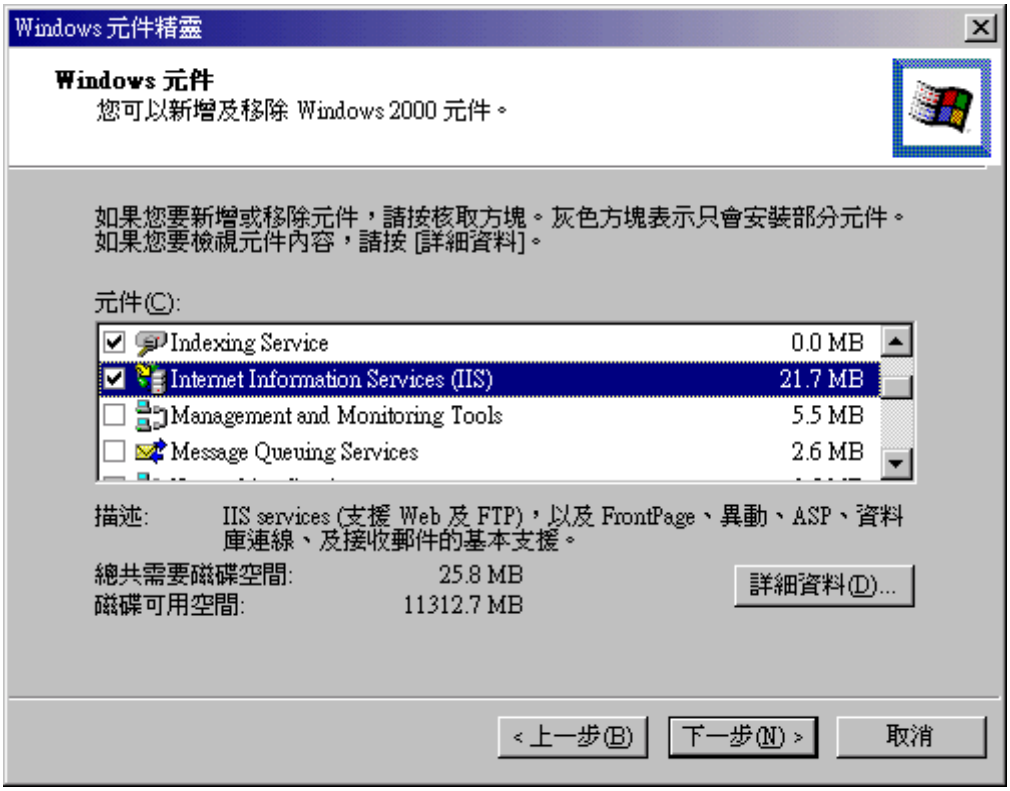
Windows 2000 Server 整合了 IIS 5.0，在預設的情況下，安裝 Windows 2000 Server 時便會自動安裝 IIS 5.0。

IIS 5.0 主要提供了下面四種服務：

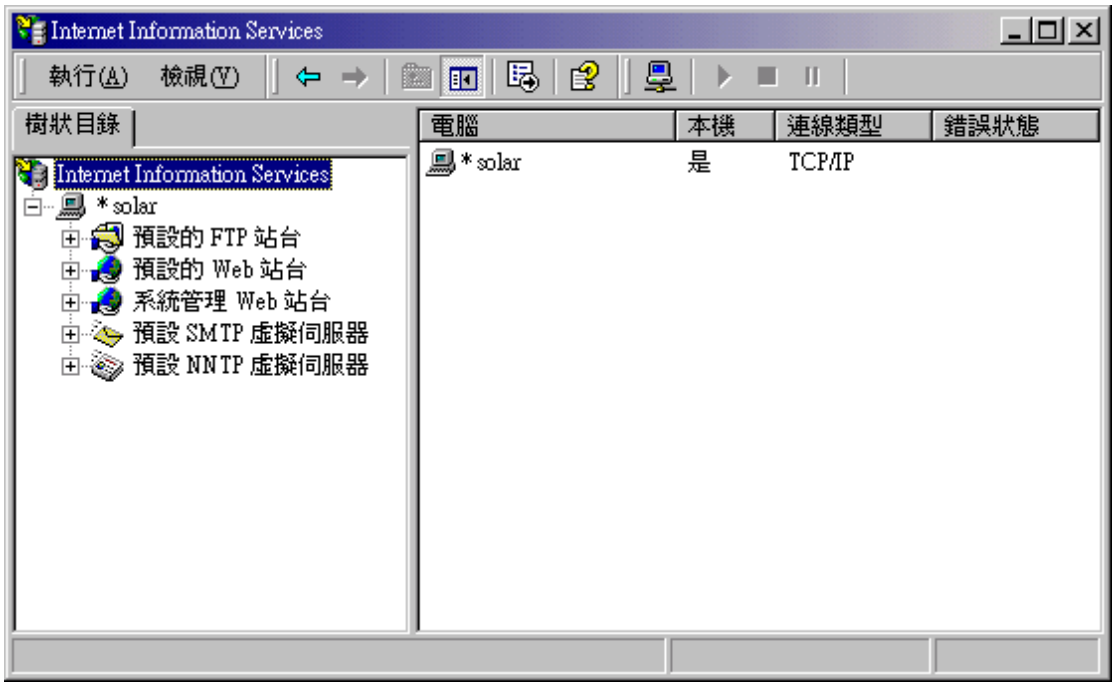
- **World Widw Web Publishing Service (WWW Servece)**
  網站伺服器 (Web Server)。
- **FTP Publishing Service (FTP Service)**
  檔案傳輸伺服器 (FTP Server)。
- **SMTP Service**
  郵件伺服器 (Mail Server)。
- **NNTP Service**
  新聞伺服器 (News Server)。

8.2.2 安裝

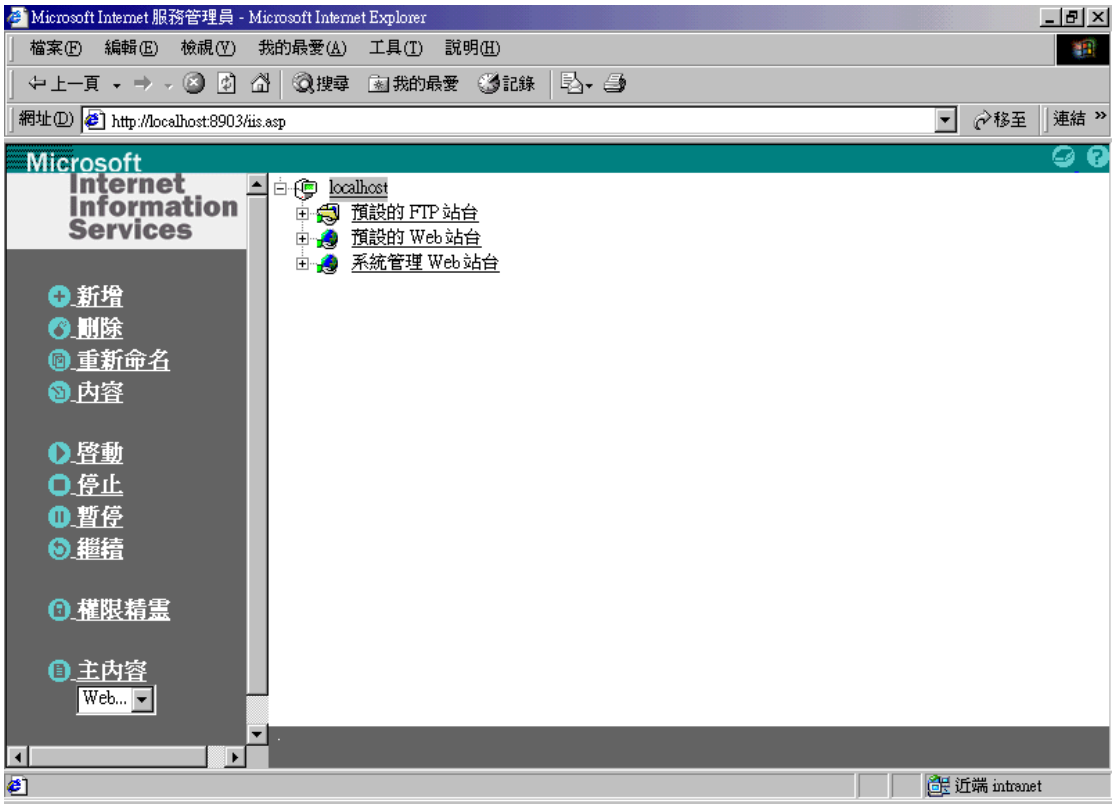
步驟一：從「控制台」→「新增或移除程式」→「新增/移除 Windows 元件」，勾選「Internet Information Services (IIS)」安裝 IIS 伺服器全部套件，或點選下方「詳細資料」進行細部的套件選取。



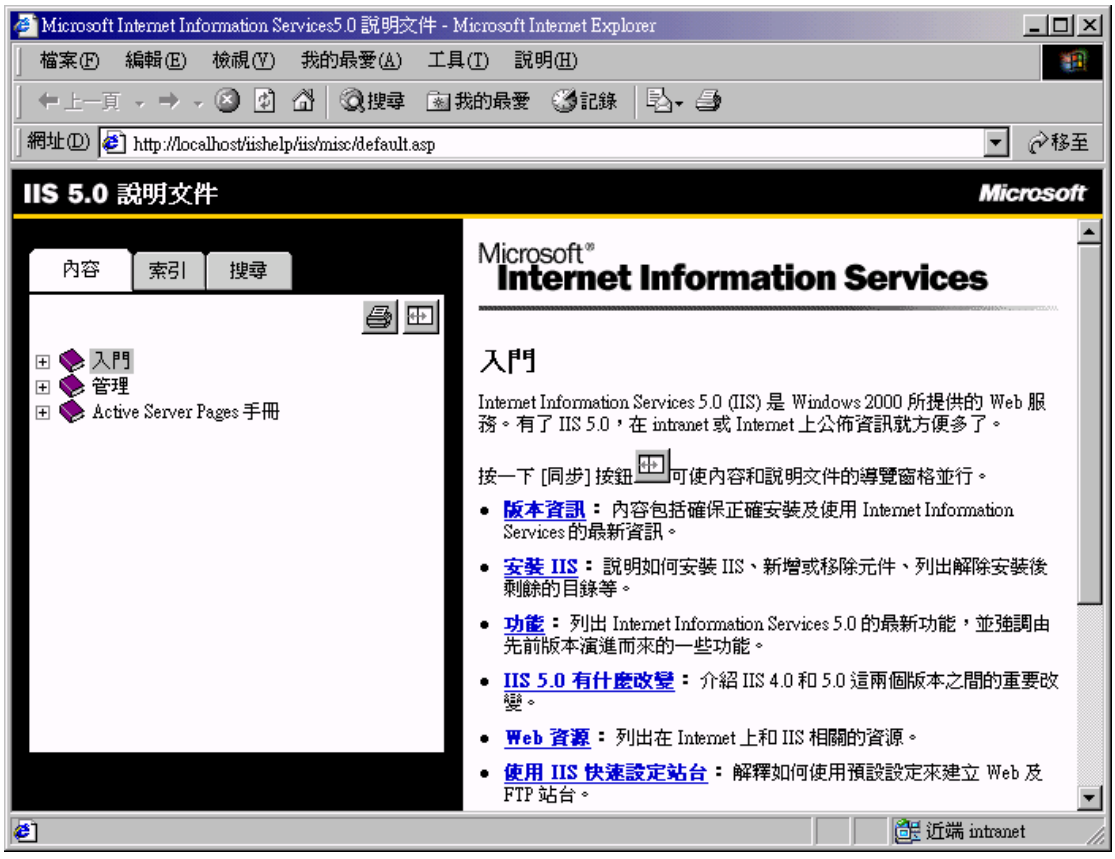
步驟二：安裝完後，可由「系統管理工具」→「Internet 服務管理員」管理整個 IIS 伺服器。



步驟三：IIS 伺服器提供 Web 介面的管理網頁，即「系統管理 Web 站台」。

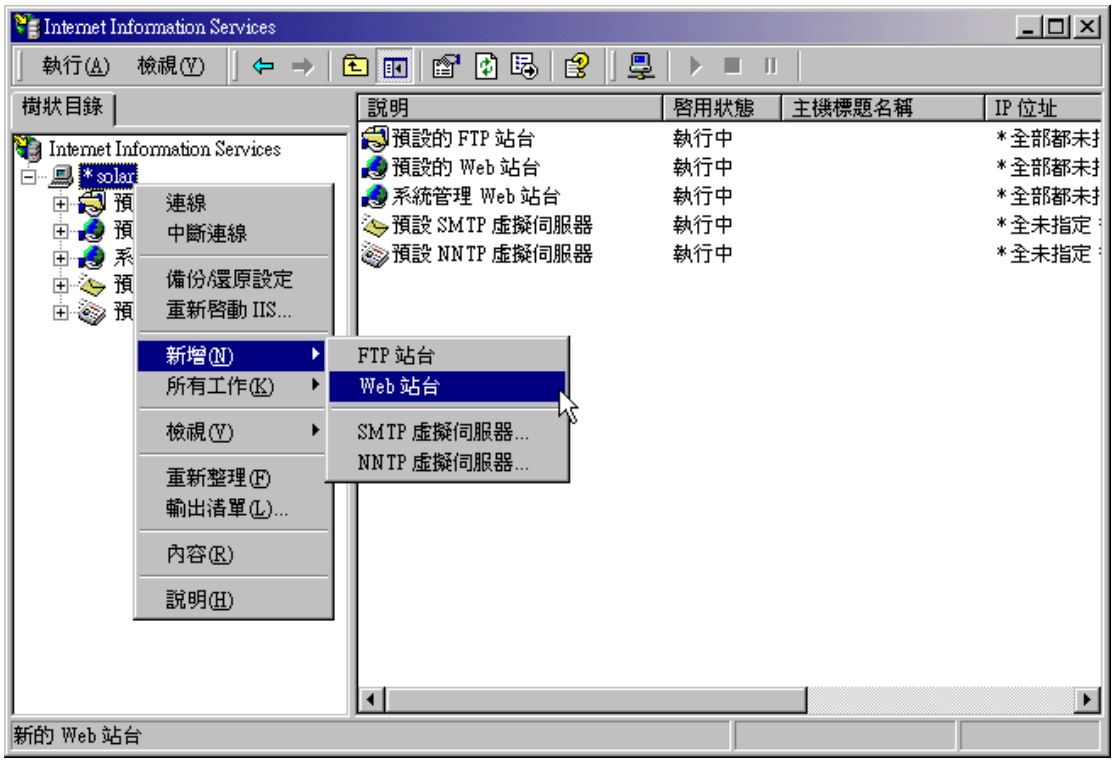


步驟四：「預設的 Web 站台」裡的內容是 IIS 的歡迎畫面及說明文件。



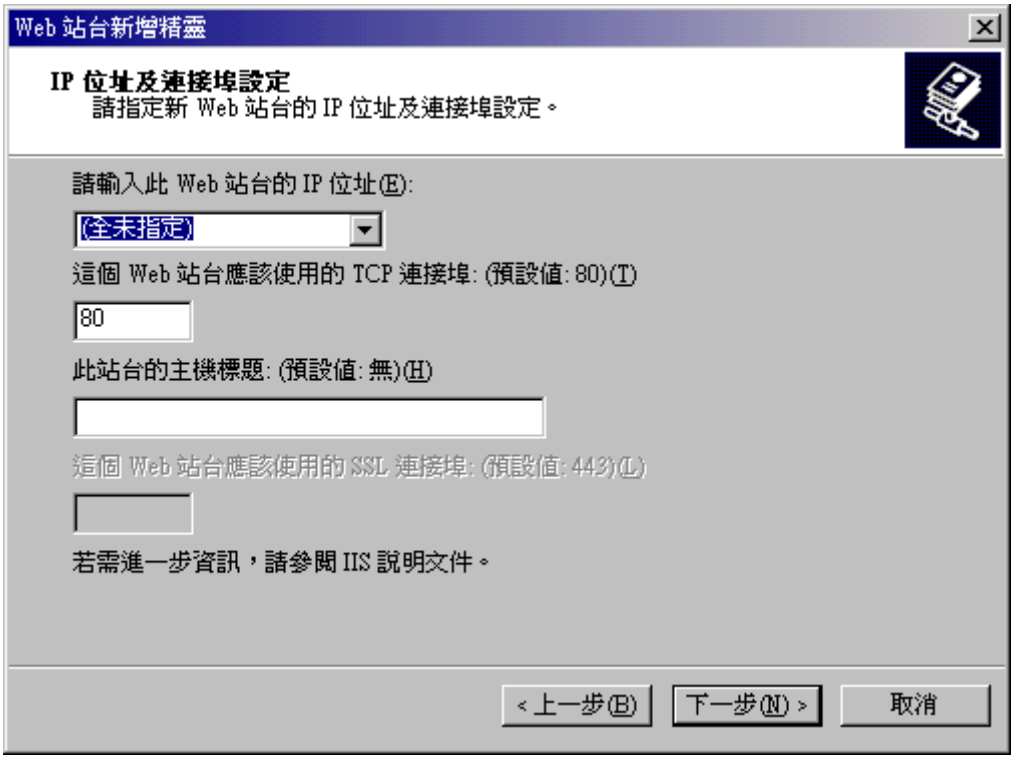
8.2.3 新增 Web 站台

步驟一：在管理工具中，本機電腦按滑鼠的右鍵，點選「新增」→「Web 站台」，接下來點選「下一步」略過歡迎使用畫面。

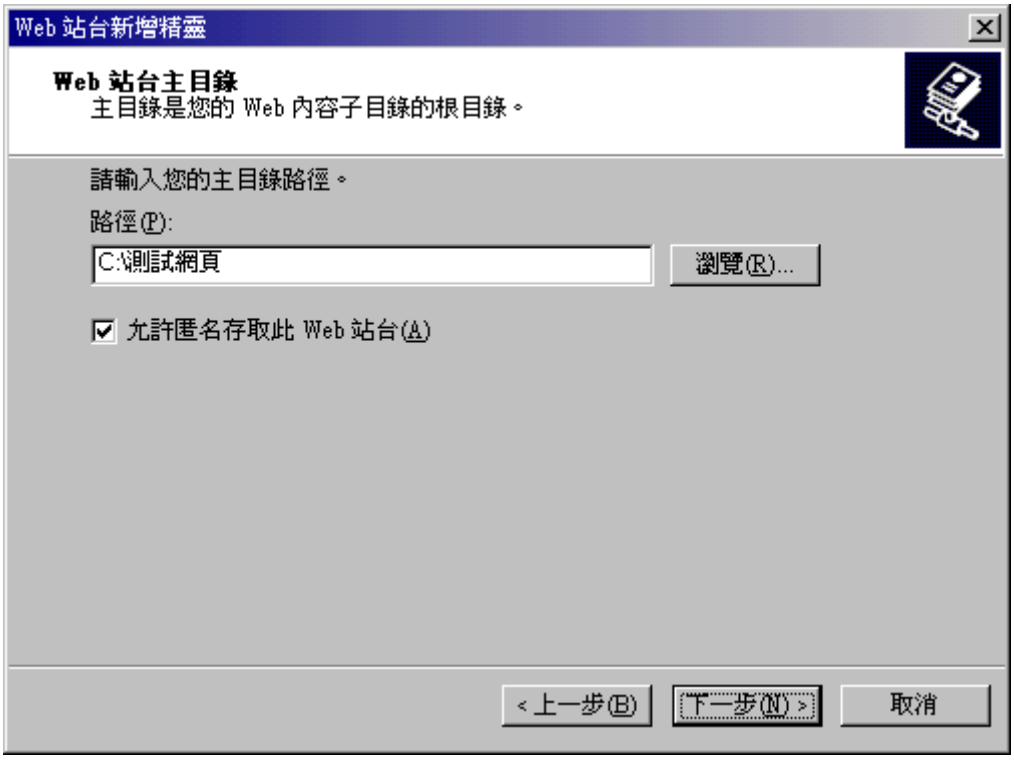


步驟二：輸入顯示在管理工具中的 Web 站台名稱。

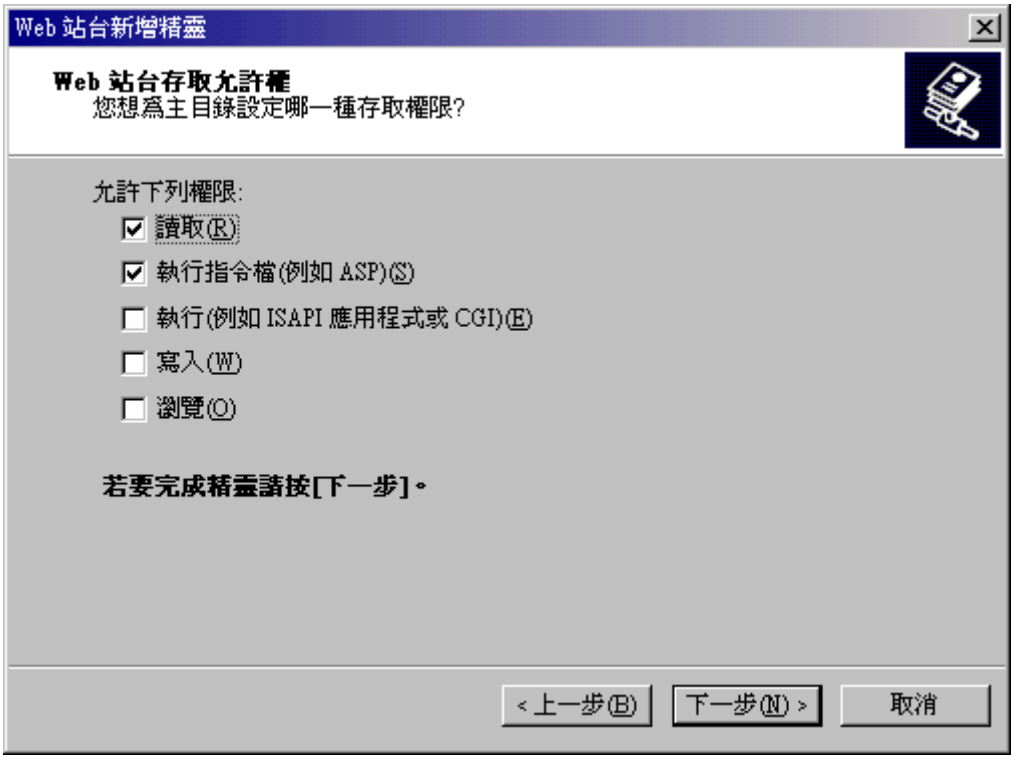
步驟三：輸入 Web 站台的 IP 位址，若選擇預設的「全未指定」，則此 Web 站台對應到該伺服主機中所有的 IP 位址(可能不只一個)，因此，連結任一個 IP 位址，都可以連到 Web 站台。



步驟四：輸入 Web 站台的根目錄。



步驟五：設定 Web 站台根目錄的存取權限。點選「下一步」後，即可完成新增 Web 站台的動作。

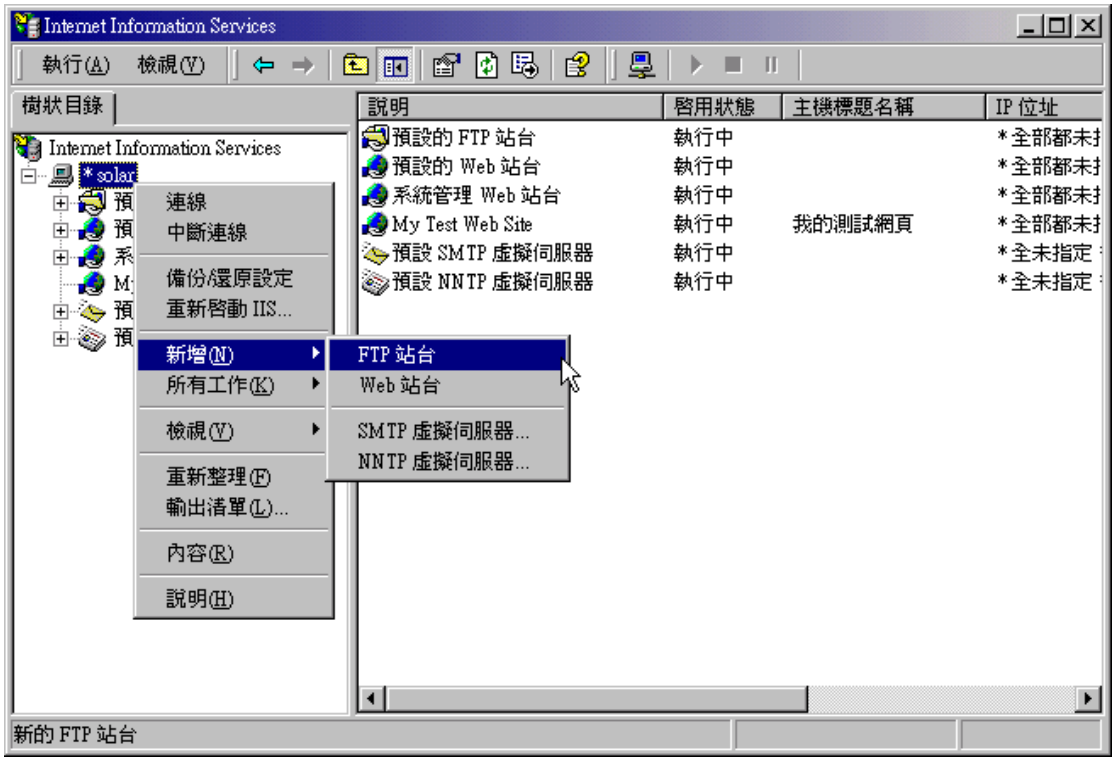


步驟六：在新增 Web 站台完成後，由於 TCP 通訊埠的重複，須先停止「預設的 Web 站台」，再將所編輯完成的網頁檔案移至 Web 站台的根目錄中即可。

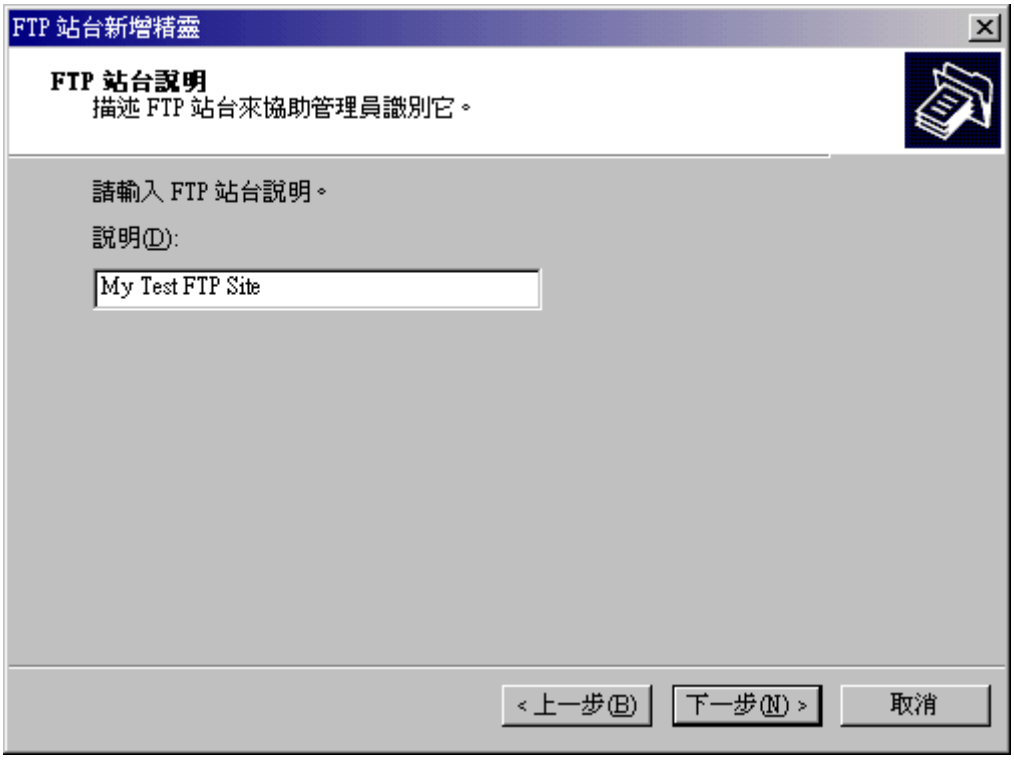
（IIS 的 Web 站台預設是自動開啟 Default.htm，而不是一般常用的 Index.htm，所以首頁的檔名要改成 Default.htm ）

8.2.4 新增 FTP 站台

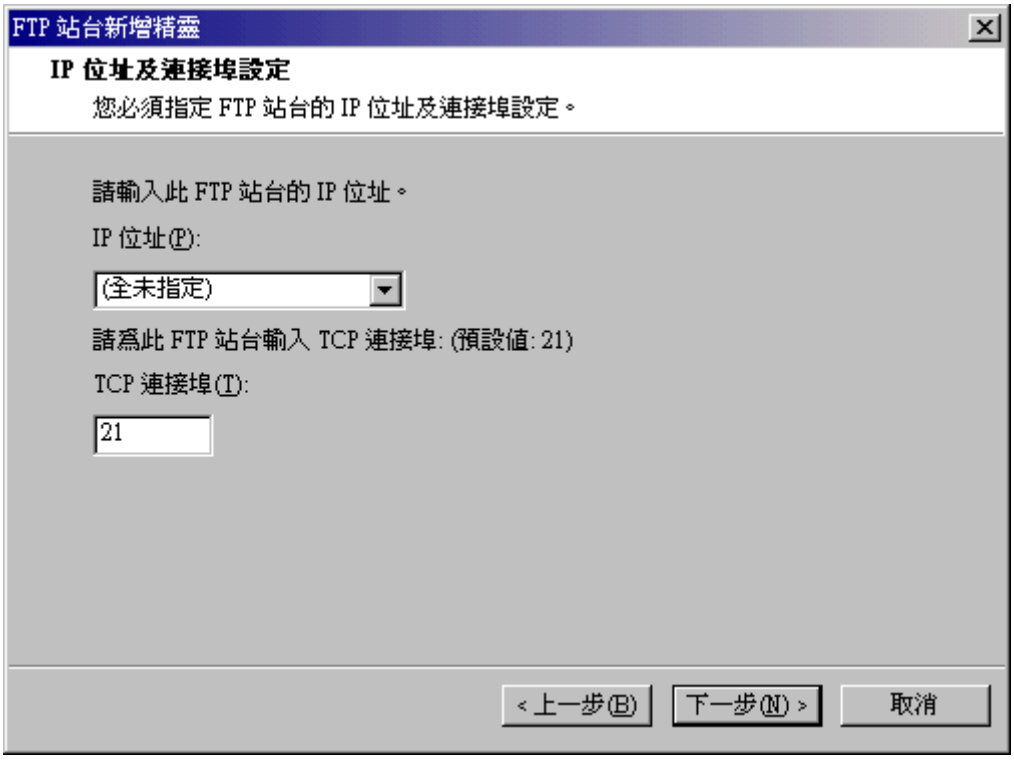
步驟一：在管理工具中，本機電腦按滑鼠的右鍵，點選「新增」→「FTP 站台」，並點選「下一步」略過歡迎使用畫面。



步驟二：輸入顯示在管理工具中的 FTP 站台名稱。

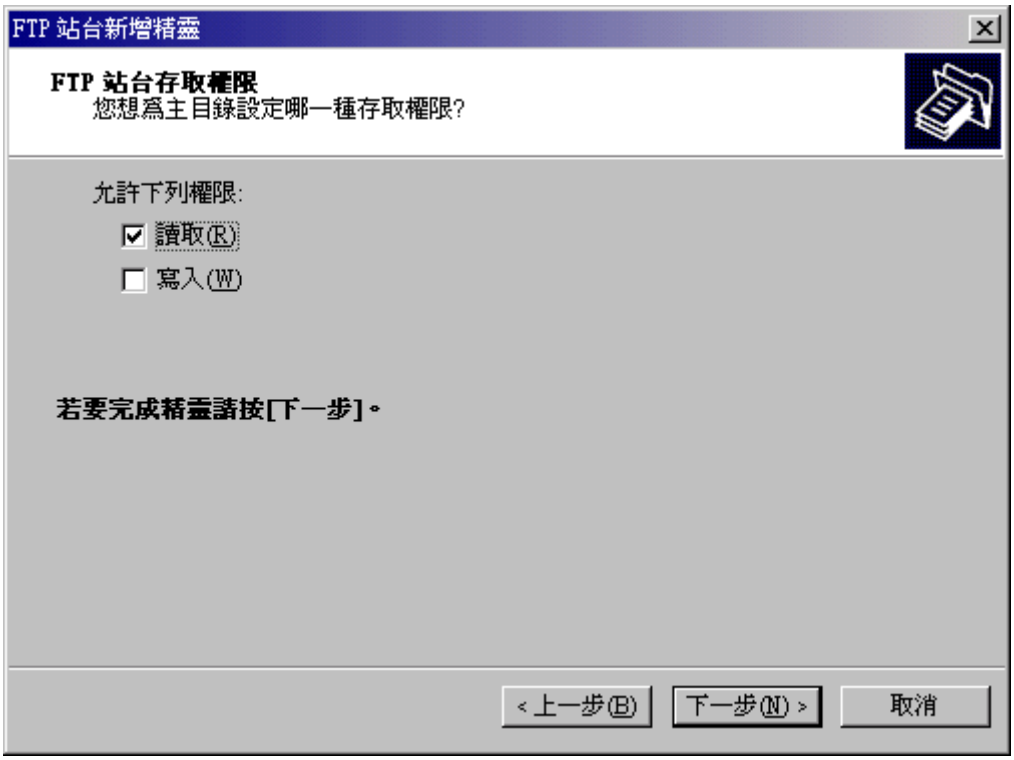


步驟三：輸入 FTP 站台的 IP 位址，若選擇預設的「全未指定」，則此 FTP 站台對應到該伺服主機中所有的 IP 位址(可能不只一個)，因此，連結任一個 IP 位址，都可以連到 FTP 站台。



步驟四：輸入 FTP 站台的根目錄。

步驟五：設定 FTP 站台主目錄的存取權限，點選「下一步」後即可完成 FTP 站台的新增動作。



第9章 系統安全

9.1 Kerberos

9.1.1 簡介

Kerberos 一詞源自希臘神話，是一隻負責守衛 Hades 以及地獄之門的三頭狗。1980 年代，Kerberos 安全機制在雅典那計畫中首次現身，雅典那計畫的主要目的是為了設計並實作一個分散式運算的環境。不過，直到第四個版本，Kerberos 才漸漸的開始嶄露頭角，受到青睞，並由 MIT 實驗室轉移到企業界中。在 Windows 2000 中所使用的 Kerberos 是第五個版本，支援許多種加密演算法，並提供雙邊認證的機制。

Kerberos 協定中有三種實體—使用者、伺服器或應用程式、被雙方信任的第三者，「被雙方信任的第三者」在 Windows 2000 Server 中指的就是會使用 Active Directory 資訊的「金鑰發行中心」(Key Distribution Center，KDC)。

9.1.2 驗證架構

1. 表示法

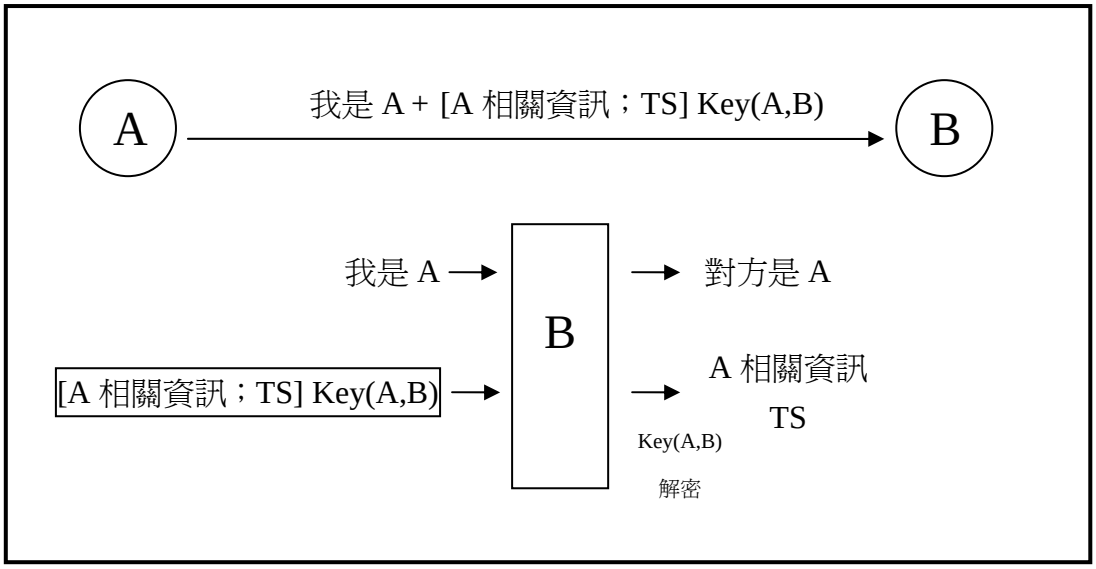
- Key(X,Y)：X 與 Y 擁有的密鑰。
- [DATA1；DATA2；…] Key(X,Y)：以密鑰 Key(X,Y)加密的資料 DATA1、DATA2、…等等。
- SKey(X,Y)：X 與 Y 共同的連線金鑰(在此稱為 Session Key)。
- T(X,Y)：X 用此 Ticket 跟 Y 驗證。
- TGT(X)：X 用來申請其他票證的票證。

2. 一對一驗證

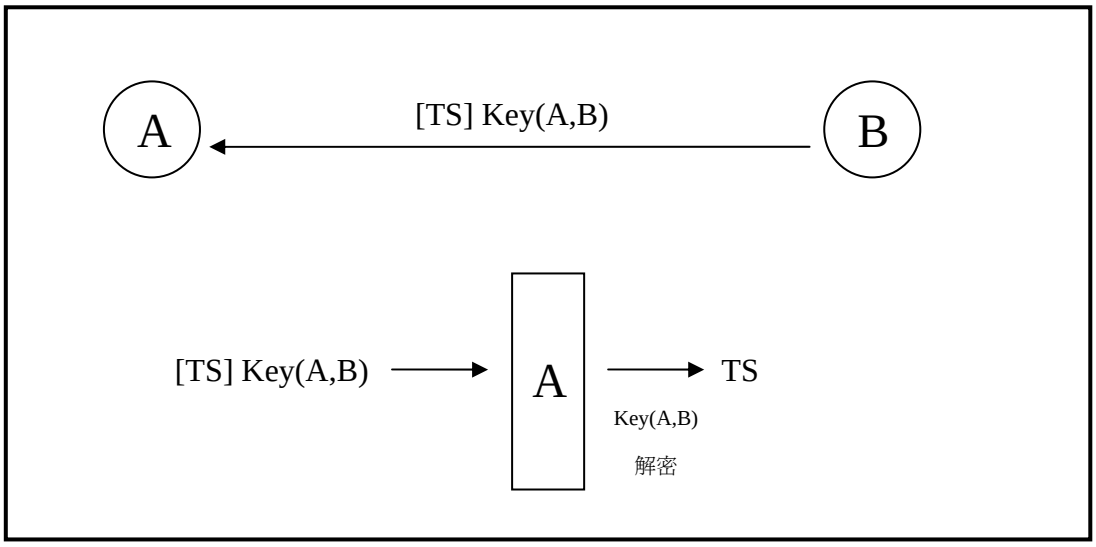
一對一驗證是最基本的架構。

狀況：A 與 B 要相互驗證。

A 與 B 分別擁有相同的密鑰 Key(A,B)。



B 收到訊息後，知道對方是 A，另外利用密鑰解密得到 A 相關資訊與時間郵戳（Time Stamp，TS），將明文部份與 A 相關資訊比對，且 TS 與 B 本身目前的時間差異不大，即可確認 A 的身分。



B 以密鑰將 A 傳來的 TS 加密，再回傳給 A，解密後將 TS 與原來所紀錄的 TS 比較，若相同則可確認 B 的身分。

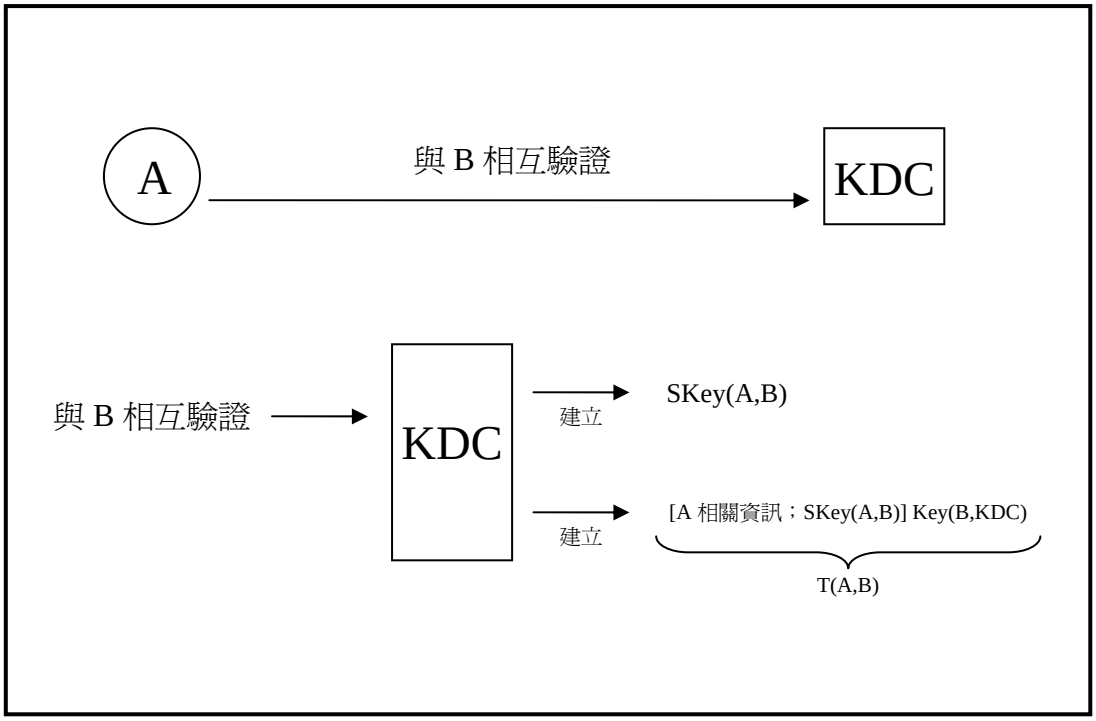
3. 集中式驗證

為了解決一對一驗證密鑰管理的不易，發展出將密鑰集中管理的金鑰發布中心（Key Distributed Center，KDC)。

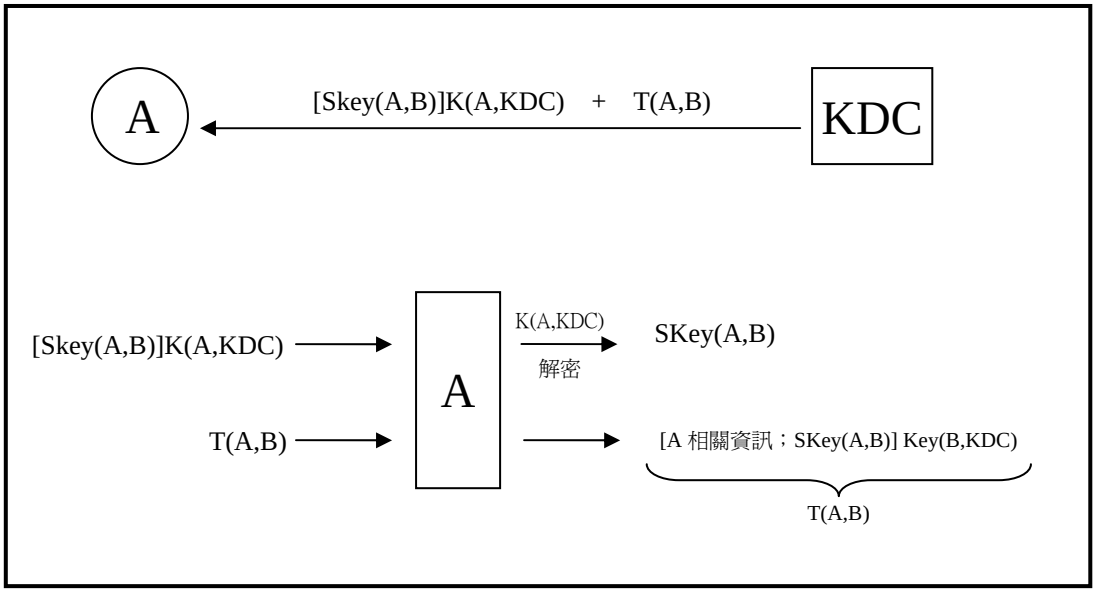
狀況：A 與 B 透過 KDC 來進行相互驗證。

A 與 KDC 分別擁有相同的密鑰 Key(A,KDC)。

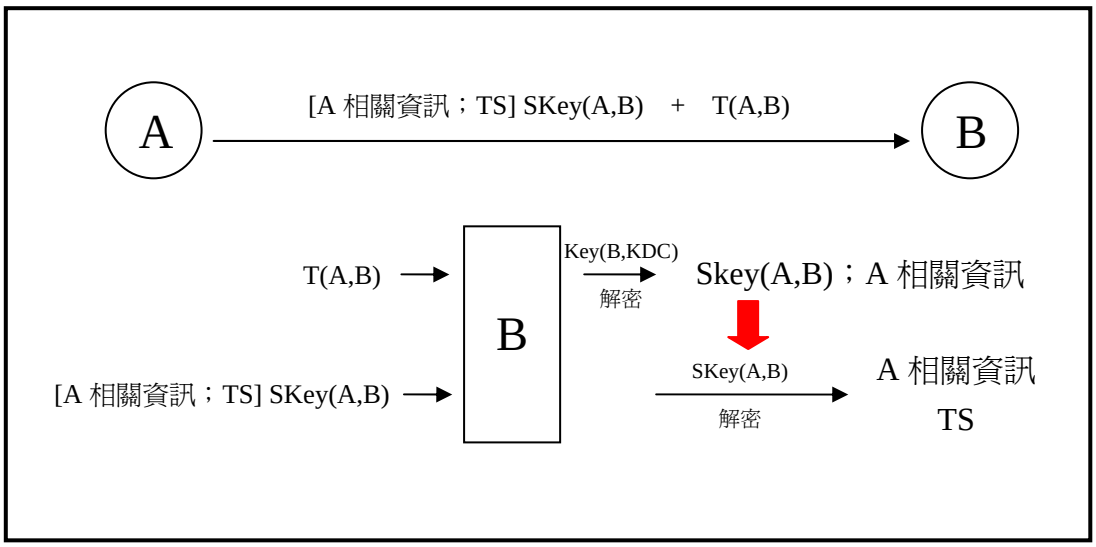
B 則與 KDC 分別擁有相同的密鑰 Key(B,KDC)。



A 必須先與 KDC 進行一對一驗證，完成後，當 A 要與 B 作驗證時，首先，A 先傳送欲跟 B 進行驗證的訊息給 KDC，這時 KDC 會產生一組(兩把)連線金鑰（Session Key「SKey(A,B)」)，以及一張票證（Ticket「T(A,B)」)，其內容包括 A 的相關資訊及 KDC 建立的其中一把連線金鑰 SKey(A,B)，票證是以 Key(B,KDC)加密。



KDC 將 Skey(A, B)以 K(A, KDC)金鑰加密，連同票證傳送給 A，A 收到後只能解密取得 A 與 B 之間的連線金鑰 Skey(A, B)。



與一對一驗證類似，A 以 Skey(A, B)金鑰加密相關資訊與時間郵戳，連同重 KDC 獲得的票正一併傳送給 B。B 在收到後，首先先以 K(B, KDC)金鑰解密得到 A 與 B 的連線金鑰 Skey(A, B)與 A 提供給 KDC 的相關資料，再以 Skey(A, B)解密的到 A 提供給 B 的相關資訊與時間郵戳，比對兩份資料且時間郵戳在容許的範圍內，即完成了確認 A 的動作。

接下來，A 確認 B 的過程與一對一驗證的確認相同，故不多加詳述。

集中式驗證架構有一些缺點：

- 使用者每次與不同的伺服器進行驗證時，都會使用到 K(X, KDC)，所以每次都要輸入一次密碼，稍嫌麻煩。
- 使用者每次向 KDC 要求票證時，KDC 必須重複的在資料庫中找出使用者的密碼，然後以雜湊函數求出 K(X, KDC)。

4. Kerberos V5 驗證

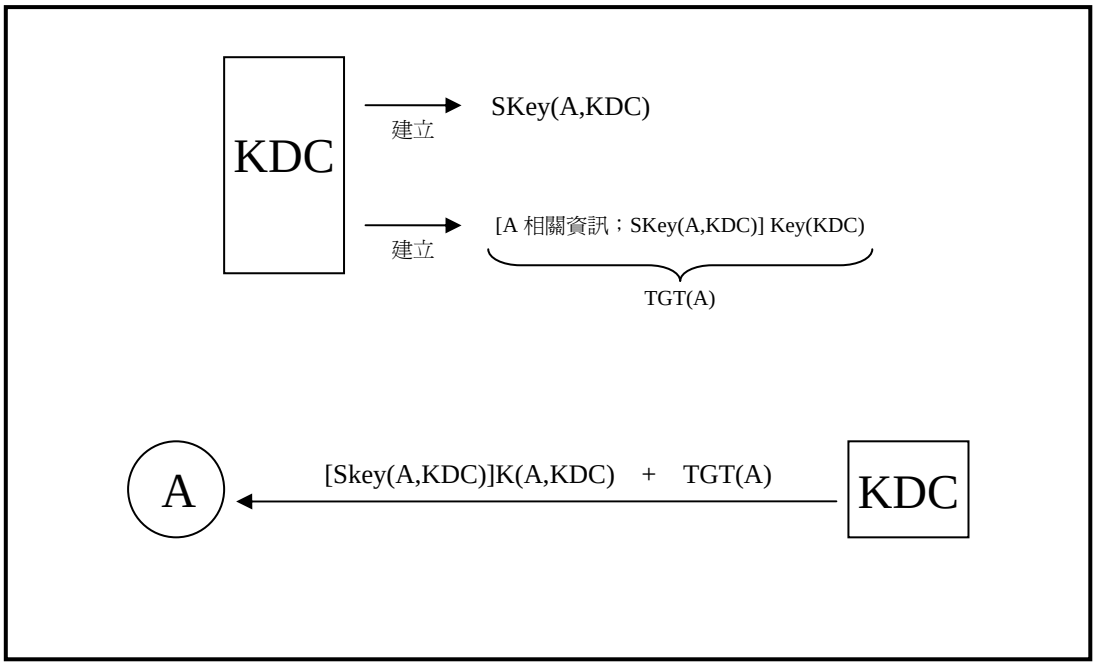
為了解決集中式驗證的不便，Kerberos 第五版中加入了票證認可票證(Ticket-Grantd Ticket，TGT)的概念。

狀況：A 與 B 透過 KDC 來進行相互驗證。

A 與 KDC 分別擁有相同的密鑰 Key(A,KDC)。

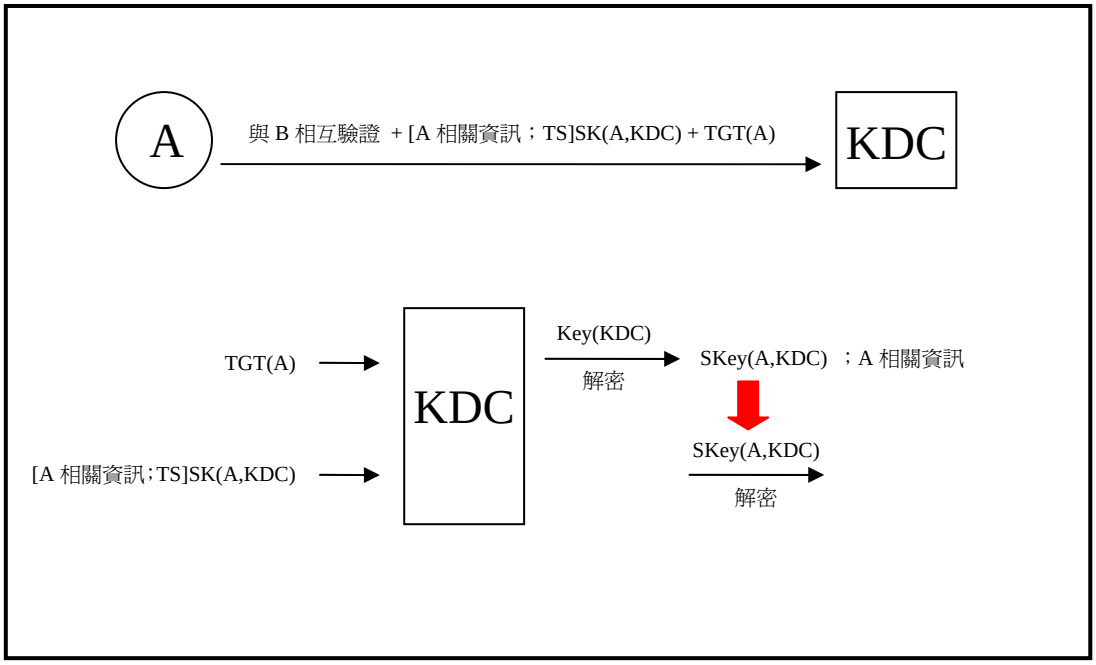
B 則與 KDC 分別擁有相同的密鑰 Key(B,KDC)。

KDC 擁有一把專屬的密鑰 Key(KDC)。



A 必須先與 KDC 進行相互驗證，完成後 KDC 會產生一組(兩把)連線金鑰(Session Key「SKey(A,KDC)」)，以及一張票證認可票證「TGT(A)」，內容包括 A 的相關資訊及 KDC 建立的其中一把連線金鑰 SKey(A,KDC)，TGT(A)是以 Key(KDC)金鑰加密。

KDC 以 Key(A,KDC)將剩下的另一把連線金鑰 SKey(A,KDC)加密後，連同 TGT(A)一起傳送給 A。A 收到後得到票證認可票證 TGT(A)及解密取得的連線金鑰 SKey(A,KDC)。



當 A 要與 B 驗證時，A 會將要與 B 相互驗證的訊息、以 SKey(A,KDC)加密的 A 相關資訊與時間郵戳「TS」以及從 KDC 獲得的票證認可票證「TGT(A)」傳送給 KDC。KDC 先解密取得 SKey(A,KDC)與 A 相關資訊後，再以 SKey(A,KDC)解密取得 A 相關資訊與時間郵戳，將兩份 A 相關資訊比對且時間郵戳在許可的範圍內，即可確認 A 的身分。

KDC 在完成 A 的身分確認後，會立即建立一組連線金鑰 Skey(A,B)，然後透過 Ticket(A,B)將兩把連線金鑰分別送給 A 和 B，接下來的步驟即與集中式驗證的步驟完全相同。

9.1.3 登入程序

1. 登入

步驟一：用戶端將帳號名稱及加密過的密碼，傳送至 KDC。

步驟二：KDC 收到用戶端傳來的資訊後，傳回一個加密的訊息給用戶端，包括了使用者的密碼、一個票證認可票證 (Ticket-Granting Ticket，TGT)以及一個時間郵戳 (Time Stamp)的連線金鑰，這個連線金鑰是用來作為用戶端與金鑰中心之間的認證。

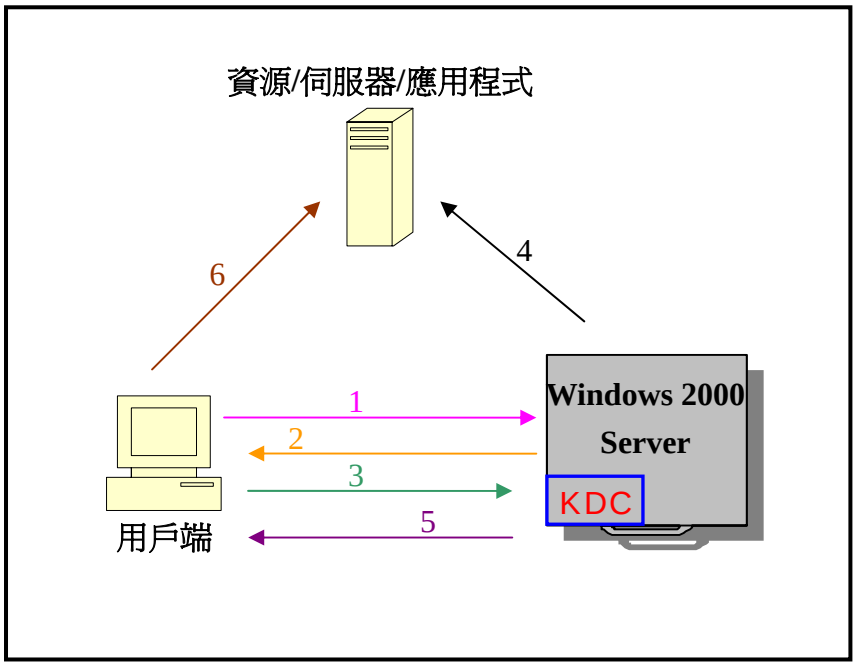
2. 使用資源(伺服器)

步驟三：用戶端以連線金鑰將請求與資源溝通的訊息加密，並傳送到 KDC。訊息的內容包括資源或伺服器的名稱、一個時間郵戳，以及票證認可票證。由於訊息是由用戶端的連線金鑰所加密，所以 KDC 知道該用戶端是一個合法用戶。

步驟四：KDC 產生一個共用的連線金鑰讓用戶端與資源(伺服器)使用，同時也產生一個特殊的票證給資源(伺服器)使用，其中包括共用的連線金鑰(Session Key)、用戶端名稱、用戶端網路卡位址、票證使用期限及一個時間郵戳。

步驟五：KDC 傳送一個訊息給用戶端，包含加密、共用的連線金鑰，及加密後的票證。該共用連線金鑰是以用戶端的連線金鑰加密，而票證是用伺服器的連線金鑰來加密。

步驟六：用戶端傳送以共用連線金鑰加密的票證(由上一步驟 KDC 取得)和一個時間郵戳給資源(伺服器)，資源(伺服器)再利用共用連線金鑰來解開加密的訊息，根據時間郵戳來檢查票證的有效性。若一切正確，則資源(伺服器)便可接受用戶端的請求。



9.2 智慧卡

智慧卡是一張和信用卡大小差不多的塑膠卡片，上面放置有一微處理晶片，可供您儲存重要的資訊。您可以在智慧卡中儲存您個人的認證和私密金鑰以執行公共金鑰的加密作業，例如驗證、數位簽章和金鑰交換。

Windows 2000 首度將智慧卡的功能整合到作業系統中，使用者可以利用智慧卡來執行驗證、電子郵件加密以及數位簽章，其優點如下：

- 您的私密金鑰和其他型式的個人身份識別不會遭到他人的竄改。
- 可以區隔涉及驗證、數位簽章和金鑰交換等重要的安全資料與系統中其他不需使用這些資料的部份。
- 保護認證和其他私人資訊在電腦之間傳送的安全性（例如，從辦公室的電腦傳送資料到家中或遠端的電腦中）。

9.2.1 啟用智慧卡

智慧卡的驗證，可以用 Windows 2000 憑證服務來執行，從憑證授權主控台來啟用 CA 發行智慧卡憑證。

使用智慧卡登入的驗證步驟為：

步驟一：使用者將智慧卡插入讀取裝置中。

步驟二：智慧卡服務要求使用者輸入 PIN 碼。

步驟三：智慧卡驗證 PIN 碼。

步驟四：確認無誤後，LSA 將驗證資訊傳給 Kerberos 用戶端來驗證網域控制站。

步驟五：使用者通過驗證後，即可登入。

9.3 憑證

9.3.1 簡介

Windows 2000 作業系統中有一個 Certification Service，它就是發行授權憑證的服務，而有許多的安全服務都是使用憑證服務的方式，透過公開金鑰，來建構使用者身份驗證，以及其他安全性服務的基礎。使用者可利用他們取得的憑證資訊，發送加密過的電子郵件，也可以對檔做簽證(即所謂的電子簽章)，甚至使用者可以將憑證資訊儲存在智慧卡 (smart card)上，透過刷卡的方式，就能登入系統。

9.3.2 憑證授權單位

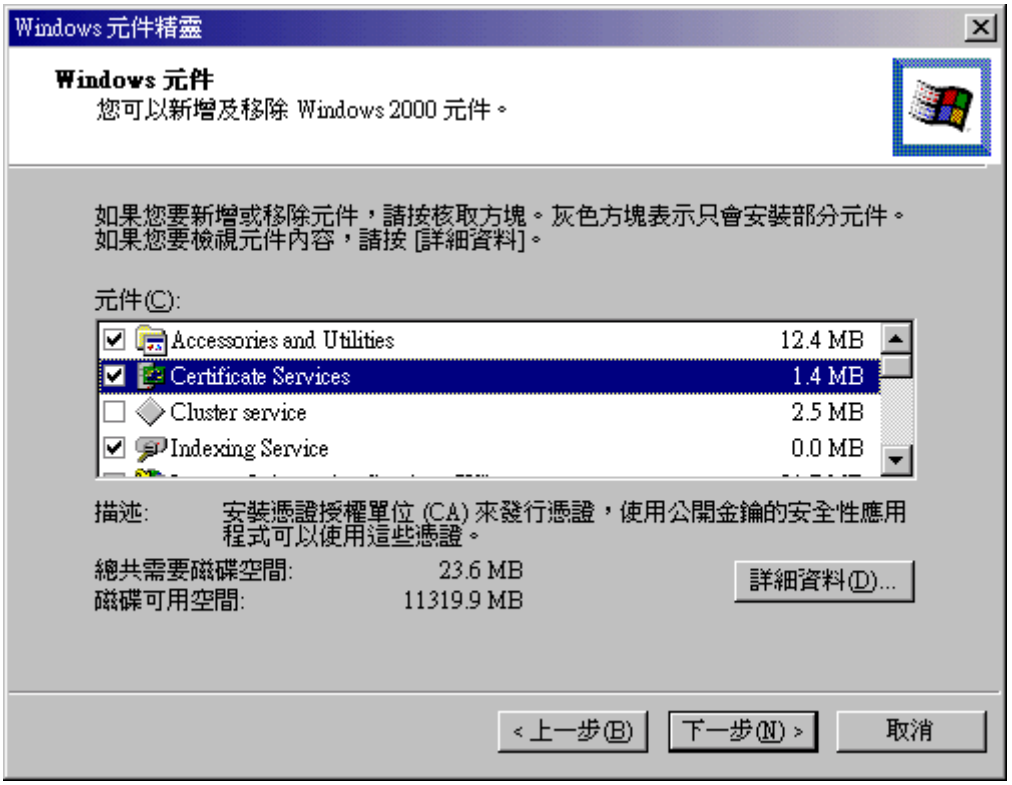
憑證授權單位 (Certificate Authority，CA)是提供發行憑證服務的單元，CA 必須負責審核每一位要求憑證資訊的使用者身份，以決定是否要將憑證資訊發行給該名使用者。

CA 的授權服務是以微軟的公開金鑰系統 (Public Key Infrustructre，PKI)的編碼技術來建立，並適用於網際網路的安全性架構，所以，CA 可以由外界的商用性來擔任，或者自行在公司內部網路建構憑證授權單位。CA 是以階層式的架構存在的，最上層的 CA 又稱為 Root CA(根憑證授權單位)，在 Root CA 底下可以有許多子 CA，又稱次級憑證授權單位 (Subordinate CA)。

Windows 2000 作業系統裡提供了兩種不同型態的憑證授權單位，一種是與 Active Directory 整合的企業型憑證授權單位 (Enterprise CA)，此類 CA 是應用在公司內部與 AD 整合的情況，所有的授權憑證是存放在 AD 上；另一種是獨立型憑證授權單位 (Stand-alone CA)，獨立型憑證授權單位通常會應用在與非企業內部的 CA 做信任的架構下，它的授權憑證則是存放在該 CA 的電腦本機上之硬碟裡。

9.3.3 安裝

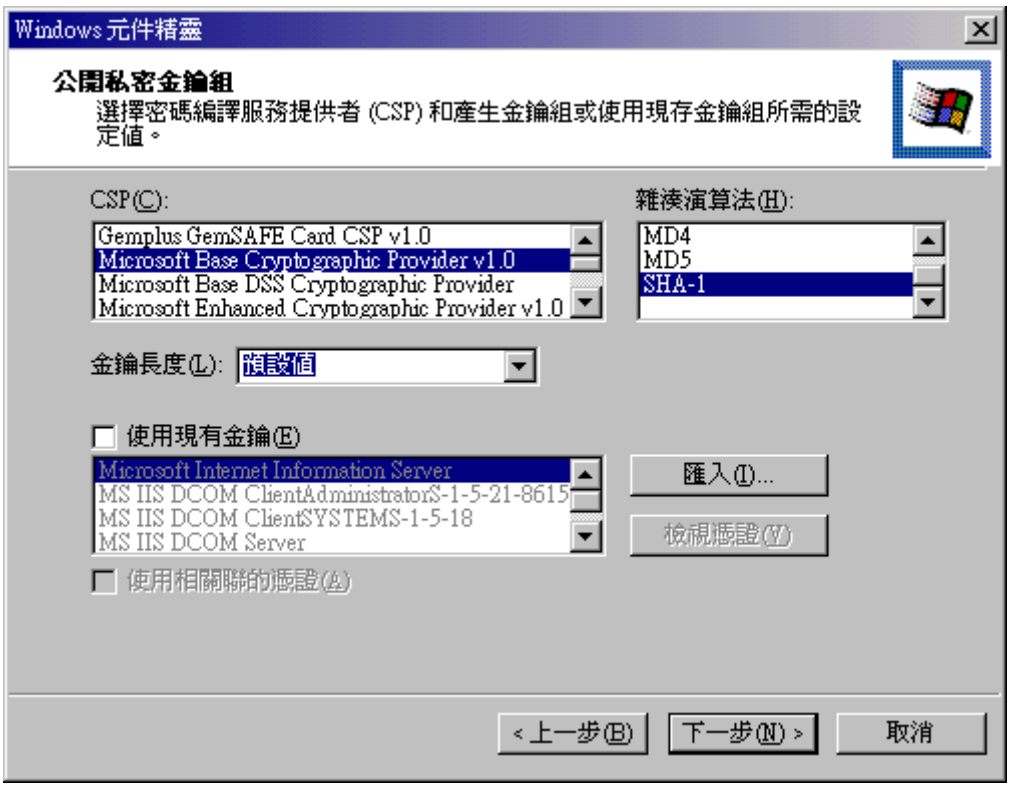
步驟一：從「控制台」→「新增或移除程式」→「新增/移除 Windows 元件」，勾選憑證服務 (Certificate Services)。



步驟二：因為安裝憑證服務後，此台電腦將無法重新命名、加入網域或從網域移除，所以要求使用者確認。

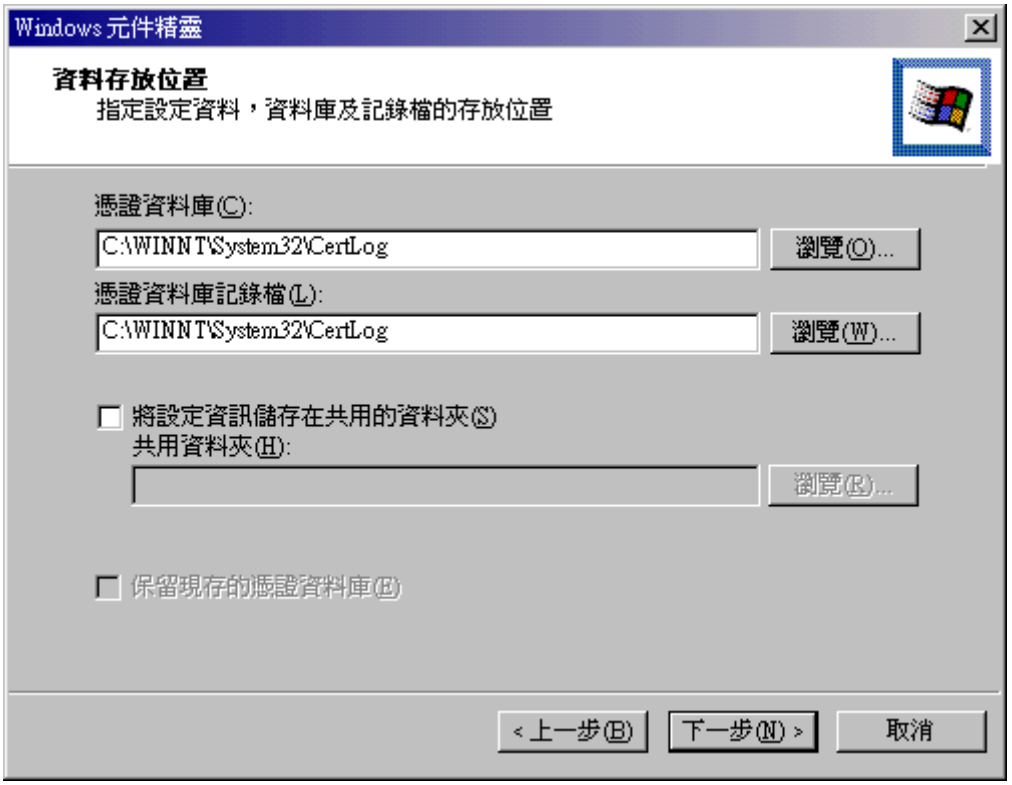
步驟三：選擇要安裝的憑證授權單位類型，下方「進階選項」是讓使用者可以自行作公開私密鑰的設定。

步驟四：選擇密碼編譯服務的提供者和其他設定值，金鑰長度越長，加密解密的時間也就越長，但密文相對的也就越安全。



步驟五：輸入 CA 的相關資訊，當 CA 的有效期限過期時，管理人員就必須重新更新一次所有的信任關係，考量到安全性及系統管理的負擔，根憑證授權單位的有效期間必須合理，預設為兩年。

步驟六：指定憑證資料庫儲存的位置。



步驟七：因為憑證服務同時也支援 IIS 伺服器的運作，所以若 IIS 伺服器已經處於啟動的狀態，安裝精靈會要求先將 IIS 伺服器停止。

接下來，便開始安裝憑證伺服器，安裝完成後可以在「系統管理工具」中啟動憑證授權的管理工具。

9.4 加密檔案系統

9.4.1 簡介

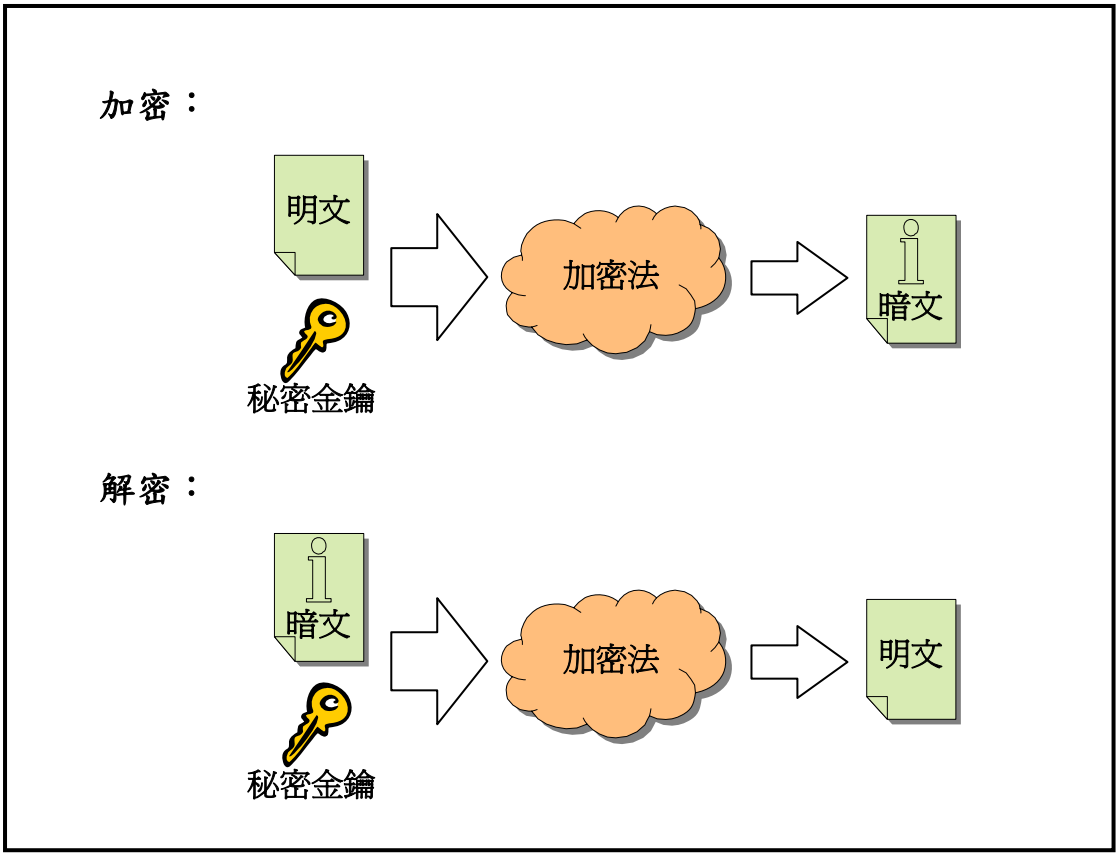
加密，是一種保護資料的技巧，通常使用數學演算法將原始資料（我們稱明文)配合特殊的方式加以編碼，形成無法輕易解讀的暗文。

加密檔案系統 (Encrypting File System，EFS)是微軟在 Windows 2000 中新提供的一項檔案安全功能。配合 NTFS 5.0，加密檔案系統能夠對檔案加密，使檔案不會受到未經授權的使用者破壞或窺視。

9.4.2 原理介紹

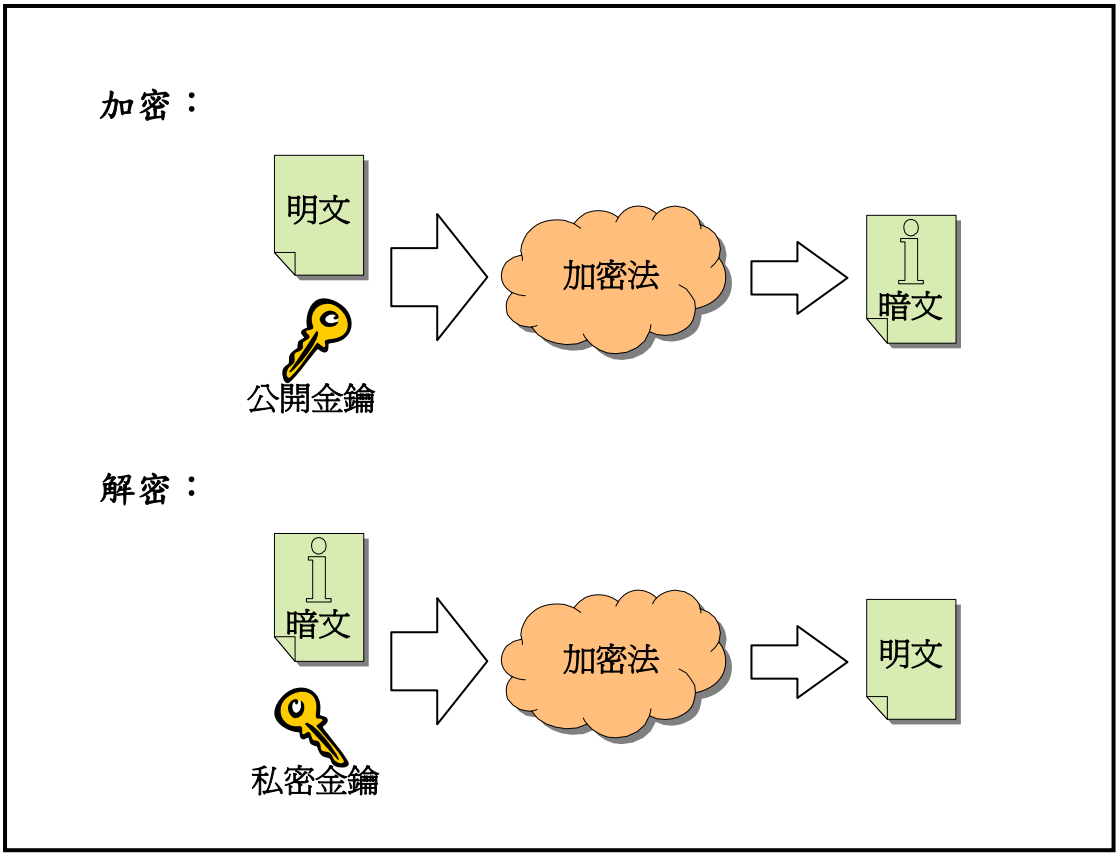
1. 對稱式加密法：

又稱傳統的秘密金鑰 (Secret Key)加密法。這種方法使用唯一的秘密金鑰來做加密/解密，具有權限的使用者都會擁有這一秘密金鑰。譬如，某甲想與某乙交換資料，甲須先利用秘密金鑰配合加密法對資料加密後傳送給乙，乙再用相同的秘密金鑰配合加密法進行解密。這種方式有個缺點，甲必須設法安全的讓乙獲得該秘密金鑰，因為任何人一旦擁有這秘密金鑰，便可以對資料解密。



2.非對稱式加密法：

公開金鑰 (Public Key，PK)加密法，因為使用不同的加密金鑰和解密金鑰而得名。每個人都擁有一對金鑰，包含了公開金鑰與私密金鑰 (Private Key)，公開金鑰處理加密的動作，將明文轉成暗文，而私密金鑰則是負責解密的動作。由於是公開的，任何想要傳送資料給甲的人，皆可以利用甲的公開金鑰對資料加密再傳送給甲，甲在收到暗文之後，再利用非公開的私密金鑰解密，回復成原來的資料。這種公開金鑰與解密過程無關的方式相當安全，即使是加密者本身也無法將暗文解密為明文，只有擁有與公開金鑰對應之私密金鑰的人，才能對暗文進行解密的動作。



加密檔案系統結合了上述兩項加密法，利用對稱式加密法對檔案進行加密與解密，而非對稱式加密法則負責加密檔案系統基本的身分驗證。

9.4.3 運作方式

加密檔案系統中的每一個檔案都會先以亂數的方式自動產生出一個檔案加密金鑰，也會對每個第一次使用加密檔案系統的使用者，根據使用者的相關資訊，自動產生一組公開金鑰與私密金鑰，以及加密檔案系統本身所使用的身分憑證資訊。

系統中，所有檔案的檔案加密金鑰清單是儲存在一個由使用者的公開金鑰加密過的檔案中，使用者要存取加密檔案系統中的檔案時，加密檔案系統會先檢查使用者的身分憑證資訊，確定使用者具有權限後，再由使用者的私密金鑰對檔案加密金鑰清單解密，找到相對應的檔案加密金鑰後，透過該檔案加密金鑰對檔案進行解密，再交由使用者使用，對於使用者而言，這些繁複的處理動作是透明的，使用者並不會感覺到這些動作的存在。

9.4.4 加密檔案系統的特性與原則

1. 特性
   - ●僅支援 NTFS 5.0 檔案格式，其餘格式 (FAT、NTFS 4.0)皆不支援。
   - ●不對資料夾加密，只針對資料夾中的檔案作加密的動作。
   - ●不支援網路傳輸資料的加密。
   - ●不能對壓縮過、唯讀或系統檔案加密。
   - ●加密過的檔案不能被壓縮、共用。
   - ●預設為 56 位元加密方式。

2. 原則
   - ●每個檔案都具有唯一的檔案加密金鑰。
   - ●使用者的公開金鑰用來對檔案加密金鑰作加密。
   - ●修復金鑰目的跟公開金鑰相同，都是為了保護檔案加密金鑰，能在無需使用私密金鑰的情況下進行解密修復。

9.5 稽核系統

9.5.1 簡介

Windows 2000 的安全子系統有兩個主要的工作：限制物件的存取並提供稽核的服務來追蹤物件的使用紀錄。稽核系統會將物件的使用資訊存入紀錄檔(Log File)，也可以讓管理人員監視物件所觸發的事件以找出可能的安全漏洞，管理人員可利用稽核紀錄來判斷漏洞對整個系統的損害，再進行修復。

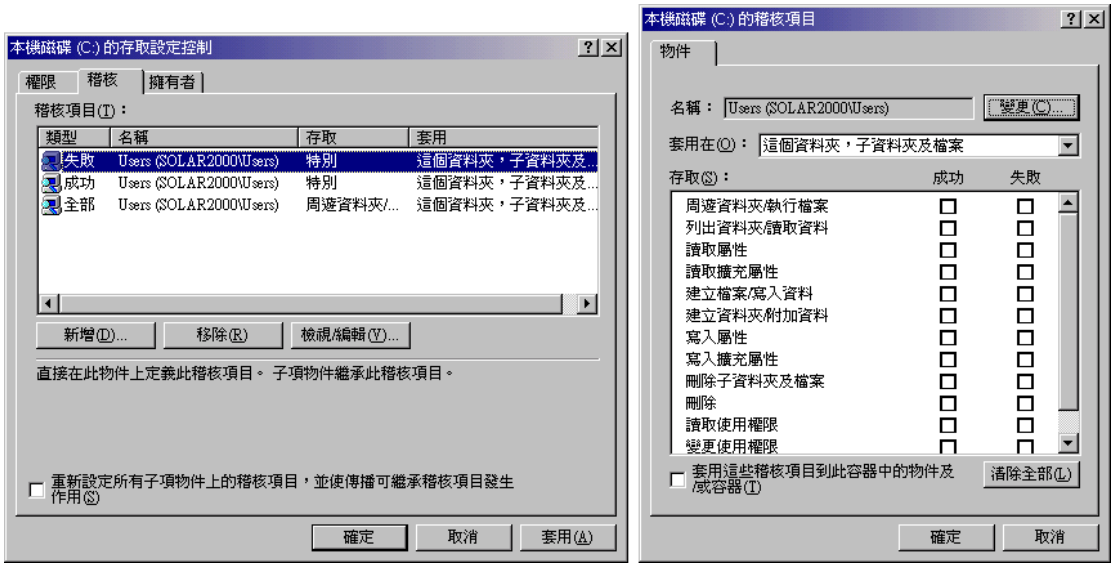
只要攻擊會對網路造成威脅或是不利於你認為珍貴的資源,則不論攻擊是否成功，稽核都應該要能辨識攻擊，這是風險評估的一部分。

稽核的事件可分為兩大類：成功的事件與失敗的事件。成功的事件是指使用者已經成功存取資源，而失敗的事件則是指使用者曾經試圖存取但是失敗了。雖然絕大部分成功的稽核事件都只是一般活動的指標,但處心積慮的攻擊者在成功存取系統時也會產生成功的事件。

決定適合的稽核等級是整體安全性策略的一部分，在決定稽核的範圍時，您應該記住：稽核的項目的越多，產生的事件就越多，要找出重要事件也就越困難；如果稽核的範圍很廣，您可以考慮使用其他工具，例如 Microsoft Operations

Manager，來幫您篩選比較重要的事件。當察覺到非法的活動時，建議稽核下列幾項事件：

- 所有安全設定的變更
- 失敗的登入嘗試
- 嘗試存取敏感性資料
- 所有系統啟動與關閉的事件



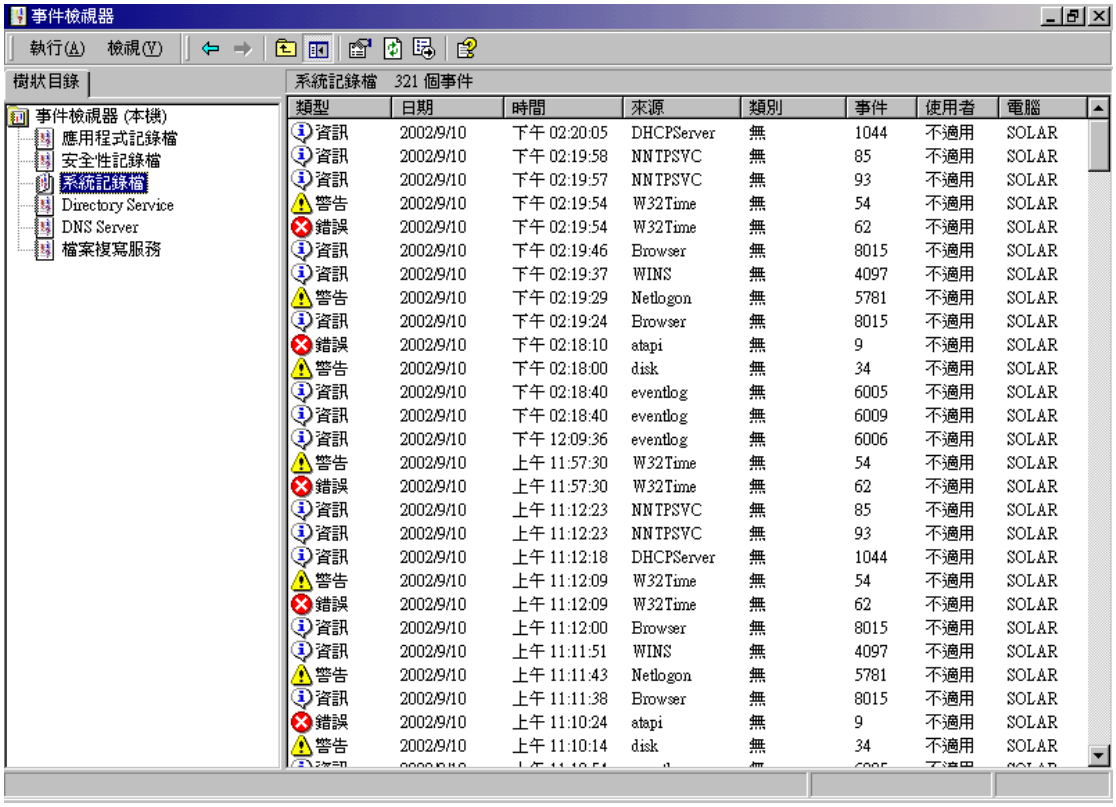
9.5.2 稽核紀錄檔

Windows 2000 用三種不同類型的紀錄檔來儲存事件，分別為：系統紀錄檔、應用程式紀錄檔、安全性紀錄檔，每個紀錄檔負責其相關的事件。

9.5.3 檢視稽核紀錄

在 Windows 2000 Server 中，稽核紀錄的檢視是透過事件檢視器來執行，要使用事件檢視器必須要有系統管理者的權限。

每個紀錄檔所包含的事件都是和紀錄檔類型相關的，系統紀錄檔包含系統元件的事件紀錄，例如：磁碟開啟錯誤或系統的開機、關機。安全性紀錄檔包含安全系統的變更、登入和登出、使用者權限的使用和安全性原則的變更，以及檔案和目錄的存取…等等事件。應用程式紀錄檔紀錄關於應用程式的事件，如：資料庫的錯誤或其他應用程式的錯誤。


事件檢視器
執行(A)
檢視(V)
樹狀目錄
事件檢視器 (本機)
應用程式記錄檔
安全性記錄檔
系統記錄檔
Directory Service
DNS Server
檔案複寫服務
系統記錄檔 321 個事件
類型 日期 時間 來源 類別 事件 使用者 電腦
資訊 2002/9/10 下午 02:20:05 DHCPServer 無 1044 不適用 SOLAR
資訊 2002/9/10 下午 02:19:58 NNTPSVC 無 85 不適用 SOLAR
資訊 2002/9/10 下午 02:19:57 NNTPSVC 無 93 不適用 SOLAR
警告 2002/9/10 下午 02:19:54 W32Time 無 54 不適用 SOLAR
錯誤 2002/9/10 下午 02:19:54 W32Time 無 62 不適用 SOLAR
資訊 2002/9/10 下午 02:19:46 Browser 無 8015 不適用 SOLAR
資訊 2002/9/10 下午 02:19:37 WINS 無 4097 不適用 SOLAR
警告 2002/9/10 下午 02:19:29 Netlogon 無 5781 不適用 SOLAR
資訊 2002/9/10 下午 02:19:24 Browser 無 8015 不適用 SOLAR
錯誤 2002/9/10 下午 02:18:10 atapi 無 9 不適用 SOLAR
警告 2002/9/10 下午 02:18:00 disk 無 34 不適用 SOLAR
資訊 2002/9/10 下午 02:18:40 eventlog 無 6005 不適用 SOLAR
資訊 2002/9/10 下午 02:18:40 eventlog 無 6009 不適用 SOLAR
資訊 2002/9/10 下午 12:09:36 eventlog 無 6006 不適用 SOLAR
警告 2002/9/10 上午 11:57:30 W32Time 無 54 不適用 SOLAR
錯誤 2002/9/10 上午 11:57:30 W32Time 無 62 不適用 SOLAR
資訊 2002/9/10 上午 11:12:23 NNTPSVC 無 85 不適用 SOLAR
資訊 2002/9/10 上午 11:12:23 NNTPSVC 無 93 不適用 SOLAR
資訊 2002/9/10 上午 11:12:18 DHCPServer 無 1044 不適用 SOLAR
警告 2002/9/10 上午 11:12:09 W32Time 無 54 不適用 SOLAR
錯誤 2002/9/10 上午 11:12:09 W32Time 無 62 不適用 SOLAR
資訊 2002/9/10 上午 11:12:00 Browser 無 8015 不適用 SOLAR
資訊 2002/9/10 上午 11:11:51 WINS 無 4097 不適用 SOLAR
警告 2002/9/10 上午 11:11:43 Netlogon 無 5781 不適用 SOLAR
資訊 2002/9/10 上午 11:11:38 Browser 無 8015 不適用 SOLAR
錯誤 2002/9/10 上午 11:10:24 atapi 無 9 不適用 SOLAR
警告 2002/9/10 上午 11:10:14 disk 無 34 不適用 SOLAR

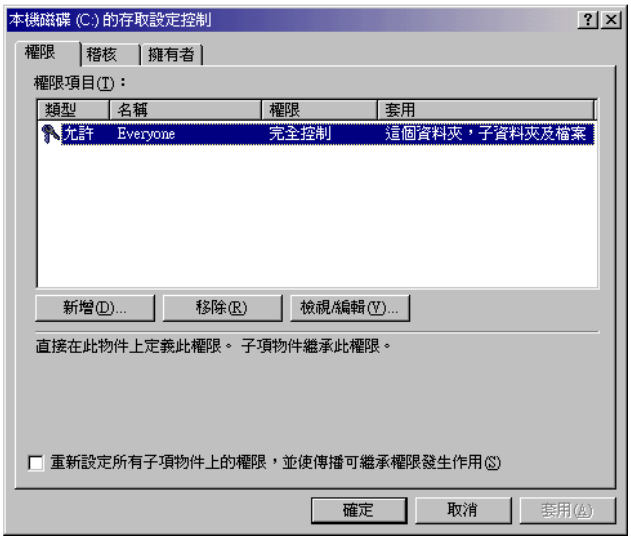
9.6 存取控制

9.6.1 簡介

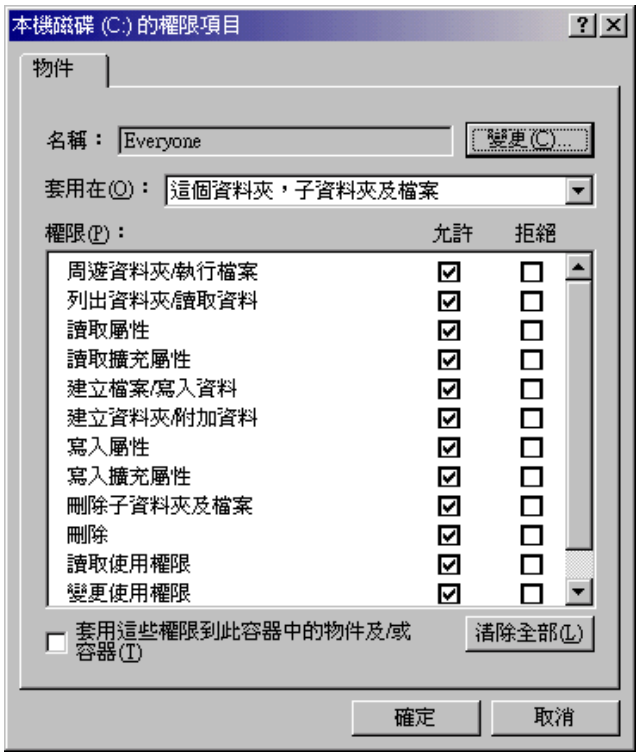
存取控制可以有委派能力和其他的功能，Active Directory 的存取控制模式類似一般的 Windows 2000 驗證模式且被整合到此模式中。在這個模式下，Active Directory 目錄服務會將使用者的存取權杖和所具備的權限向請求的目錄物件作安全描述的檢查，依據存取檢查的結果，Active Directory 目錄服務會給予存取或是拒絕存取。

每個安全描述包含了判別存取控制清單 (Discretionary Access Control List，DACL)和系統存取控制清單 (System Access Control List，SACL)。DACL 是一組的存取控制項 (Access Control Entities，ACE)，ACE 可以應用在每個擁有 SID (Security Identifier)的使用者、群組或電腦上，並且它的權限將決定使用者和群組是否允許或拒絕從事某特定動作。事實上，DACL 是為單一保護的項目或為目錄的整個部份定義出一個安全原則。

目錄服務需要一些方式來確認存取者的身分以有效的運用存取控制清單 (Access Control List，ACL)，因此，所有的目錄大多是透過陳述身分的方式來驗證使用者，假使使用者無法提出有效的身分證明，Active Dirtectory 目錄服務便會將使用者視為匿名身分，給予最低的一般權限。



系統存取控制清單 (SACL)允許管理人員透過Windows 2000 事件檢視器的安全紀錄，稽核使用者到物件的存取和瀏覽這些與安全性相關的事件。當存取一個物件時，SACL 可以指定稽核某群組或使用者帳戶、存取事件和每個存取事件成功或失敗的屬性。Active diretory 目錄服務的 SACL 可以引導作業系統對物件的任何屬性進行稽核的讀取或寫入動作，或者只對單一屬性進行讀取或寫入動作的稽核。



9.6.2 存取權限的繼承

在 Active Directory 網域內，子物件會從上層物件如組織單位 (OU)、容器、或網域物件本身中繼承存取控制項 (ACE)。在一個網域內，物件的繼承性會把一個物件所得到的 ACE 或所有屬性所得到的 ACE 散播到此物件下的所有物件中。網域管理人員可以在組織單位使 ACE 取得建立和刪除子物件權限，以達到部門的委派控制管理。

繼承性也是以物件為基礎的，預設的情況下，所有的 Active Directory 物件會從父系物件中繼承 ACE，但是網域管理人員仍可以針對某些物件選擇關閉繼

承性。


測試資料夾 內容
一般
Web 共用
共用
安全
名稱
Everyone
新增(D)...
移除(R)
權限(P)：
允許
拒絕
完全控制
修改
讀取及執行
清單資料夾內容
讀取
寫入
進階(V)...
允許來自父項的可繼承權限可以傳播至此物件(H)
確定
取消
套用(A)