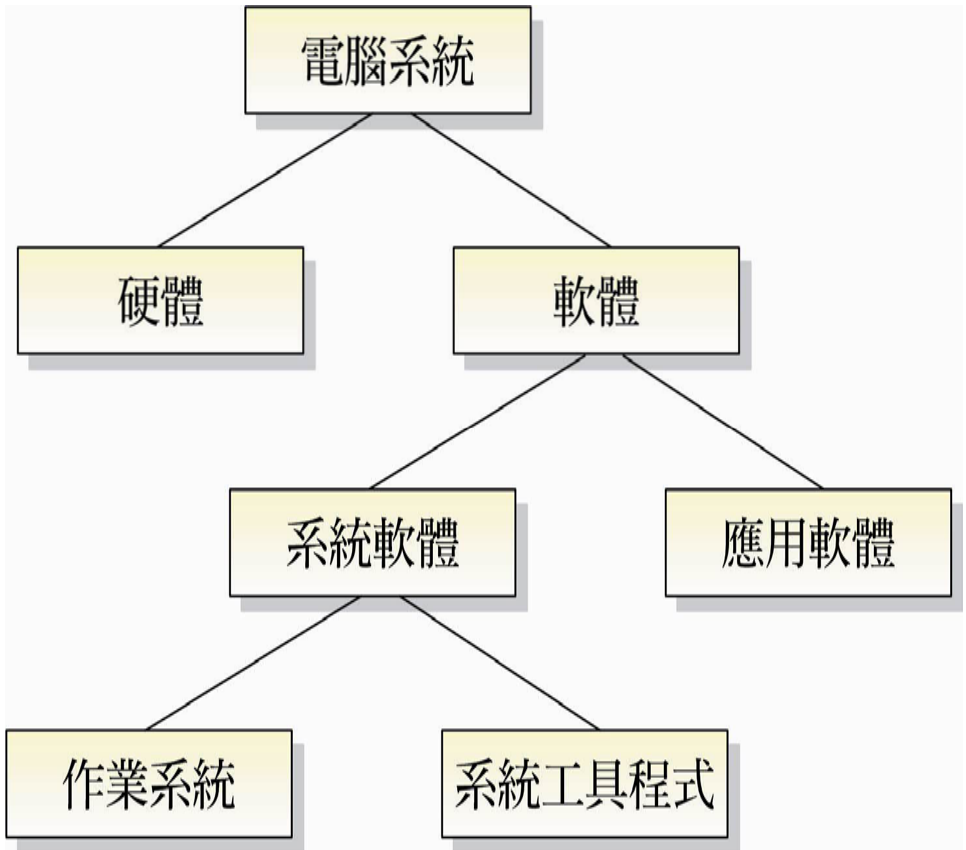
第5章 作業系統

內容

- 作業系統簡介
- 作業系統的演進
- 記憶體管理
- 處理程序管理元件
- 磁碟管理元件
- 常見的作業系統

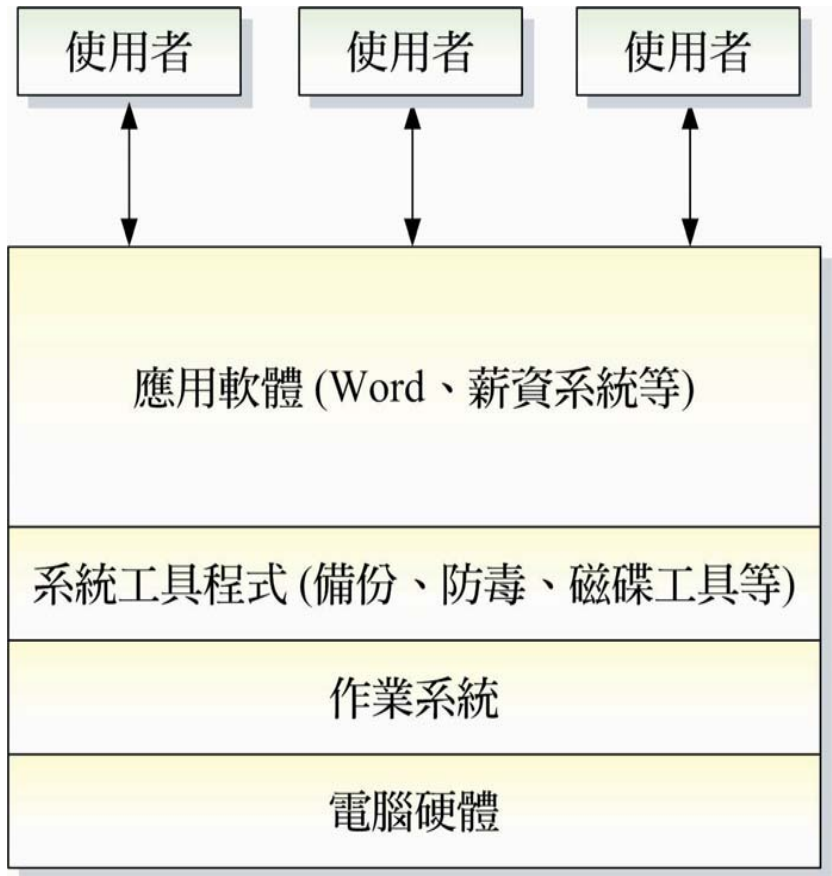
5-1 作業系統簡介

- 電腦系統：
  - 硬體
  - 軟體
- 軟體：
  - 應用軟體
  - 系統軟體
- 系統軟體：
  - 作業系統
  - 系統工具程式



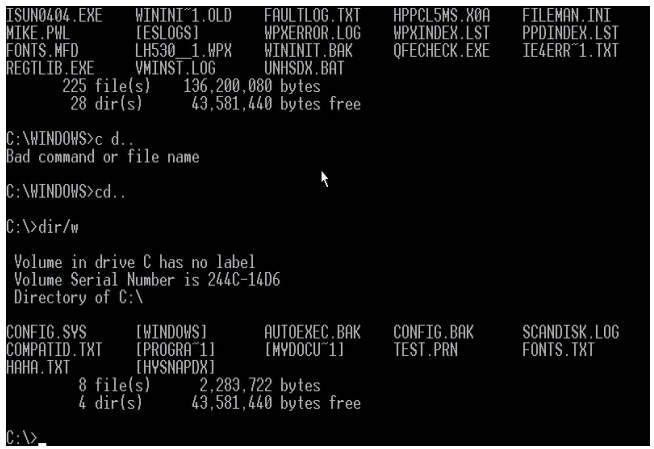
作業系統的角色

- 作業系統是每個電腦系統最重要的部份，它介於使用者與電腦硬體之間，負責處理程式的執行與管理各種軟硬體資源的存取
- 電腦系統中最接近使用者的是應用程式，而作業系統是一組介於應用程式和硬體之間的程式



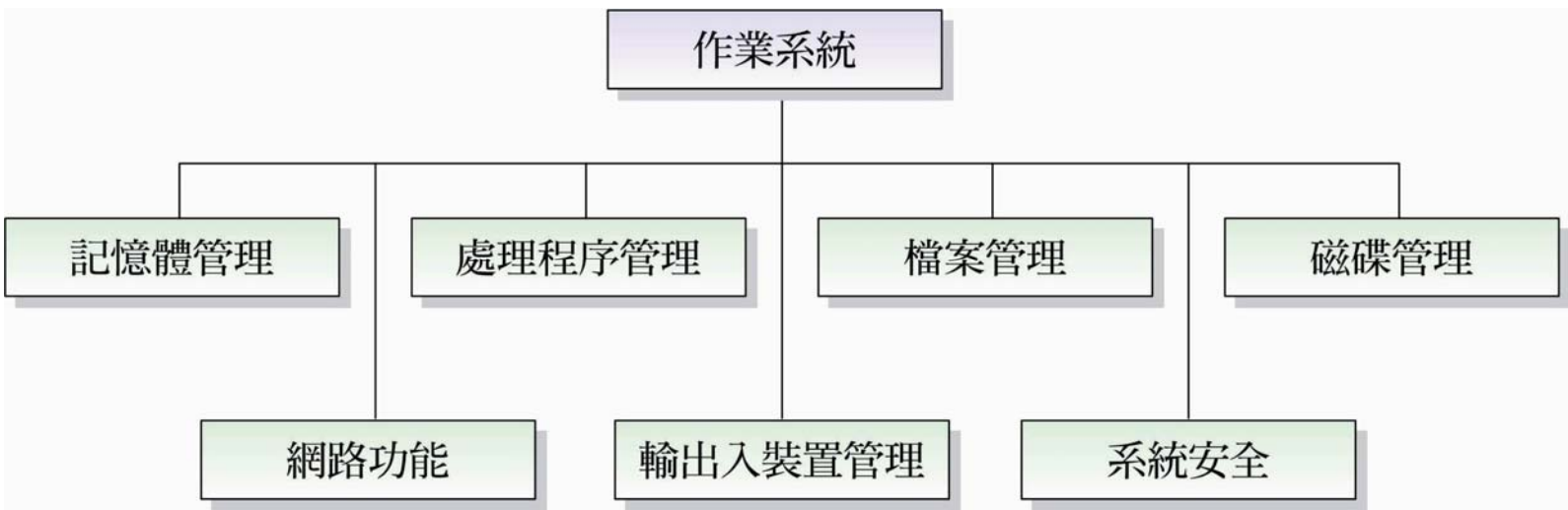
作業系統的功能

- 管理電腦系統中的各種資源的使用與溝通
- 提供使用者介面
  - 命令列
  - 圖形化介面 (GUI)
- 執行應用軟體並提供服務



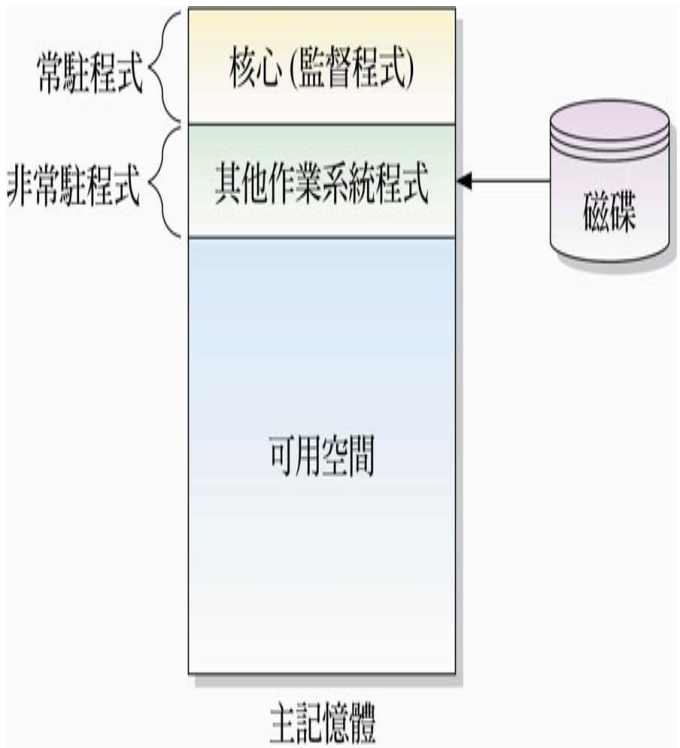
作業系統的元件

- 記憶體管理元件
- 處理程序管理元件
- 檔案管理元件
- 磁碟管理元件
- 輸出入裝置管理元件
- 網路元件
- 系統安全元件



作業系統的啟動方式

- 常駐程式：即作業系統的核心(或稱監督程式)
  - 在整個運作過程中它們都會一直在記憶體中
  - 保持越小越好
  - 控制了全部的系統操作
- 非常駐程式：作業系統裡除核心以外的部分
  - 是由核心負責將它從磁碟載入到記憶體中

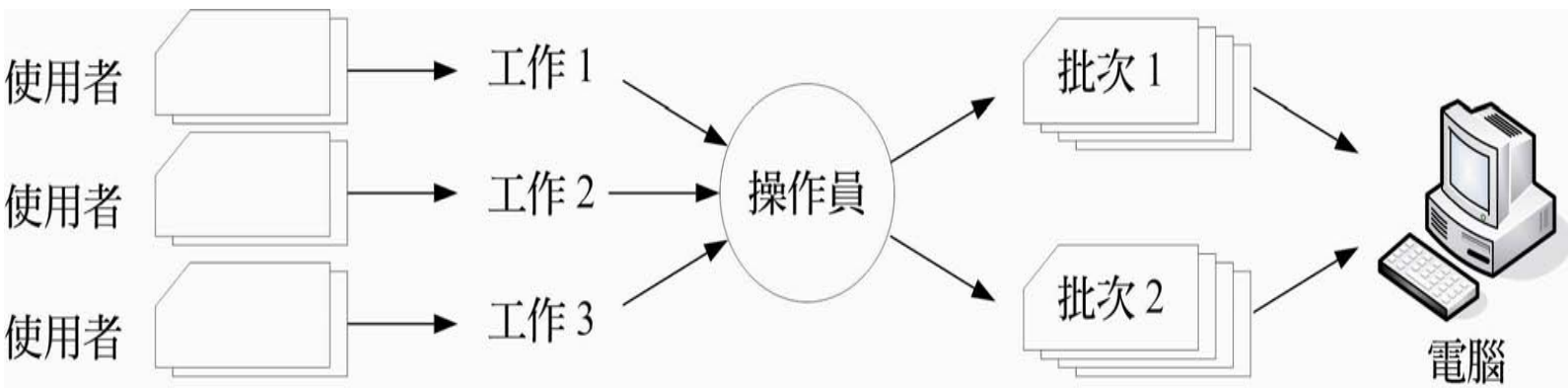


5-2 作業系統的演進

- 批次處理系統
- 多元程式處理
- 分時系統(多工系統)
- 多處理器系統
- 分散式系統
- 個人電腦作業系統
- 叢集式系統
- 即時系統
- 手持式系統

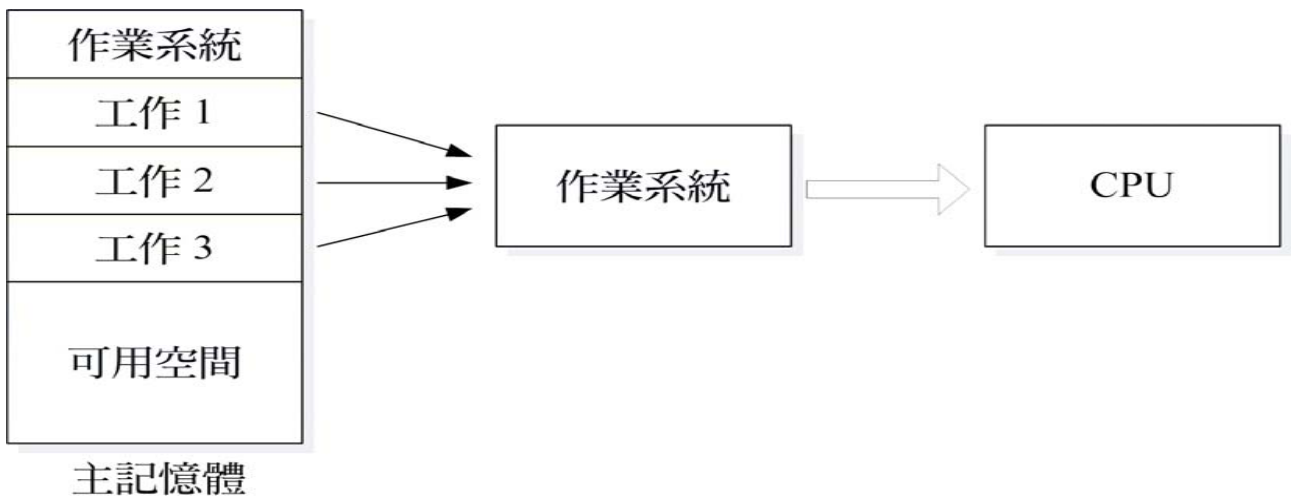
批次處理系統

- 輸入是一疊打孔卡片
- 送入電腦後解讀成指令和資料
- 所有要執行的打孔卡片交給操作員，由操作員整理後整批送進電腦來處理，因此稱為批次處理
- 每個執行的程式稱為工作 (job)，而一起送進電腦執行的多個工作統稱為一個批次 (batch)



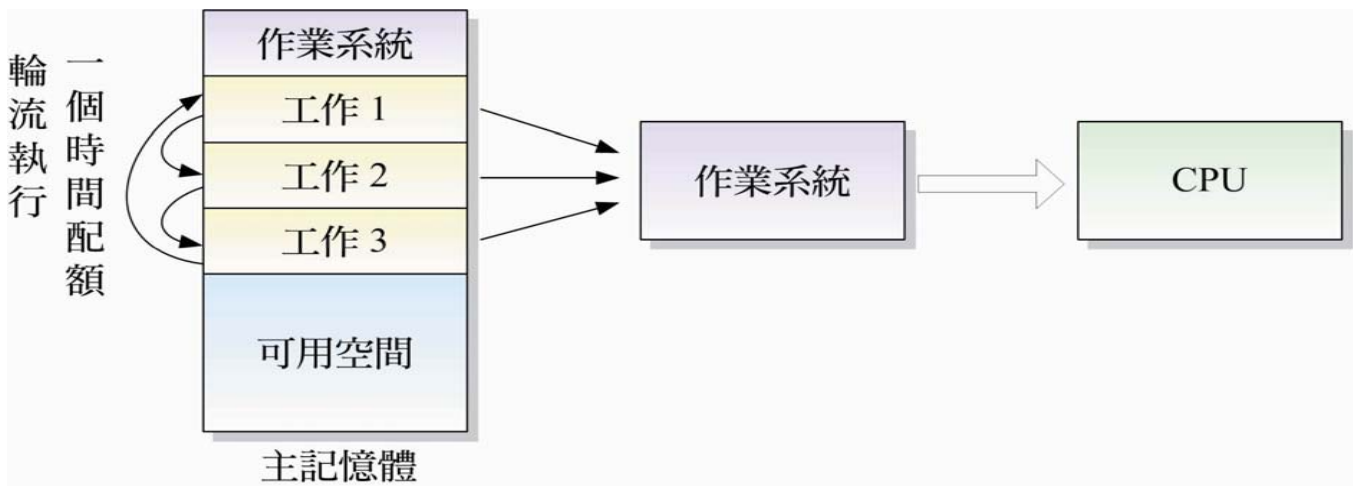
多元程式處理

- 多元程式處理 (multiprogramming) 會在記憶體裡同時存放著多份工作，讓這些程式依照作業系統安排的順序來爭取CPU時間
- 工作排程功能是負責從等待進入記憶體的眾多工作之間，挑選出下一個可以被載入記憶體的工作



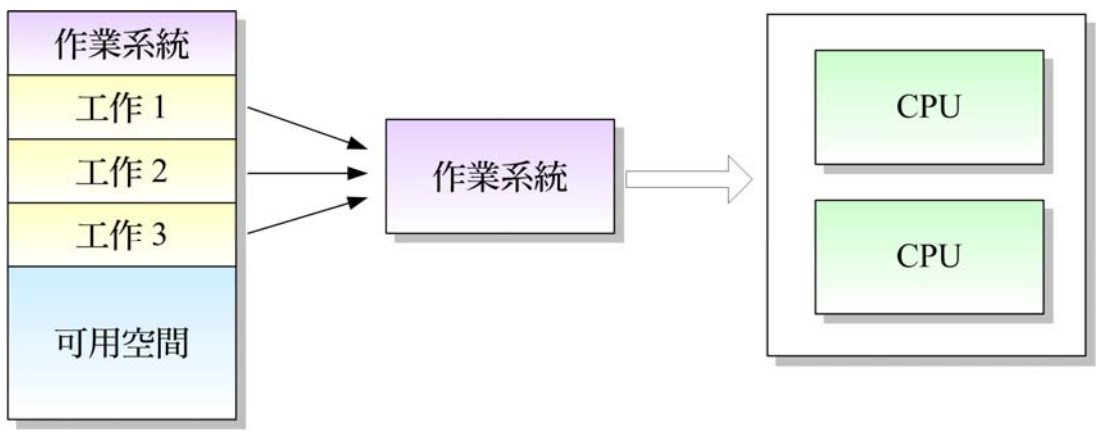
分時系統(多工系統)

- 分時系統(或稱多工系統)是把CPU時間切割成許多小段，稱為時間配額或時間片段(time slice)，輪流分配給每個使用者的每個工作
- 它是一種特殊形式的多元程式處理作業系統，也常被稱為交談式(interactive)電腦系統



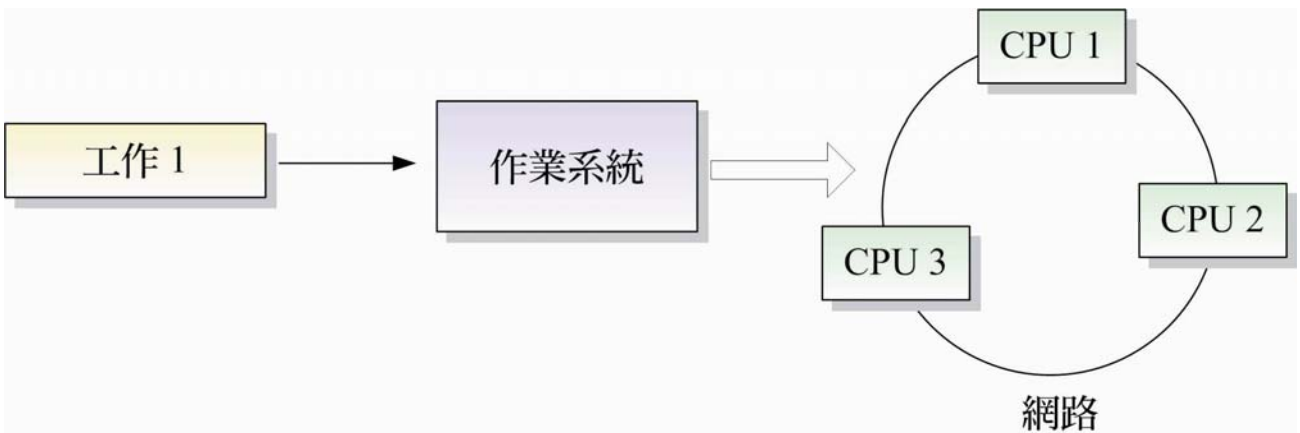
多處理器系統

- 單處理器系統：只有一顆CPU
- 多處理器系統
  - 也稱為平行系統或是緊密耦合系統
  - 擁有一個以上的CPU，這些CPU之間彼此緊密的相互聯繫，而且共用系統時脈、匯流排、記憶體，甚至是週邊裝置
  - 可容納的工作量變大
  - 提高系統的可靠度
  - 作業系統會更加複雜
  - 使用對稱式多元處理 (SMP) 模式或非對稱式多元處理模式



分散式系統

- 將工作拆成幾部分，透過快速的網路通訊指派給多台電腦分別完成
- 電腦的實體位置可以相距遙遠，甚至是透過Internet來溝通
- 資源也可以分散指派
- 比一般的作業系統更複雜
- 安全性方面的控管也要更加小心



其他種類的作業系統

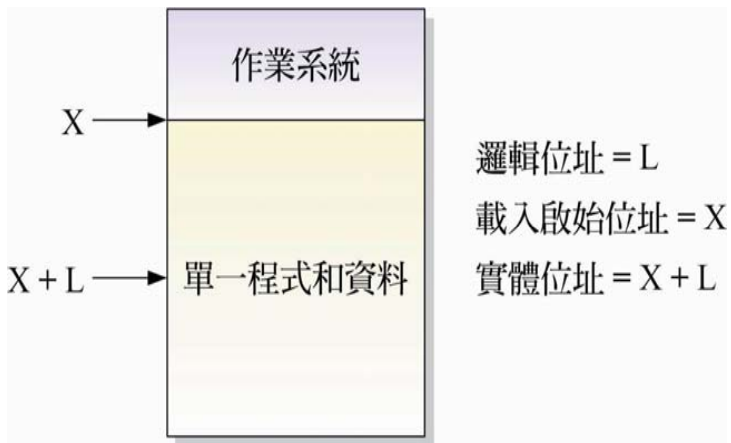
- 個人電腦作業系統：比較輕巧而容易學習和使用的作業系統
- 叢集式系統：集合數台個別的系統一起完成計算工作
- 即時系統：在運作過程中必須要隨時考慮到時效性
- 手持式系統：其硬體的記憶體比PC小很多，處理器速度也只是PC的幾分之一，螢幕尺寸也縮小

5-3 記憶體管理

- 邏輯位址：程式中指定的位址，相對於程式本身而非主記憶體
  - 又稱為虛擬位址或相對位址
- 實體位址：指的是實際上在主記憶體內的實體位置
- 位址鏈結：將程式中的每個邏輯位址轉換成實體位址

單一程式處理

- 主記憶體固定分成兩部份：
  - 部份記憶體存放作業系統
  - 其餘空間放置一個正在執行中的程式和資料
- 優點：實作和管理都很簡單
- 缺點：
  - 程式大小一定要在記憶體裡放得下才能執行
  - 其他程式就無法同時執行

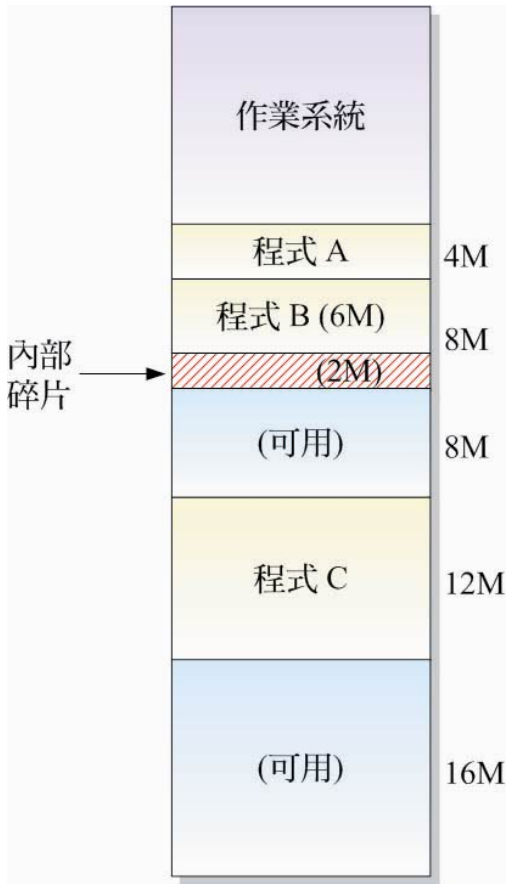


多元程式處理

- 在同一時間內安排多個程式在記憶體中輪流使用CPU
- 分割法 (partitioning)
- 分頁法 (paging)
- 分段法 (segmentation)
- 虛擬記憶體 (virtual memory)
- 需求分頁法 (demand paging)
- 需求分段法 (demand segmentation)
- 分頁式的分段法

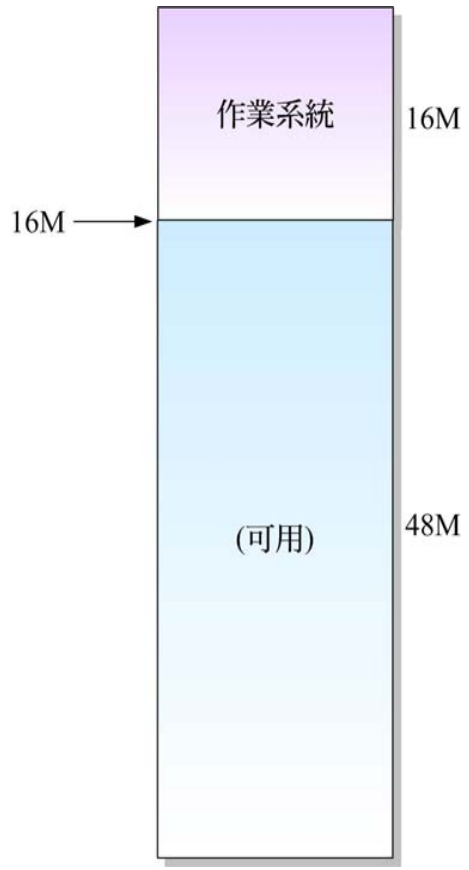
分割法

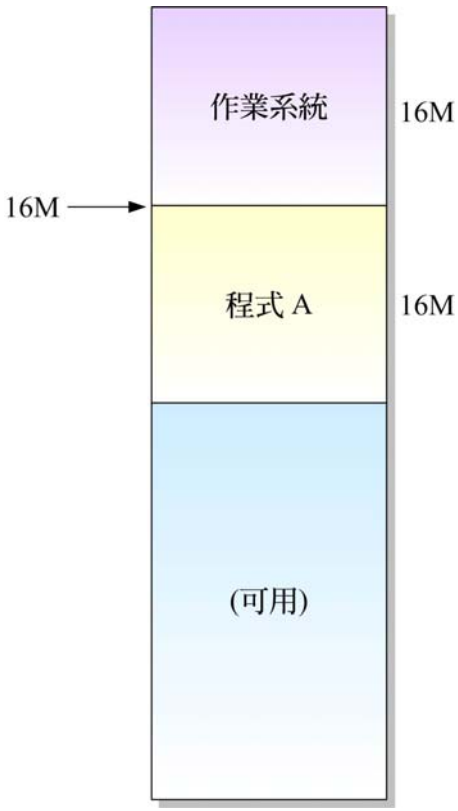
- 將主記憶體切分成多個區塊，稱為分割區 (partition)
- 固定分割法：分割成固定大小的分割區
  - 作業系統使用表格記錄分割區的起始位址與長度
  - 每個分割區只能載入一個程式
  - CPU輪流執行這些程式
  - 分割區中剩下沒有用到的空間就叫做內部碎片



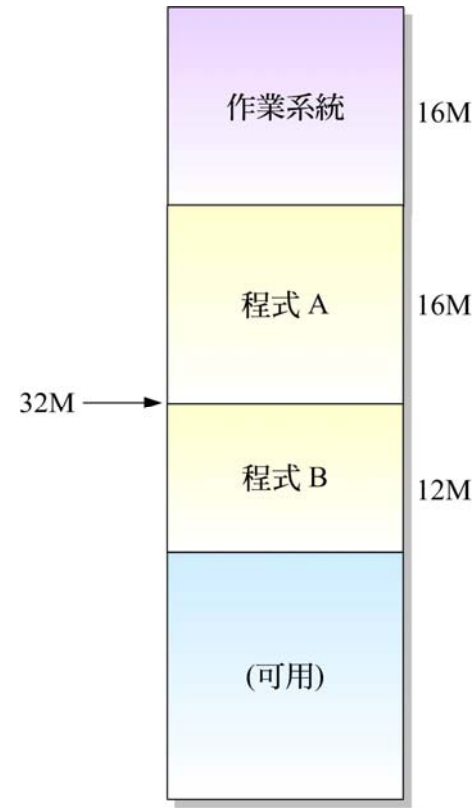
動態分割法

- 是為了改善內部碎片的問題而開發出的技術
- 根據程式大小劃分剛好大小的分割區指派給它
- 同樣使用表格來記錄分割區的位址和長度
- 範例：一開始主記憶體裡只有作業系統，沒有載入任何程式
  - 作業系統：大小為16 M
  - 可用空間：大小為48 M，起始位址為16 M

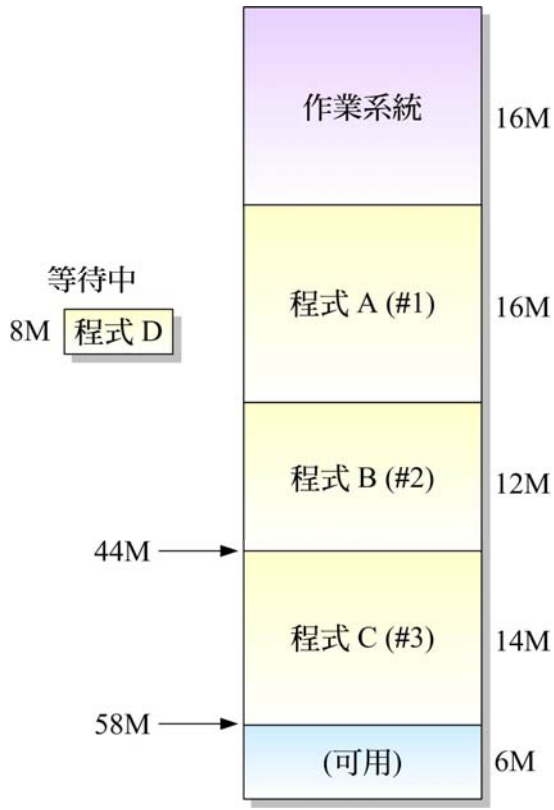




- 載入程式A (大小16 M)，在分割區表格中記錄分割區#1的起始位址為16 M，大小為16 M
- 載入程式B (大小12M)，在分割區表格中記錄分割區#2的起始位址為32 M，大小為12 M



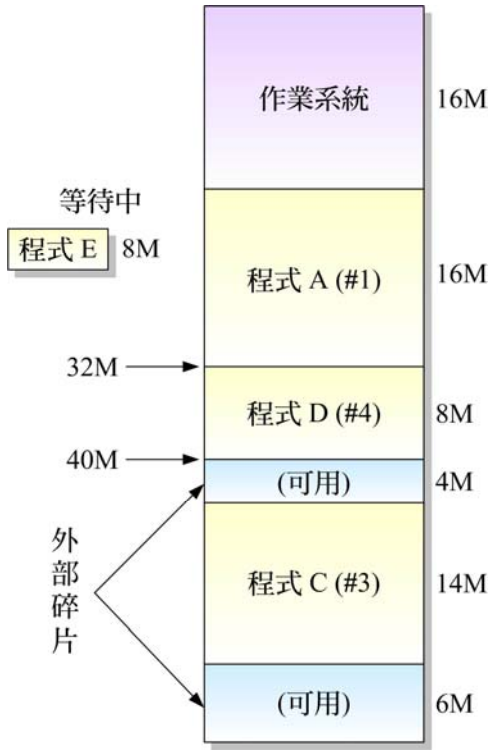
- 載入程式C (14 M)，在分割區表格中記錄分割區#3的起始位址為44 M，大小為14 M。此時主記憶體的可用空間：
  - 大小：6 M
  - 起始位址：58 M
- 接下來要載入程式D (8M)，因為太大而無法載入，必須等待之前的程式執行完畢釋出記憶體



已使用空間分割表

	起始位址	長度
#1	16M	16M
#2	32M	12M
#3	44M	14M

- 現在程式B執行完畢，釋出起始位址為32 M，大小為12 M的一塊記憶體
- 接下來作業系統劃分出一塊8 M的分割區給D程式，同時在分割區表格中記錄分割區#4的起始位址為32 M，大小為8 M
- 結果多出一塊起始位址為40 M，大小為4 M的可用空間
- 接下來要載入程式E (8M)，雖然目前有兩塊各為4 M和6 M的可用空間，但是兩塊不連續，因此無法使用，程式E必須繼續等待



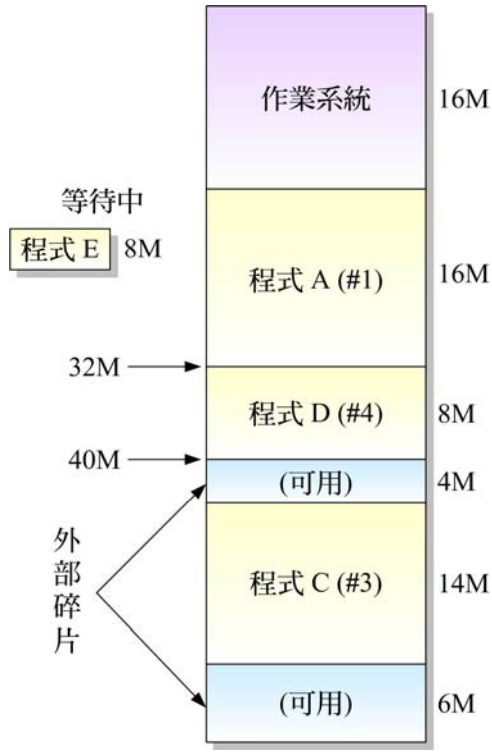
已使用空間分割表

	起始位址	長度
#1	16M	16M
#3	44M	14M
#4	32M	8M
⋮	⋮	⋮
⋮	⋮	⋮

可用空間分割表

	起始位址	長度
#1	58M	6M
#2	40M	4M

- 當可用空間的總大小足以容納新程式，但是卻因為空間不連續而個別空間都太小，導致無法利用，此時這些可用空間就稱為外部碎片



已使用空間分割表

	起始位址	長度
#1	16M	16M
#3	44M	14M
#4	32M	8M
⋮	⋮	⋮
⋮	⋮	⋮

可用空間分割表

	起始位址	長度
#1	58M	6M
#2	40M	4M

- 無論是固定或動態的分割法，特點是程式必須整個能放進記憶體才能執行，而且必須是連續空間
- 分割法可提昇CPU的使用率，但是有以下幾個問題存在：
  - 分割區的大小是在程式執行之前就必須先決定
  - 若太小，就有許多程式會無法載入執行
  - 若太大，就會出現很多內部或外部碎片
  - 分割區碎片的問題會隨著時間過去而越來越嚴重，導致CPU和記憶體空間的利用率降低
  - 當外部碎片很多時，記憶體管理元件就會進行聚集(compaction) 動作，把分割區碎片都聚集在一起，變成一大塊空間，但是這個動作需耗費額外的CPU時間，而且必須等待適當的時機

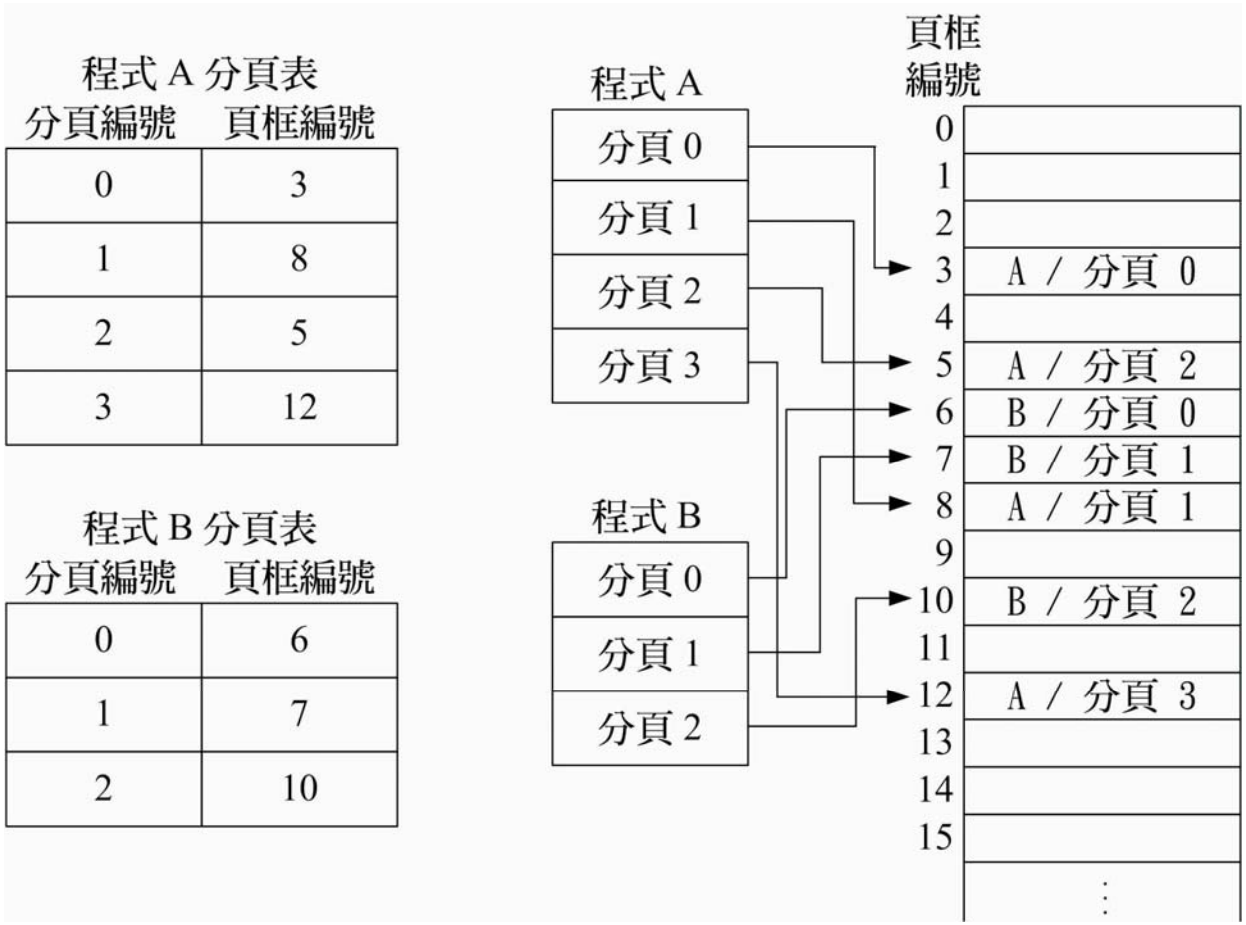
選擇分割區的策略

- 最先配合 (first fit)：將第一個可以容納得下的分割區指派給此程式
  - 優點：下決定的速度最快
- 最佳配合 (best fit)：從所有可以容納得下此程式的分割區中選出最小的指派給它
  - 優點：浪費掉的分割區碎片最小
  - 缺點：因為碎片太小，因此日後很難讓其他程式來使用，結果會造成碎片越來越多
- 最差配合 (worst fit)：從所有可以容納得下此程式的分割區中選出最大的指派給它
  - 優點：剩下的分割區碎片是最大的，這樣就很有機會讓其他程式來使用，反而不會浪費掉

分頁法

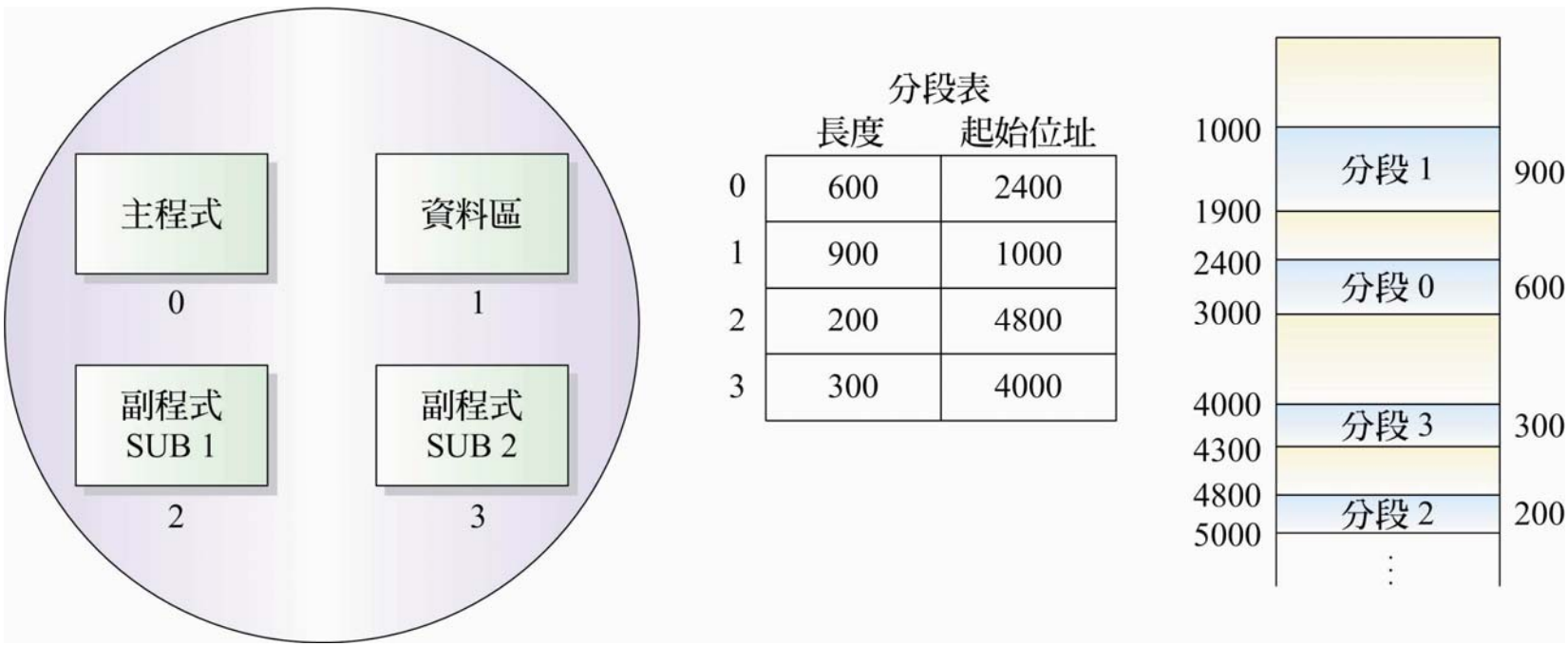
- 外部碎片是無法避免的，只是程度的輕重不同
- 後來開發的技術重點，是允許同一個程式可以載入到不連續的記憶體空間中，只要可用空間的總大小足夠，就可以執行此程式
- 分頁法
  - 頁框 (frame)：將實體記憶體分割成相同大小的區塊
  - 分頁 (page)：將執行程式的邏輯記憶體也分割成相同大小的區塊

– 分頁表：或稱分頁對映表，負責記錄每個程式的每個分頁是載入到主記憶體的哪一個頁框中



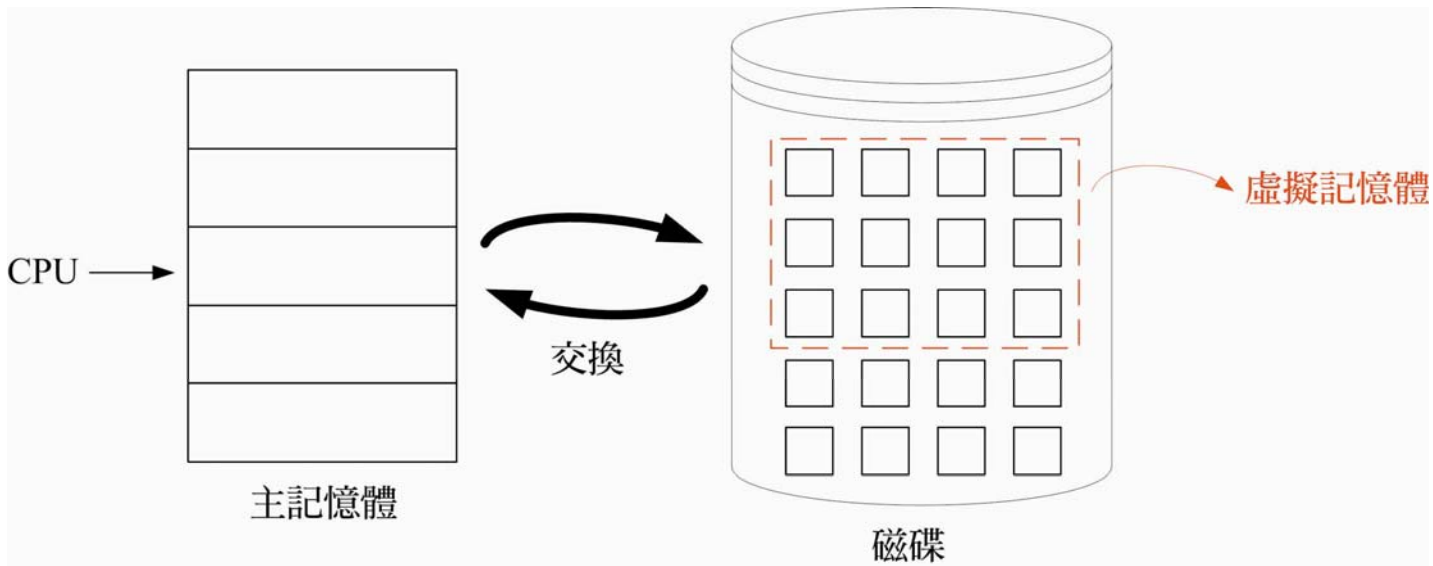
分段法

- 分段法是把程式的邏輯記憶體分割成許多分段(segment)，並記錄每一分段的編號和長度。並透過所謂的分段表 (segment table) 記錄每一個分段是對應到主記憶體的哪個位置



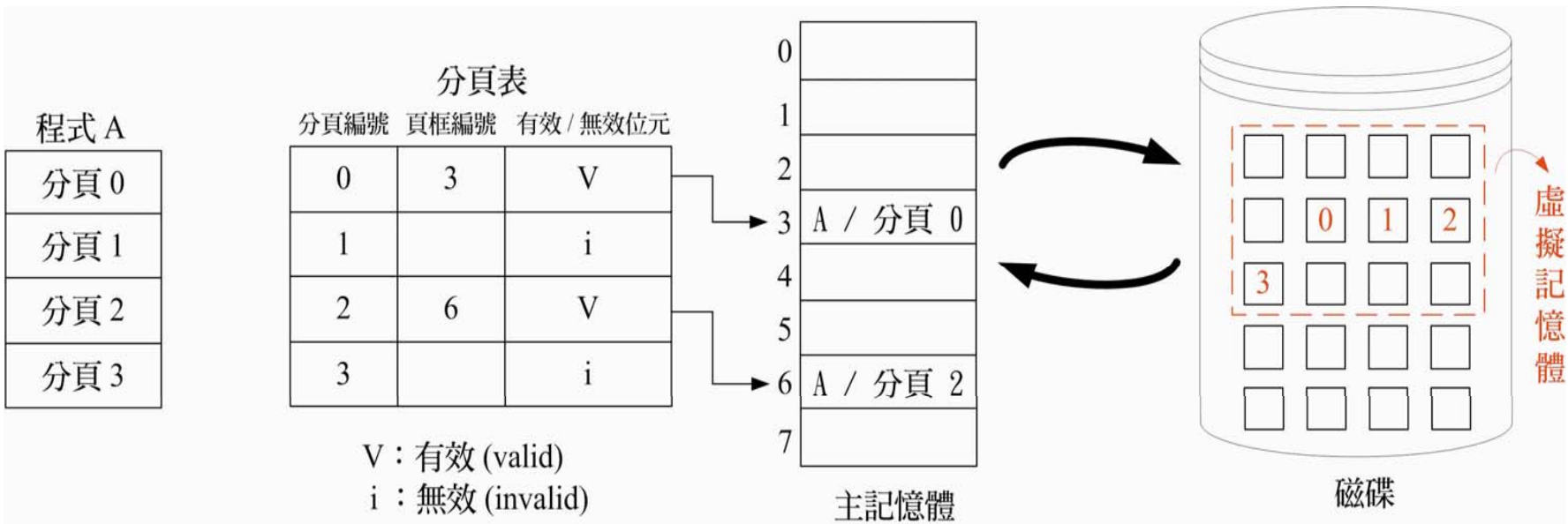
虛擬記憶體

- 一種允許程式不必整個都在記憶體中，而仍然可以執行的技術
- 優點：可載入更多個程式到記憶體中，因此記憶體的利用率與CPU的使用率都會提高



需求分頁法

- 將分頁法配合虛擬記憶體技術而改良開發
- 與分頁法類似，不同點在於程式不必全部載入，只要記錄下有哪些分頁是真的有載入主記憶體中，而其餘哪些分頁還留在磁碟上即可



需求分段法

- 將分段法配合虛擬記憶體技術而改良開發。
- 程式不需要載入全部的分段就可以開始執行。
- 在執行中若查到某個分段是記錄成無效的，那麼表示這個分段不在主記憶體中，必須到磁碟去讀取。

分頁式的分段法

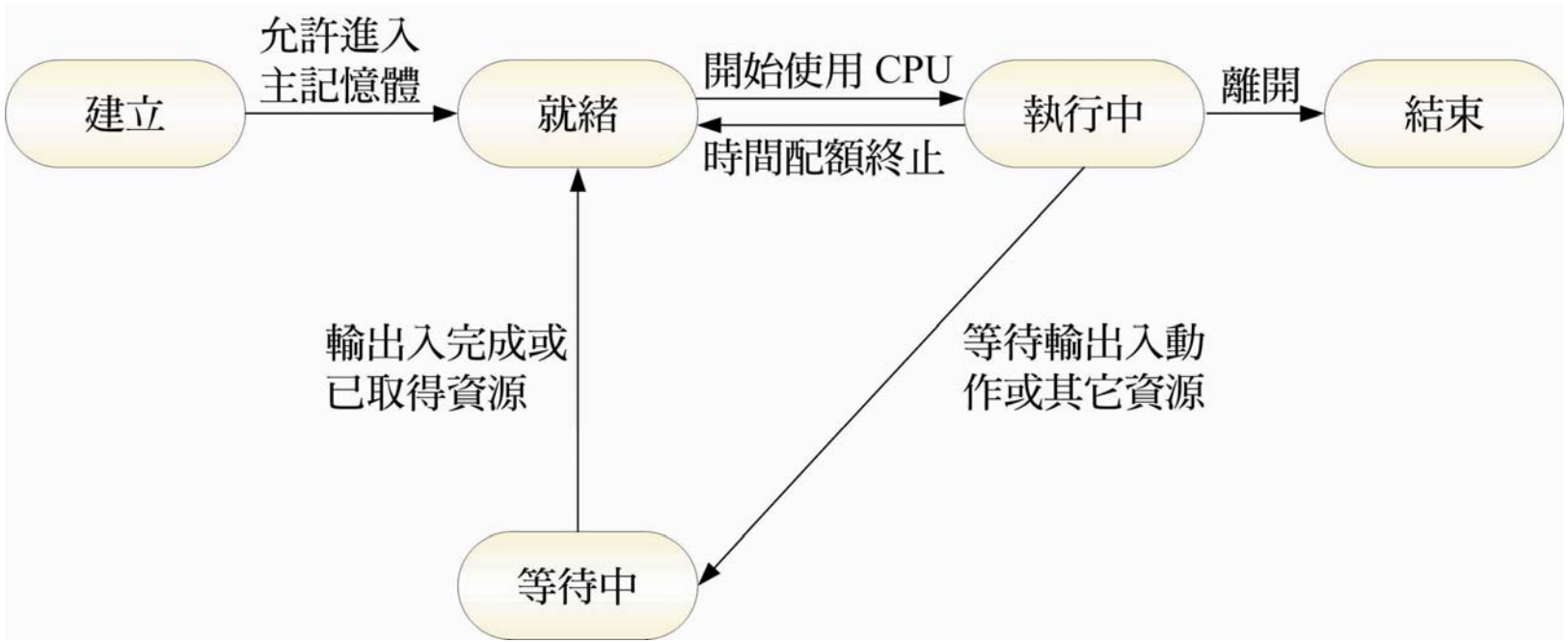
- 若分段太大，可以把分段再分割成多個分頁。
- 實體記憶體也分割成頁框，再搭配虛擬記憶體技術，即可在需要時才分批載入各個分頁 。

5-4 處理程序管理元件

- 處理程序 (process) 或簡稱程序：正在執行中的程式
- 程式是一組靜態的指令，而程序則是程式在執行時的動態實體
- 處理程序管理元件：作業系統的元件之一，負責記錄每個程序的進度，以及它目前的計算成果和狀態

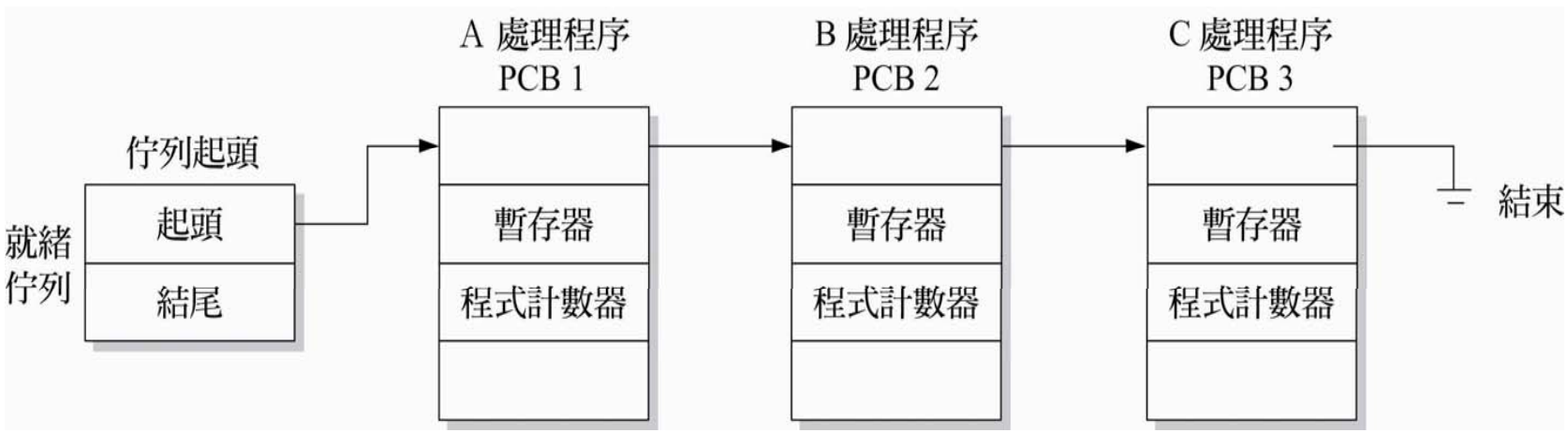
程序狀態圖

- 在電腦系統中，每個程序都會經過以下5種狀態：(1) 建立 (new)，(2) 就緒 (ready)，(3) 執行中 (running)，(4) 等待中 (waiting)，(5) 結束 (terminated)
- 程序狀態圖 (process state diagram) 則是用來表示這些狀態彼此之間的關係



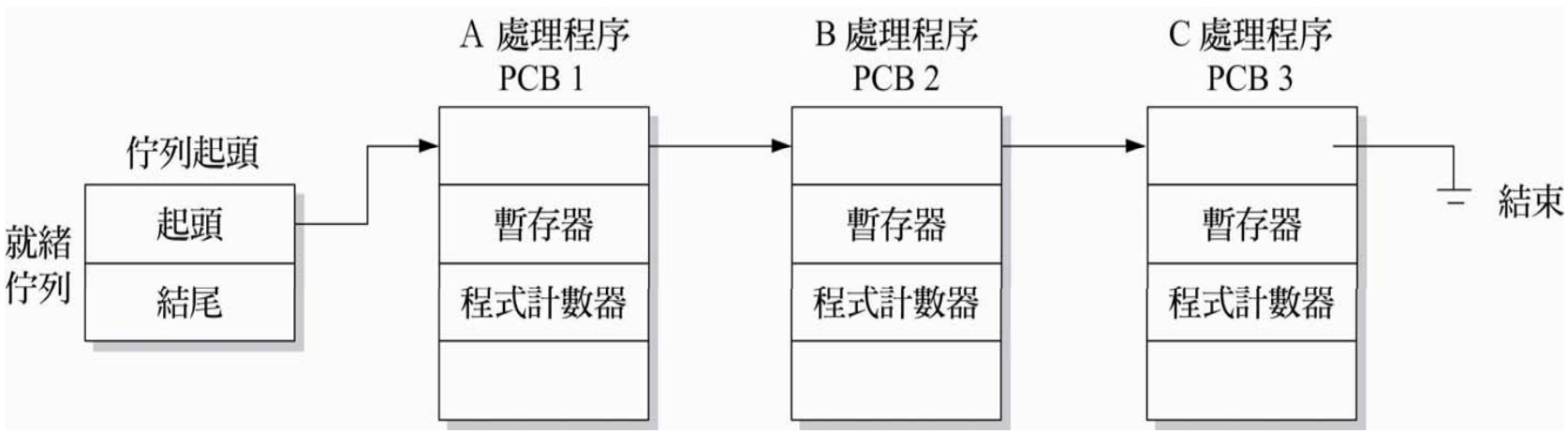
程序控制區塊 (PCB)

- 為了要管理各種不同狀態的程序，作業系統必須為每一個程序儲存許多資訊，通常是儲存在“處理程序控制區塊 (process control block)” 的資料結構中



程序控制區塊 (PCB)

- 作業系統在建立新程序時，同時就會為它建立一個新的PCB，此後將不斷的維護這個PCB直到這個程序結束為止
- 就緒佇列 ：所有在「就緒」狀態等待的程序所形成

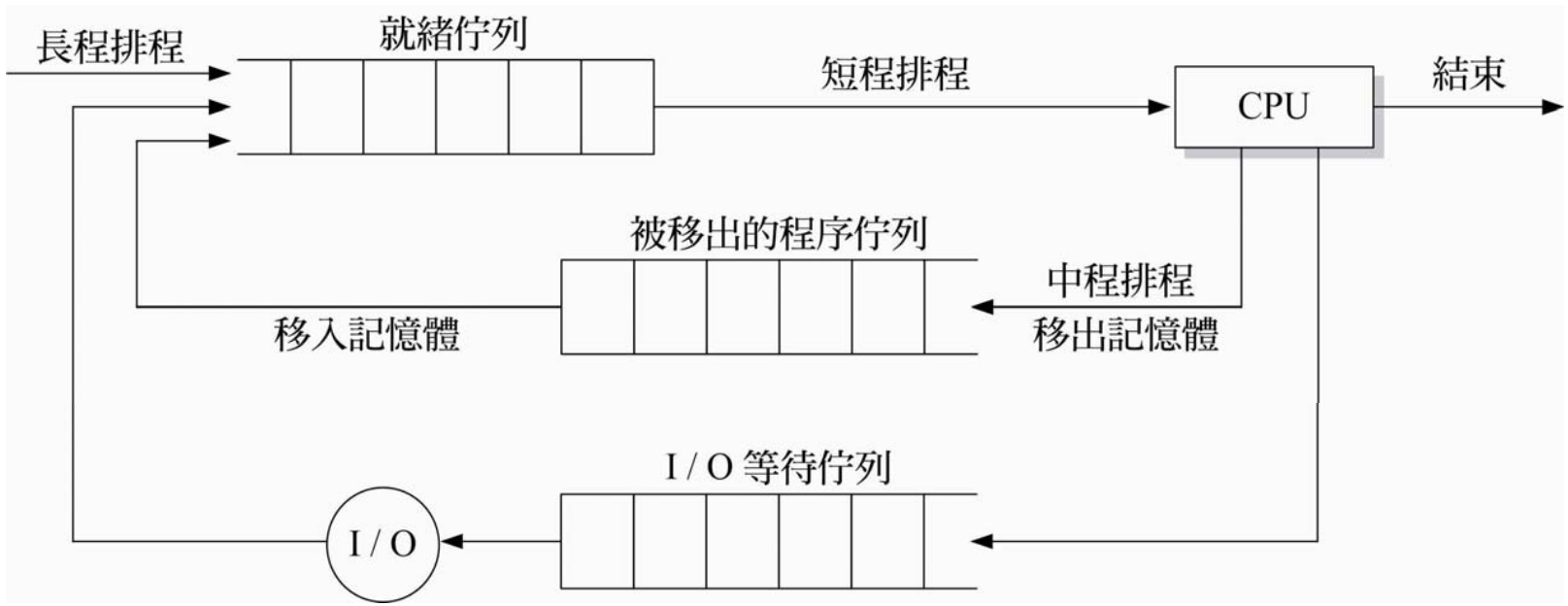


多執行緒

- 近代的作業系統常把程序更進一步切分成多個所謂的執行緒 (thread)，每個執行緒有自己的控制流程，但共用同一塊程式碼區域、資料區和作業系統資源
- 多執行緒 (multithreading)：一種新的多工方式，允許電腦在單一程式中執行一個以上的工作
- 能夠執行多執行緒應用程式的PC作業系統：包括Linux、Mac OS 8及以後版本，以及在Windows 95之後的所有Microsoft Windows版本

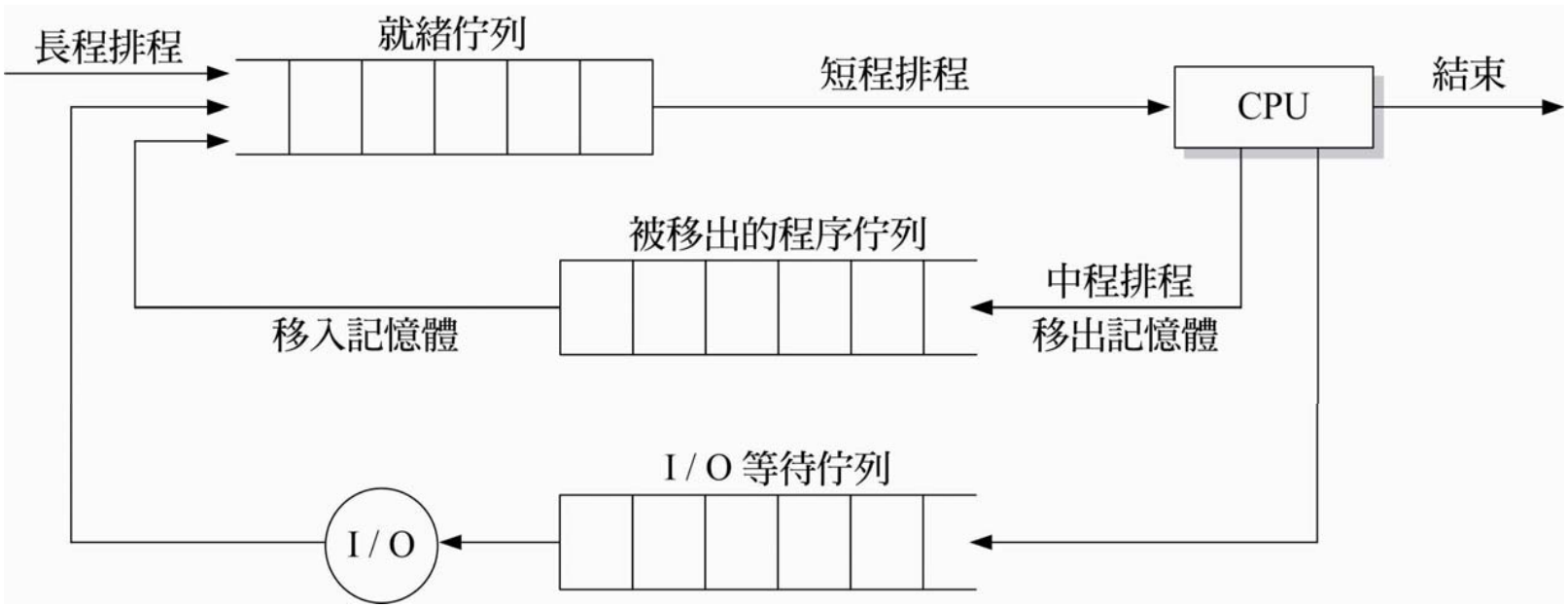
排程程式的分類

- 長程排程程式：負責選擇和安排可載入主記憶體的程序
- 短程排程程式：負責從主記憶體中選擇某個程序，將CPU使用權交給它，這個程序也就從「就緒」狀態進入了「執行中」狀態，又稱為CPU排程程式



排程程式的分類

- 中程排程程式負責決定是否要將程序暫時移出主記憶體，它是介於長程與短程排程之間的排程程式



CPU排程的演算法

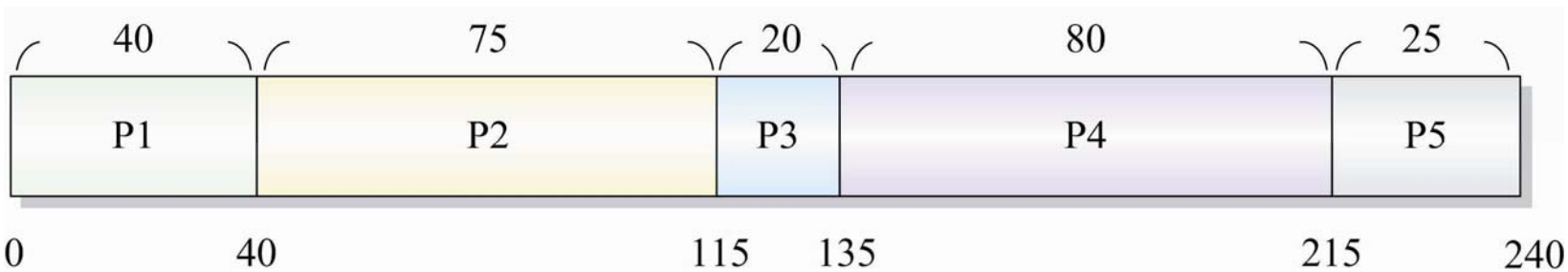
- 先來先做 (First-Come，First-Served，簡稱FCFS)
- 最短的工作先做 (Shortest Job First，簡稱SJF)
- 優先權 (Priority)
- 循環分配 (Round Robin，簡稱RR)

FCFS演算法

- 以程序到達的先後順序來執行，先來的先做
- 甘特圖：用來表示每個程序的執行次序和時間軸

程序	所需時間單位
P1	40
P2	75
P3	20
P4	80
P5	25

- 平均的回復時間 = (40 + 115 + 135 + 215 + 240) / 5，等於149

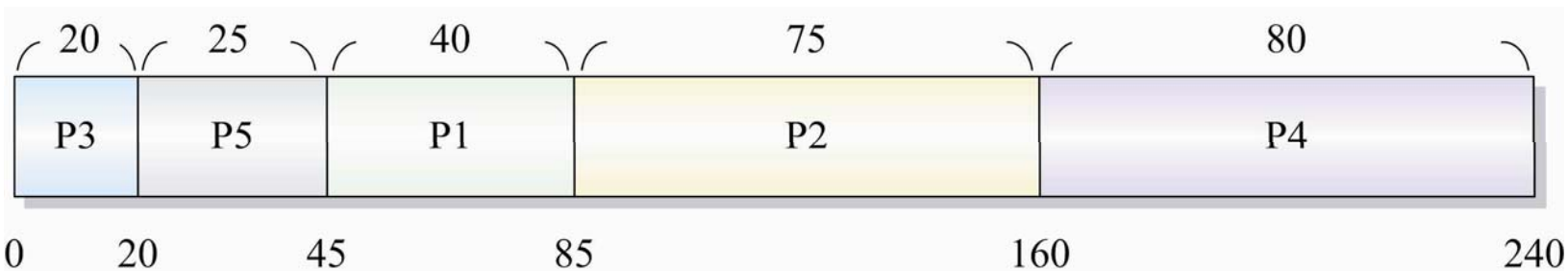


SJF演算法

- 先看一遍在就緒狀態下的所有程序，將它們的所需執行時間從小排到大，然後從時間最短的開始執行
- 執行順序：P3、P5、P1、P2 、 P4

程序	所需時間單位
P1	40
P2	75
P3	20
P4	80
P5	25

- 平均的回復時間 = (20 + 45 + 85 + 160 + 240) / 5，等於110



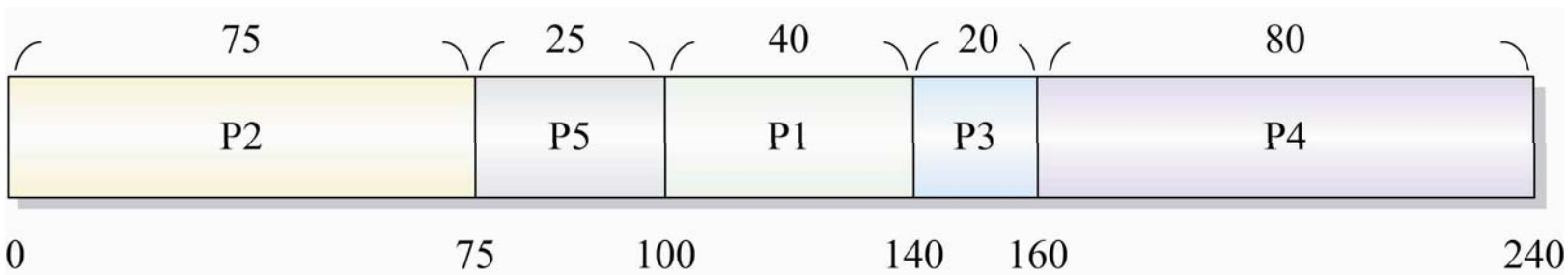
優先權演算法

- 依照事先決定好的優先權定義，計算出每個程序的優先順序，然後照著優先順序的先後給予CPU使用權

- 執行順序：P2、P5、P1、P3 、 P4

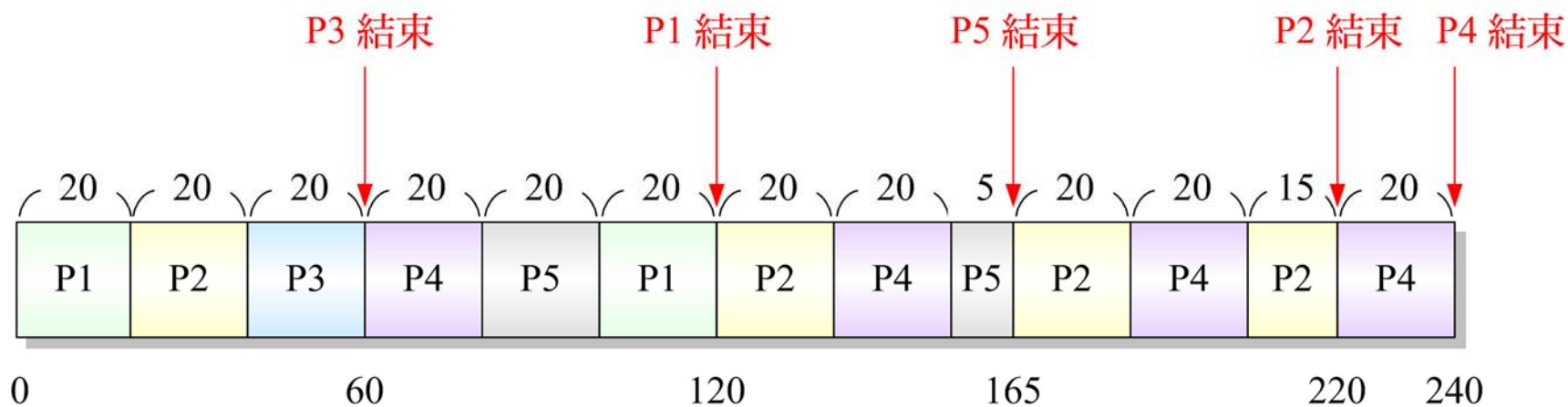
程序	所需時間單位	優先順序
P1	40	3
P2	75	1
P3	20	4
P4	80	5
P5	25	2

- 平均的回復時間 = (75 + 100 + 140 + 160 + 240) / 5，等於143



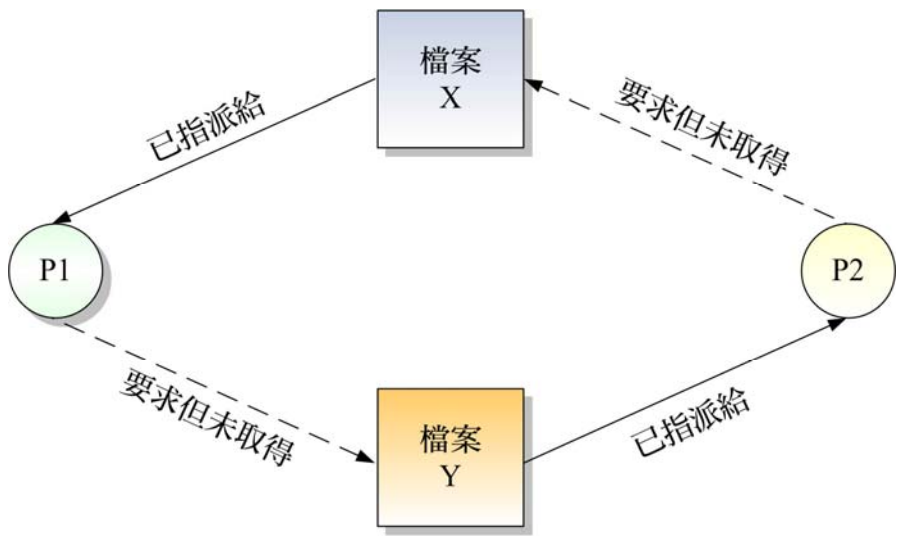
RR演算法

- 事先定義好固定的時間配額，讓每個程序輪流分配一個時間配額。假如不夠讓程序執行完成，下一次就再分配一個時間配額給它
- 平均回復時間 = (60 + 120 + 165 + 220 + 240) / 5，等於161

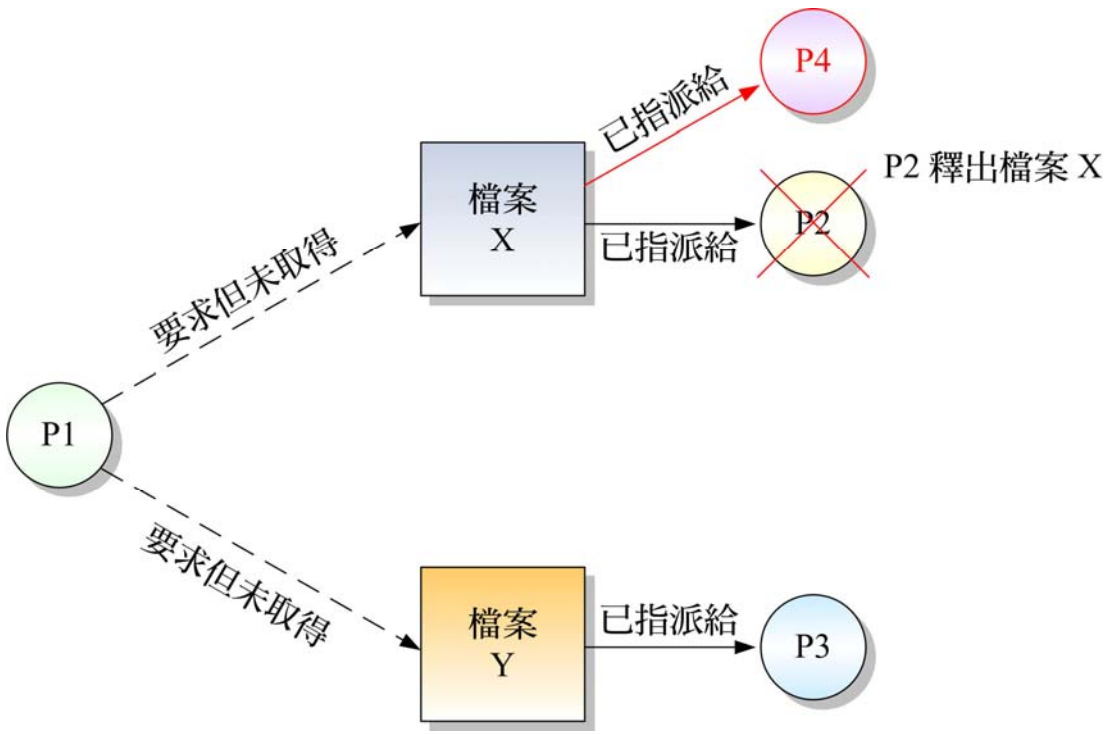


程序的同步課題

- 死結 (deadlock) 與飢餓 (starvation)
- 死結並不一定會發生，必須以下這四項條件都同時成立：
  - 互斥
  - 佔用並等候
  - 不可搶先
  - 循環式等候



- 飢餓現象：是死結的相反，它的成因是因為作業系統對於某個程序加上過多的資源限制條件，導致這個程序一直等不到全部的資源都齊全，因此無法執行

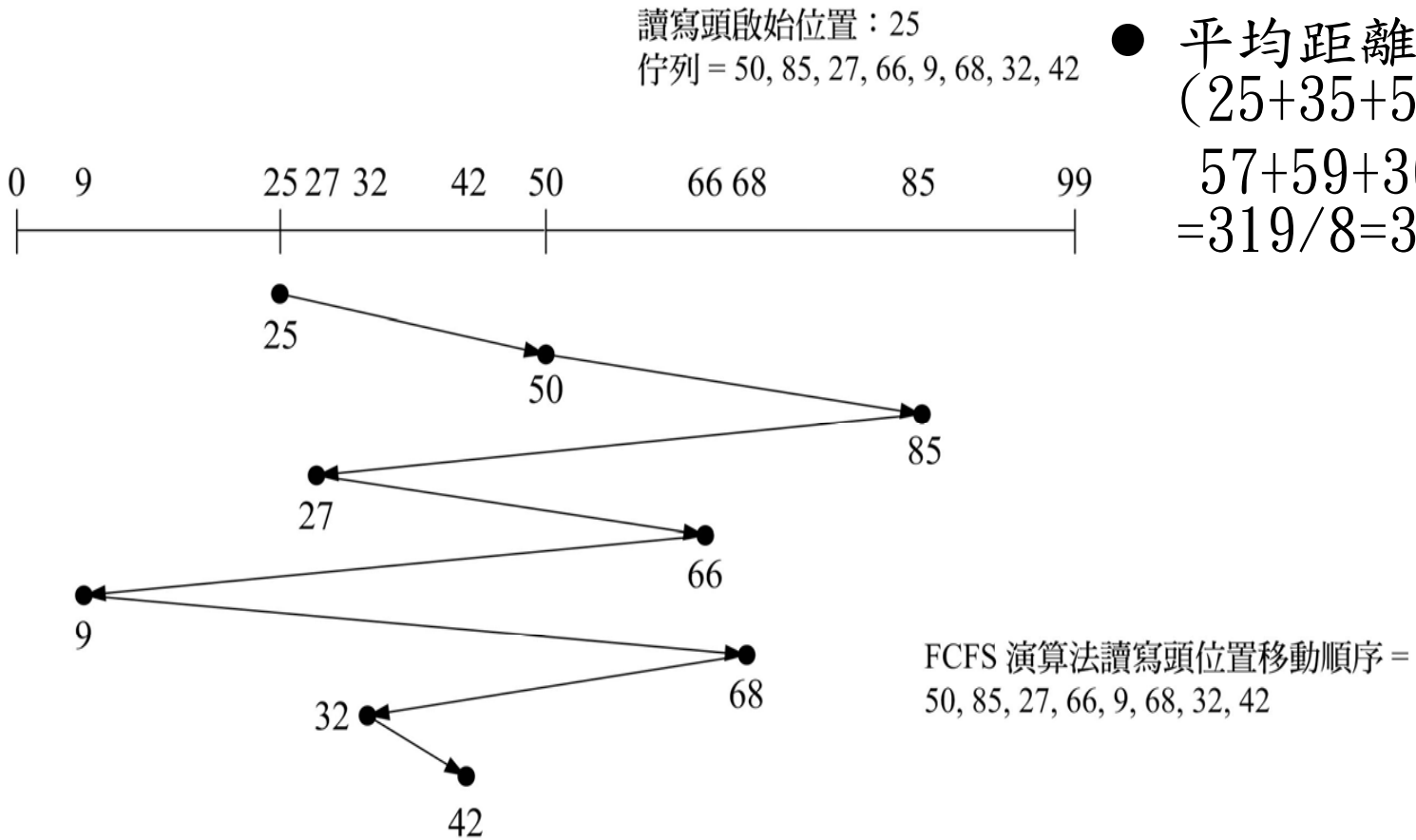


5-5 磁碟管理元件

- 磁碟排程的考慮重點是如何縮短搜尋時間，所以演算法的目標在於讓存取臂的移動距離和動作越少越好
- 常見的磁碟排程演算法有以下幾種：
  - FCFS演算法
  - SSTF演算法
  - 掃描法 (SCAN)
  - LOOK演算法

FCFS演算法

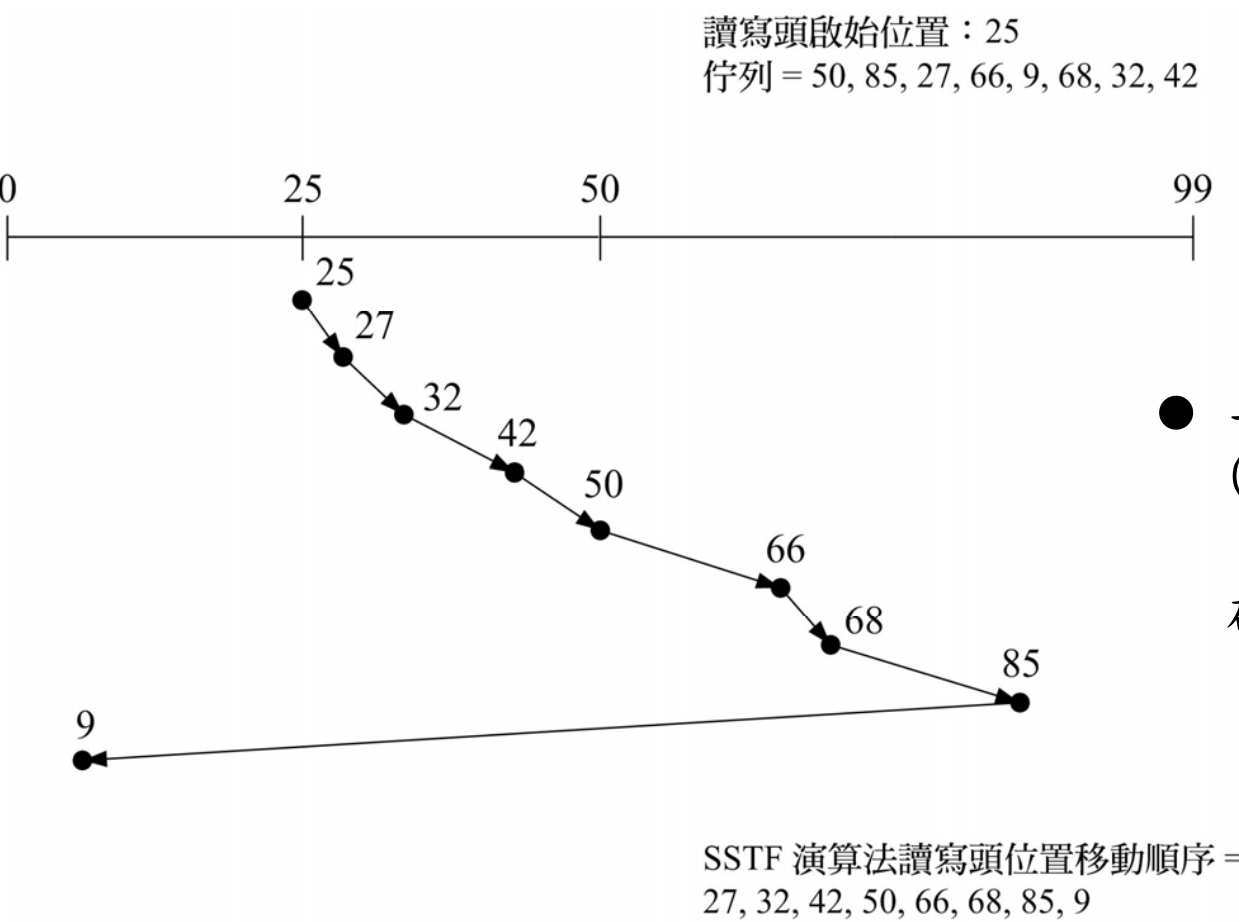
- 照先後來到的順序一個個來處理



- 平均距離 = (25+35+58+39+ 57+59+36+10)/8 =319/8=39.875 磁軌

SSTF演算法

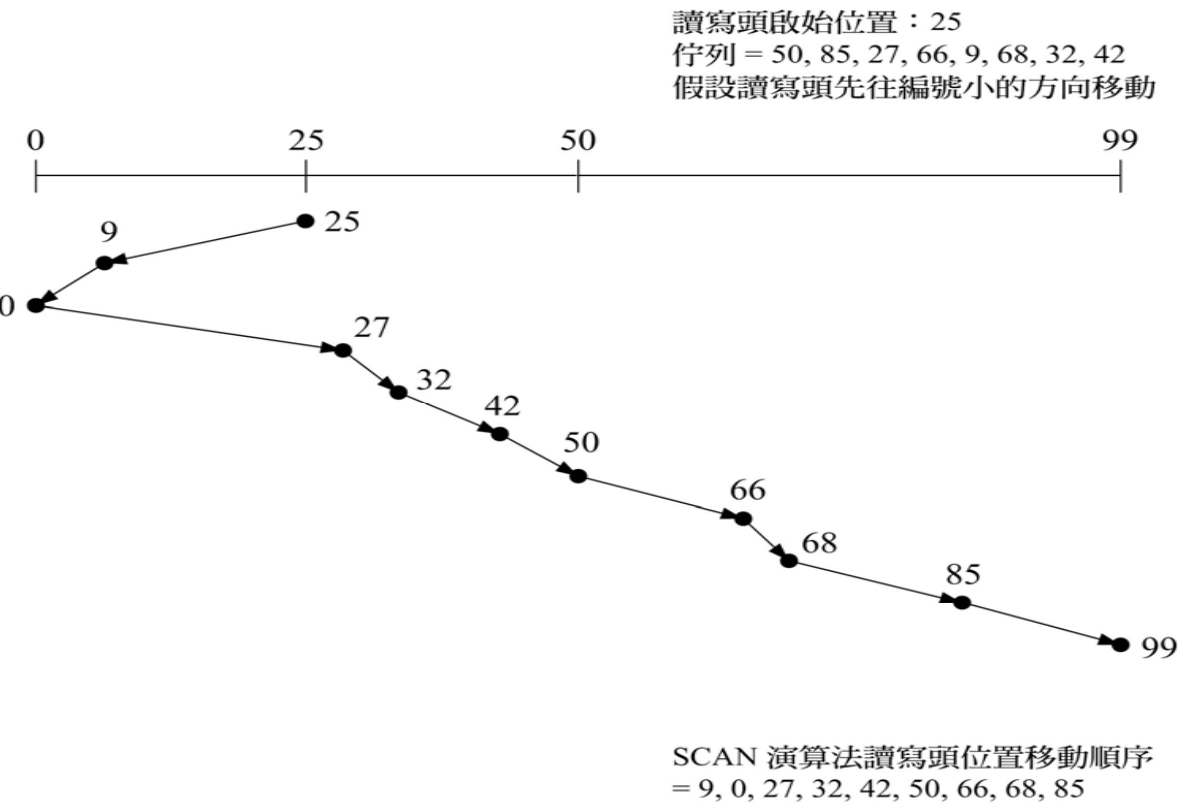
- 距離目前讀寫頭位置最近的工作先處理



- 平均距離 = (2+5+10+8+16+3+17+76)/8 =137/8=17.125 磁軌

掃描法 (SCAN)

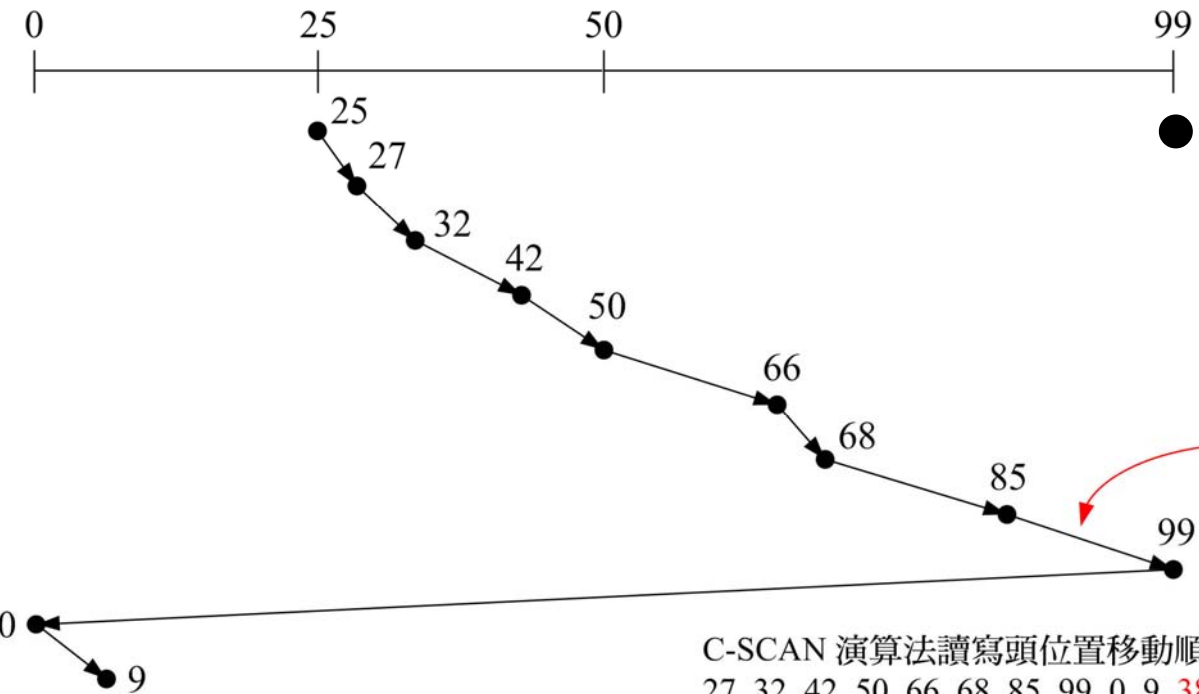
- 讓讀寫頭從磁碟的一端為起點，往另一端移動，在經過的途中一路處理，一直到另一端然後再反方向移動回來



- 平均距離 = (16+9+27+5+10+12+16+3+17+14)/8 =129/8=16.125 磁軌

循環掃描法 (C-SCAN)

- 在讀寫頭到達另一端時，不要反方向慢慢回頭，而是直接移回起點，以同方向往另一端處理



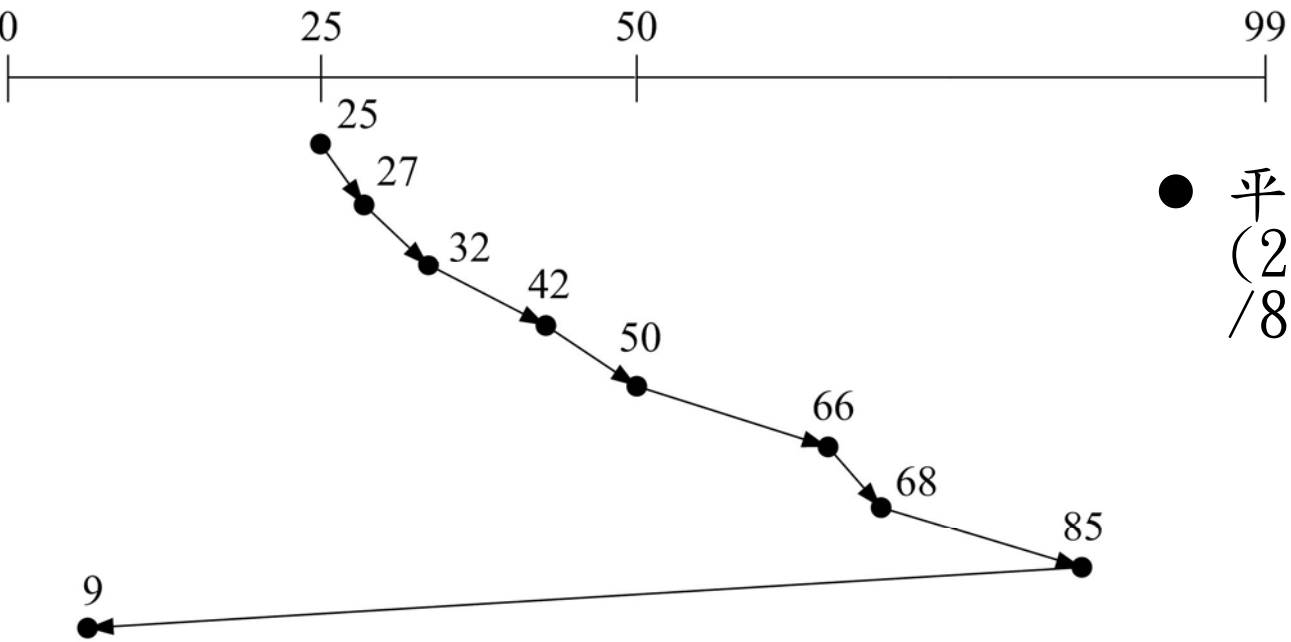
- 平均距離 = (2+5+10+8+16+3+17+14+99+9)/8 =183/8=22.875 磁軌

C-SCAN 演算法讀寫頭位置移動順序 =
27, 32, 42, 50, 66, 68, 85, 99, 0, 9, 38, 59, 73

而如果是 SCAN 演算法，結果 =
27, 32, 42, 50, 66, 68, 85, 99, 73, 59, 38, 0, 9

LOOK與C-LOOK演算法

- 是分別從SCAN和C-SCAN演變而來，主要的改變在於讀寫頭不必走到磁碟的起點和盡頭，只要走到目前磁碟工作編號最大和最小的位置

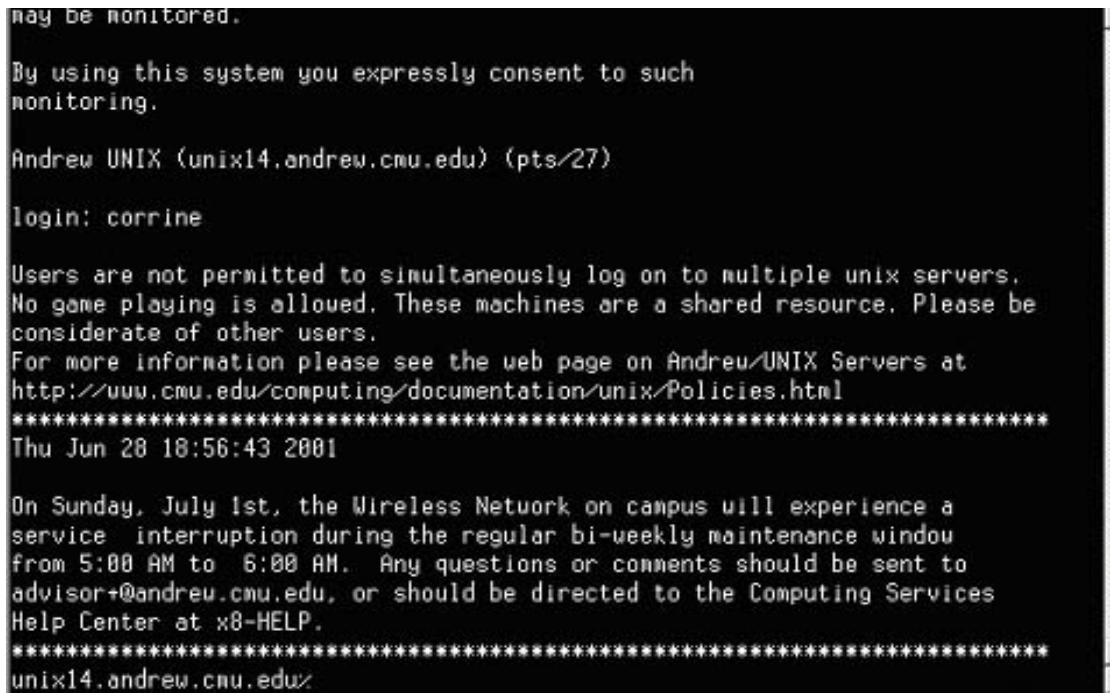


- 平均距離 = (2+5+10+8+16+3+17+76)/8 =137/8=17.125 磁軌

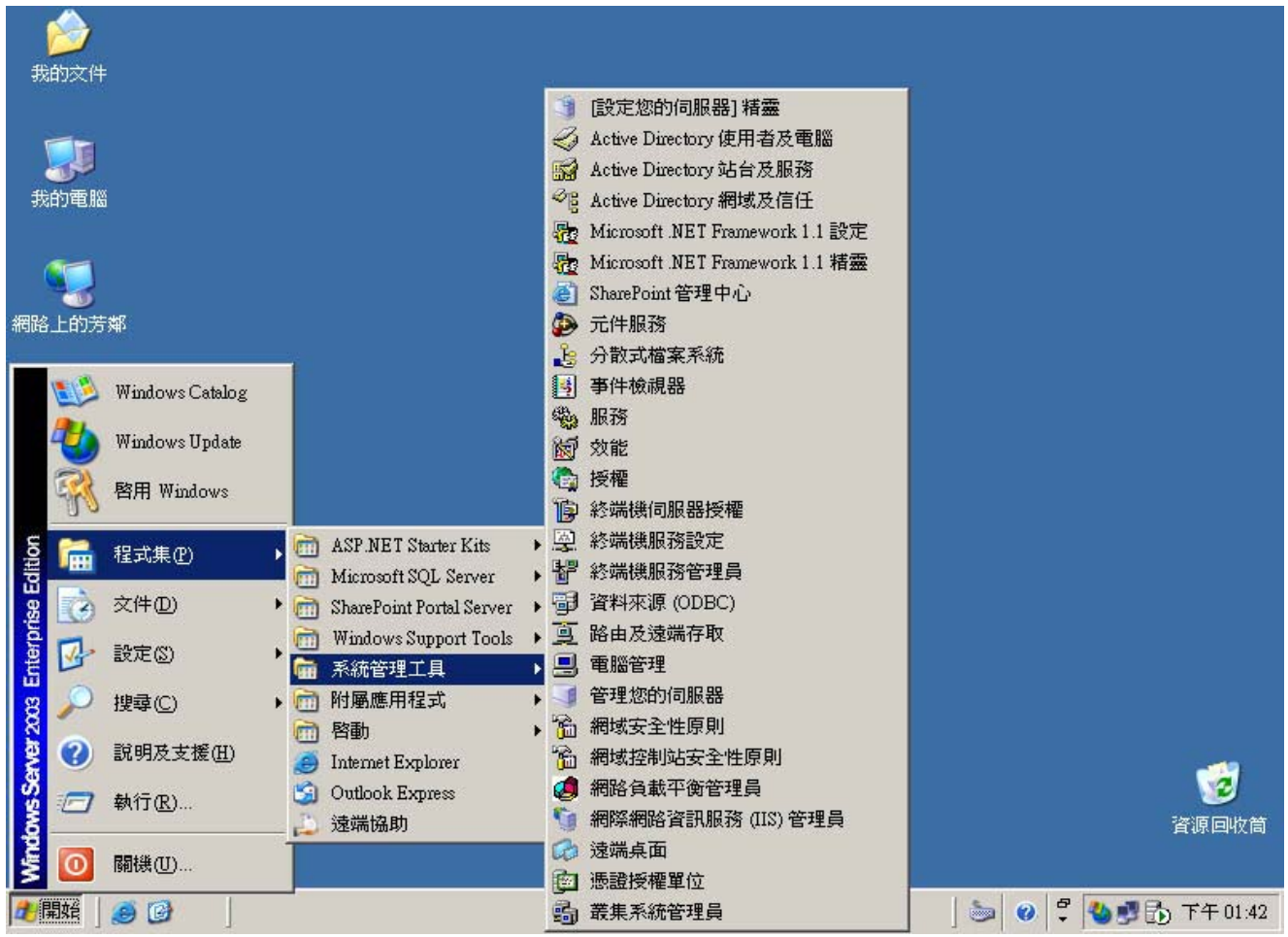
C-LOOK 演算法讀寫頭位置移動順序 =
27, 32, 42, 50, 66, 68, 85, 9

5-6 常見的作業系統

- Unix：多使用者的分時作業系統
- Unix是個可移植 (portable) 的作業系統
- Unix有一組功能非常強大的工具程式



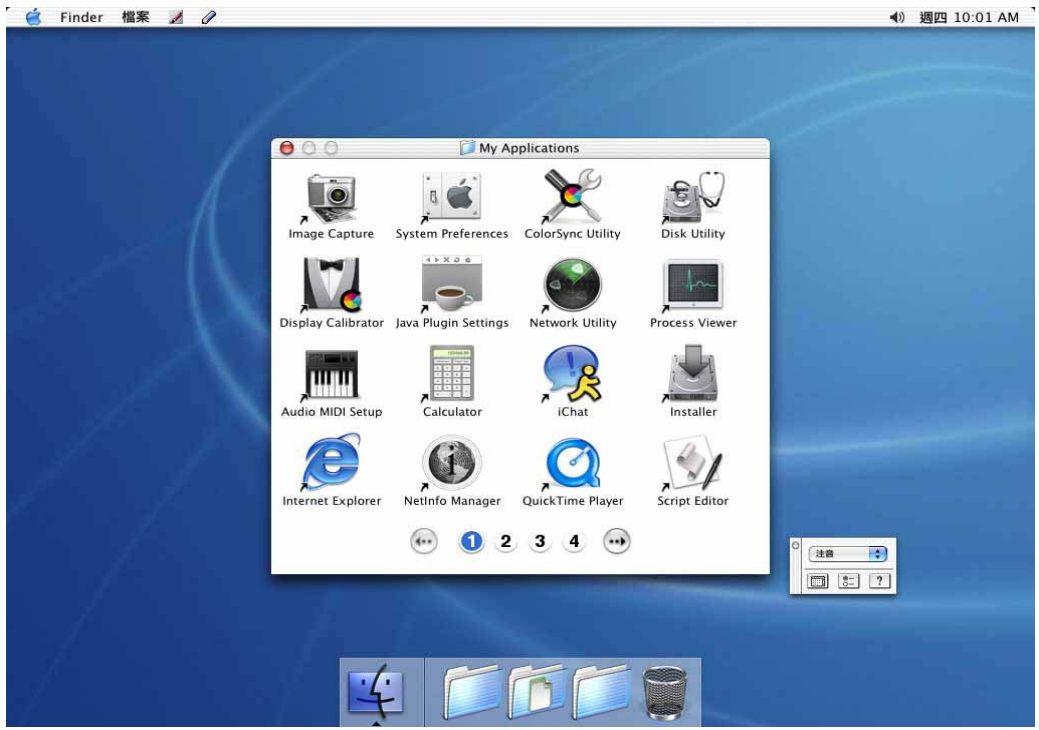
- Microsoft 的伺服器作業系統：Windows NT Server / Windows 2000 Server / Windows Server 2003



- Microsoft 的用戶端作業系統
  - 家用系列：Windows 95/98/ME/XP Home Edition，著重在使用的親和性和視聽娛樂方面，對多媒體、PC遊戲、家庭網路和熱門的週邊裝置有最佳的支援
  - 企業系列：NT Workstation / Windows 2000 Professional / Windows XP Professional，用在企業網路中當作用戶端作業系統，著重於穩定度和企業網路的支援，目標是要能夠融入企業的主從式架構網路環境，方便連線與存取伺服器上的程式和檔案

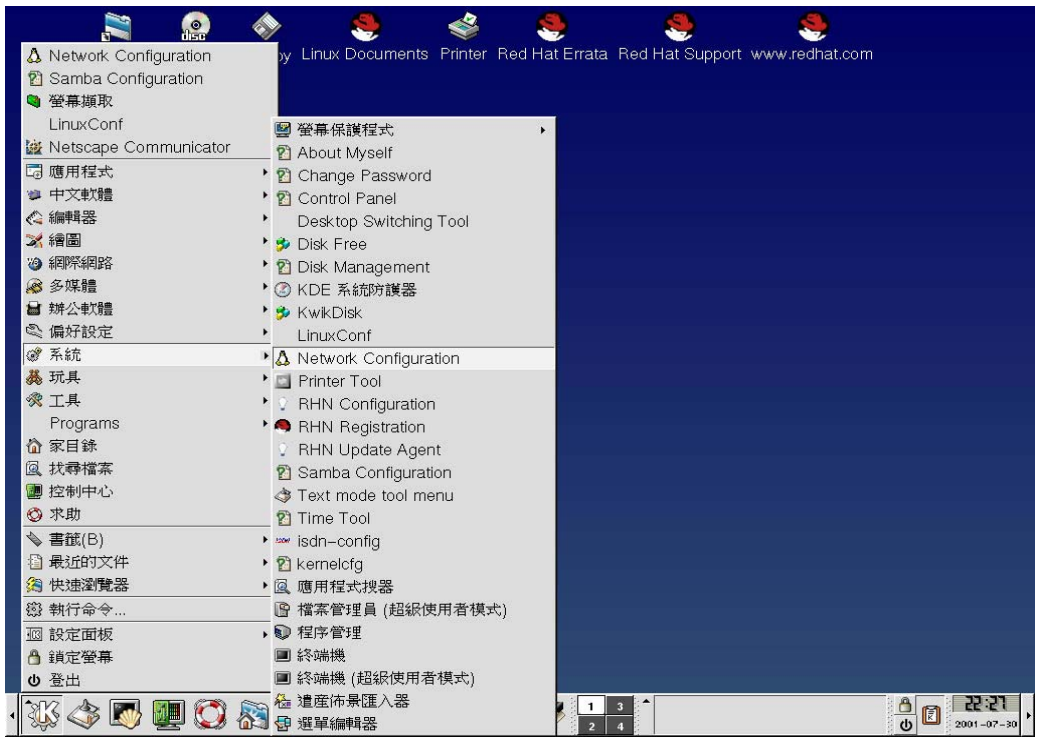
Mac OS

- 是Apple公司Macintosh電腦的作業系統
- 在專業的美工排版與美術設計領域常見
- 最新版是Mac OS X



Linux

- 是一種開放原始碼 (open-source) 軟體
- 1991年由芬蘭大學生Linus Torvalds所創作 ，核心類似Unix



Windows CE

- 是專為PDA、嵌入式系統與Internet家電市場所設計的，以Windows為基礎的模組化作業系統
- 嵌入式系統 (embedded system)：是一種整合在其他產品中的計算裝置
- 能夠與桌上型電腦快速的同步，還內建有手寫辨識和錄音的功能