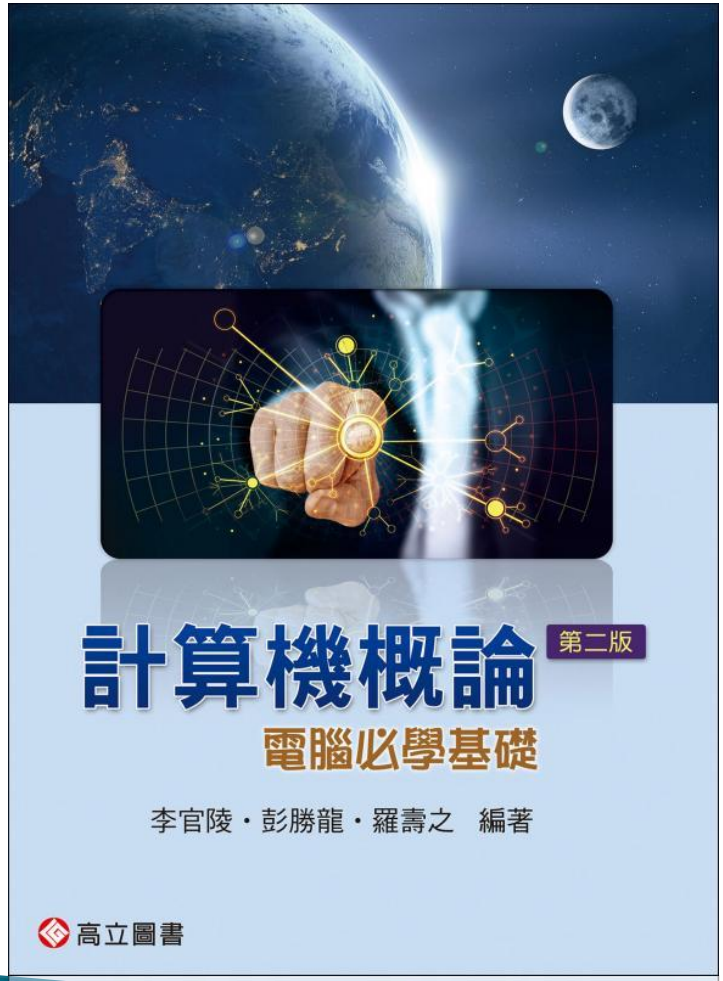


Ch05
作業系統

李官陵 彭勝龍 羅壽之

學習目標

- 電腦未開機是一部冰冷的機器，但一開機後就「活」了起來，很重要的原因是「作業系統」
- 作業系統提供人機操作介面，也管理電腦的軟硬體
- 也是「資源管理者」或「資源分配者」

大綱

- 行程管理
- 記憶體管理
- 儲存體管理

何謂作業系統

- 作業系統是一個介面，一種人機介面。由於它必須管理電腦的硬體設備，所以也可以說是資源管理者或資源分配者，所謂的資源，範圍也蠻廣的，除了看得見的硬體資源，如記憶體暫存裝置、各種磁碟輸出入裝置、印表機輸出裝置等，還有看不見的軟體資源，如程式、檔案等。

作業系統概觀

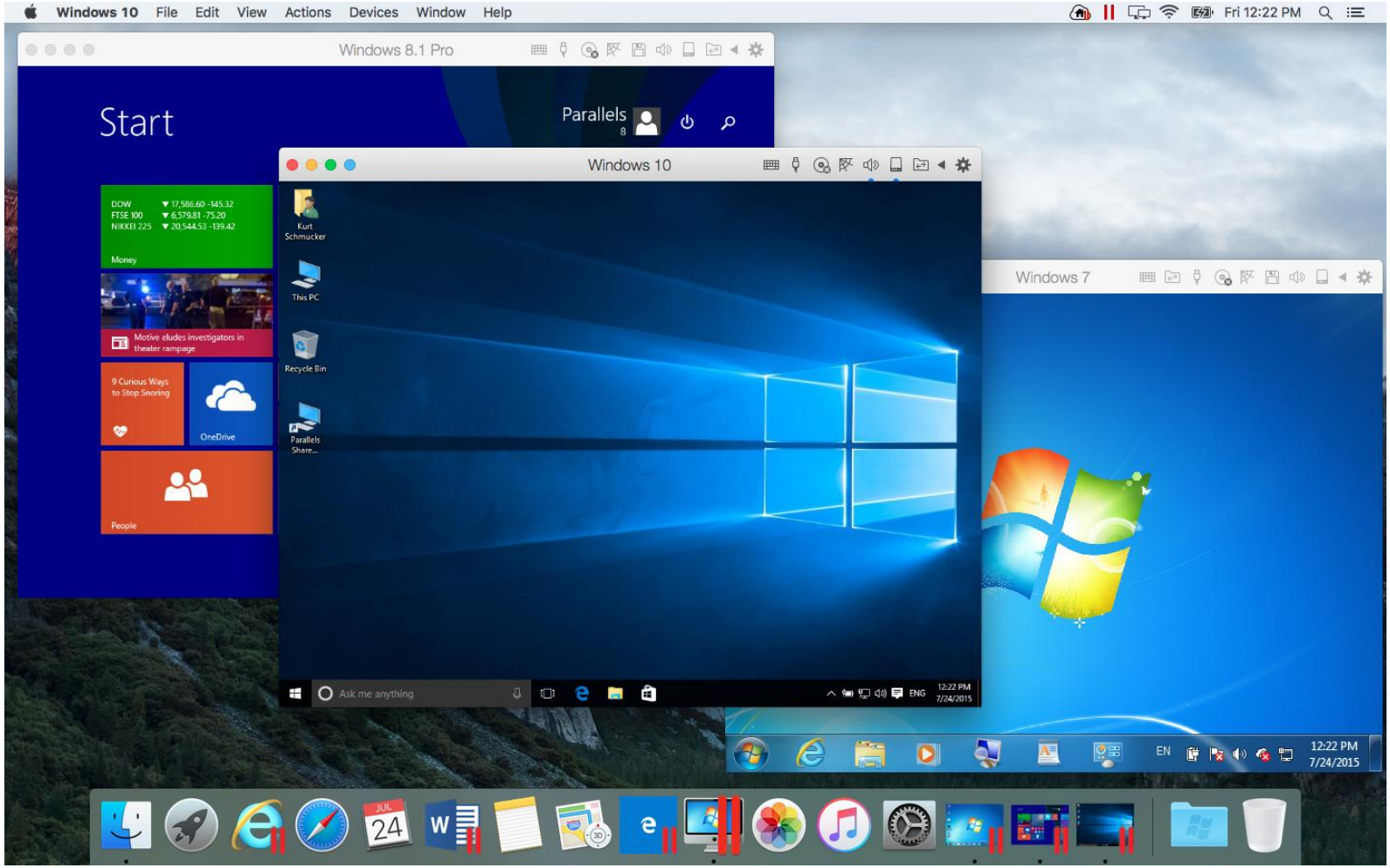


圖 5.1 Mac OS 和 Windows 的操作畫面（圖片來源：Parallels 公司官網）

作業系統概觀(續)

- 智慧型手機作業系統

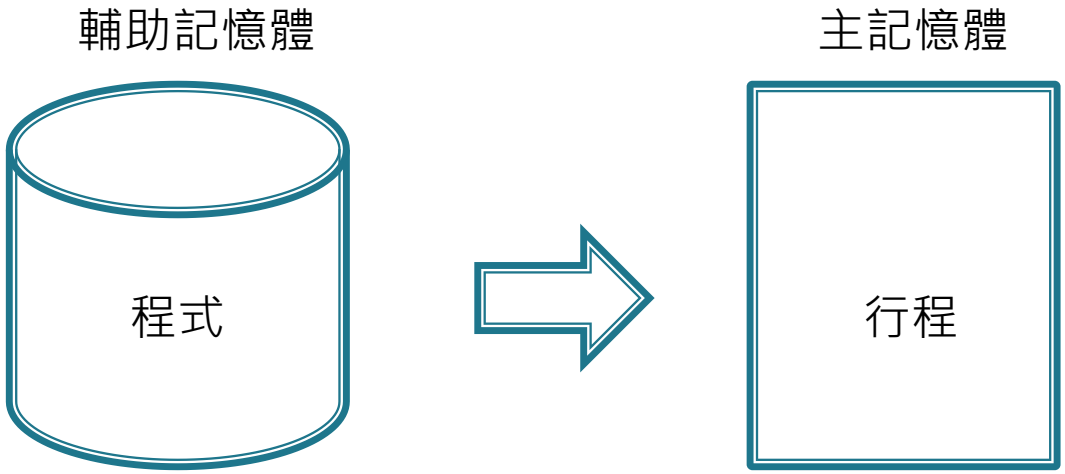
https://read01.com/nxBKk3.html#.WbtQwMgjFPY

Apple vs. Microsoft

- https://www.youtube.com/watch?v=f_Q0XPher3o&feature=iv&src_vid=D_RE2uC8QhE&annotation_id=annotation_627499127

行程管理

- 行程 (process) 是一個執行中的程式，也有人稱為工作元。
- 多工(multitasking)系統就是允許多個行程同時待在記憶體裡等待分配執行的系統。



不同的多工系統

- 即使只有擁有單一計算核心，也可以透過行程切換（或置換）的管理方式，達到多工的效果
- 行程的切換管理基本有三種形式
  - 多程式（multiprogramming）系統：一個行程會因為輸出入事件進入等候狀態，將CPU執行權交給另一個行程
  - 分時（time-sharing）系統：一個行程使用一段CPU時間後會自動將執行權交給另一個行程
  - 即時（real-time）系統：一個進入等候的行程能在限定時間內重新取得CPU執行權

行程管理

- 新生 (new)：產生新的行程。
- 執行 (running)：正在執行。
- 等待 (waiting)：等待某事發生，例如等待使用者輸入完成或等待印表機輸出完畢。
- 就緒 (ready)：排班中，等待使用 CPU。
- 結束 (terminated)：完成工作，結束行程。
- 這五種狀態可以下圖說明它們之間的關係。

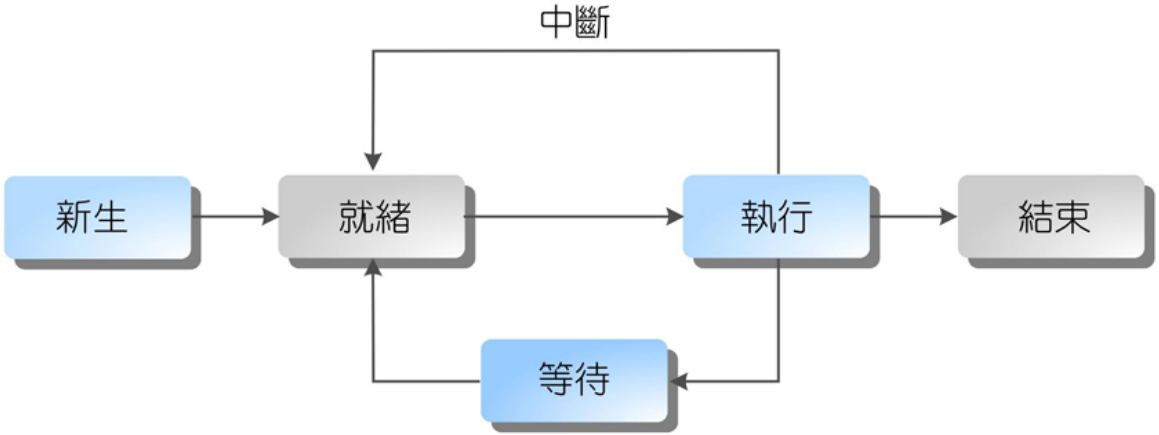
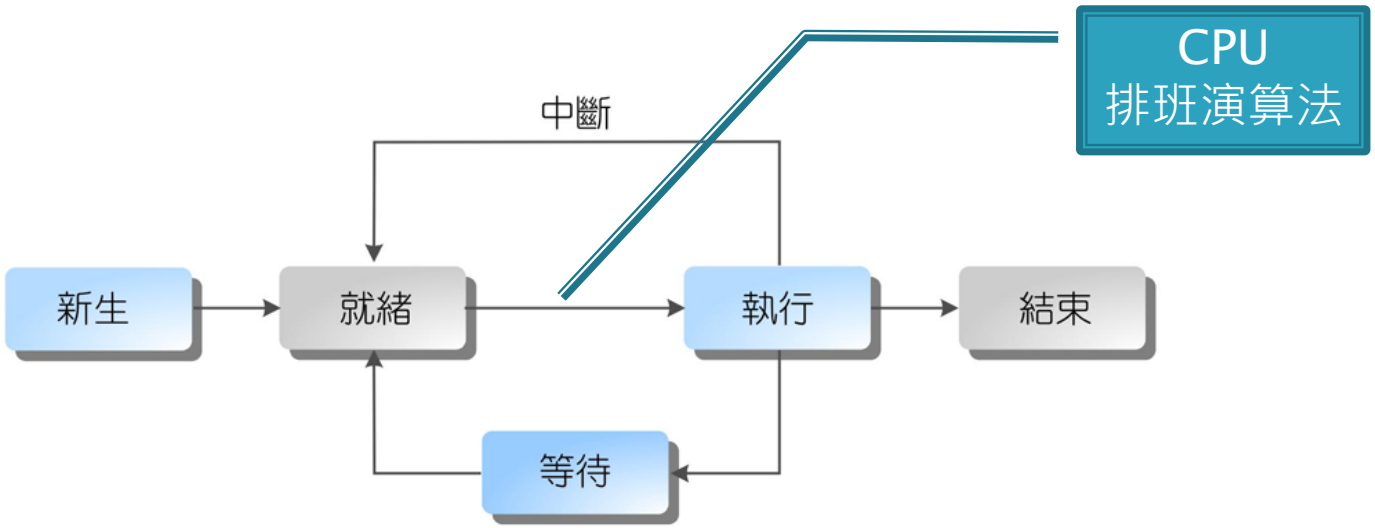


圖 5.2　行程狀態圖

行程管理

- 行程管理最重要的工作，就是要有一個好的排班演算法，CPU-排班演算法 (CPU-scheduling algorithm)，評估排班演算法的優劣，一般都是用等待時間 (waiting time) 來衡量。等待時間越長，顧客滿意度就越低。

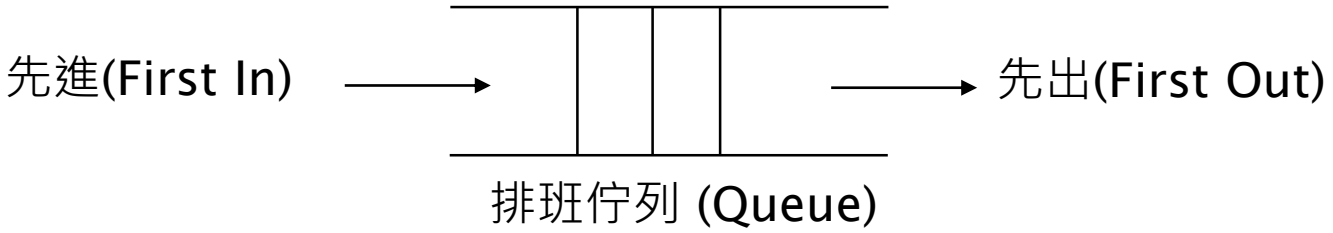


行程管理(續)

- 行程排班法則
  - 先到先服務 (FCFS)
  - 最短工作優先 (SJF)
  - 最短剩餘時間優先 (SRTF)
  - 優先權排班法
  - 循環排班法 (RR)
  - 多層佇列排班法 (MQ)
  - 多層回饋佇列排班法 (MFQ)

先到先服務排班法

- 先到先服務排班法(First Come First Serve, FCFS)，就是誰先到，誰就先使用 CPU。此法並不允許行程在還沒完成工作前被置換出去，屬於不可奪 (non-preemptive) 的排班演算法。



先到先服務排班法 (續)

- 三個行程 P_1、P_2 和 P_3，它們完成工作需要的時間分別為 5、2 和 3 毫秒，那麼使用 FCFS 排班法的執行甘特圖如圖 5.3。

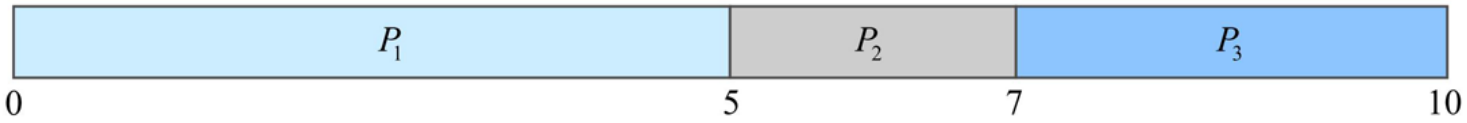


圖 5.3 FCFS 排班法的執行甘特圖

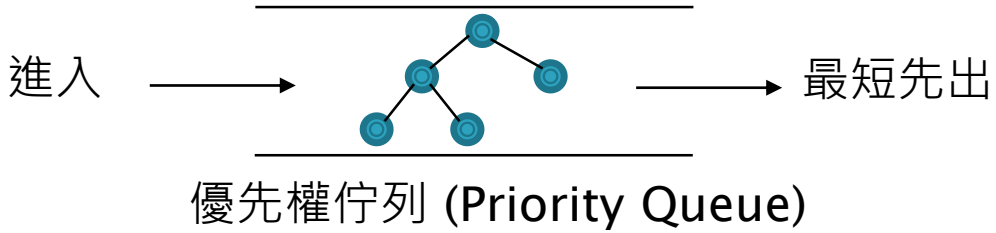
- 由上面的甘特圖，我們可以算出 P_1 的等待時間是 0 毫秒，P_2 的等待時間是 5 毫秒，而 P_3 的等待時間則是 7 毫秒，如此可以算出總等待時間是 0 + 5 + 7 = 12 毫秒，也可以計算每個行程的平均等待時間為 12 / 3 = 4 毫秒。

隨堂練習

- 三個行程 P_1、P_2 和 P_3，它們完成工作需要的時間分別為 7、5 和 3 毫秒，那麼使用 FCFS 排班法的總等待時間和平均等待時間各為何？

最短工作優先排班法

- 最短工作優先(Shortest Job First, SJF)，亦即比較小的工作具有較高的執行優先權。 SJF 也是不可奪的排班法則



最短工作優先排班法(續)

- 三個行程 P_1、P_2 和 P_3，它們完成工作需要的時間分別為 5、2 和 3 毫秒，那麼使用 SJF 排班法的執行甘特圖如圖 5.4。

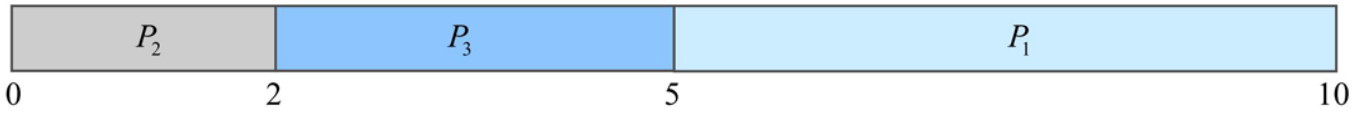


圖 5.4　執行甘特圖

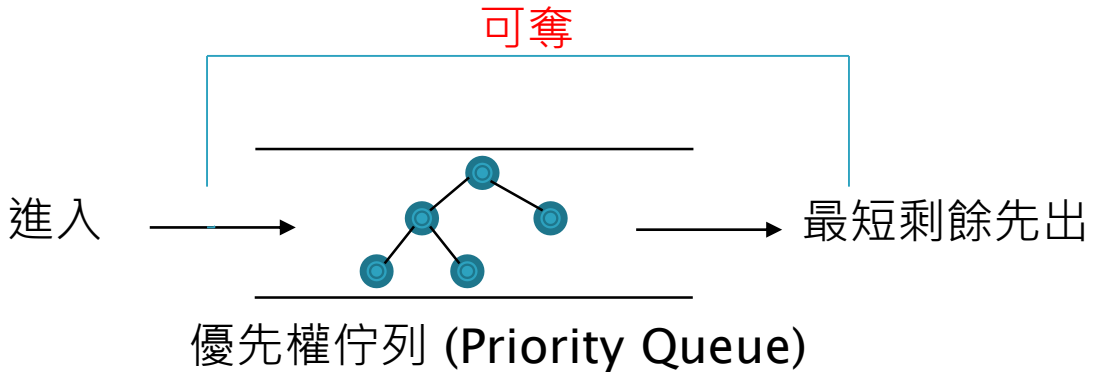
- 同樣的，我們可以算出 P_1 的等待時間是 5 毫秒，P_2 的等待時間是 0 毫秒，而 P_3 的等待時間則是 2 毫秒，如此可以算出總等待時間是 5 + 0 + 2 = 7 毫秒，也可以計算出每個行程的平均等待時間為 7 / 3 = 2.3 毫秒。

隨堂練習

- 三個行程 P_1、P_2 和 P_3，它們完成工作需要的時間分別為 7、5 和 3 毫秒，那麼使用 SJF 排班法的總等待時間和平均等待時間各為何？

最短剩餘時間優先排班法

- SJF 是不可奪的排班法則，如果我們允許行程被中途暫停，也就是可奪的 (preemptive)，那麼就變成最短剩餘時間優先 (Shortest Remaining Time First, SRTF)，在 SRTF 中，如果我們目前在執行一個需要 10 毫秒的工作，並且已經執行 5 毫秒，如果這時進來了一個只需要 2 毫秒的工作，那麼排班程式就會把目前行程送入就緒佇列，而先執行目前新進的行程，這就是 SRTF。

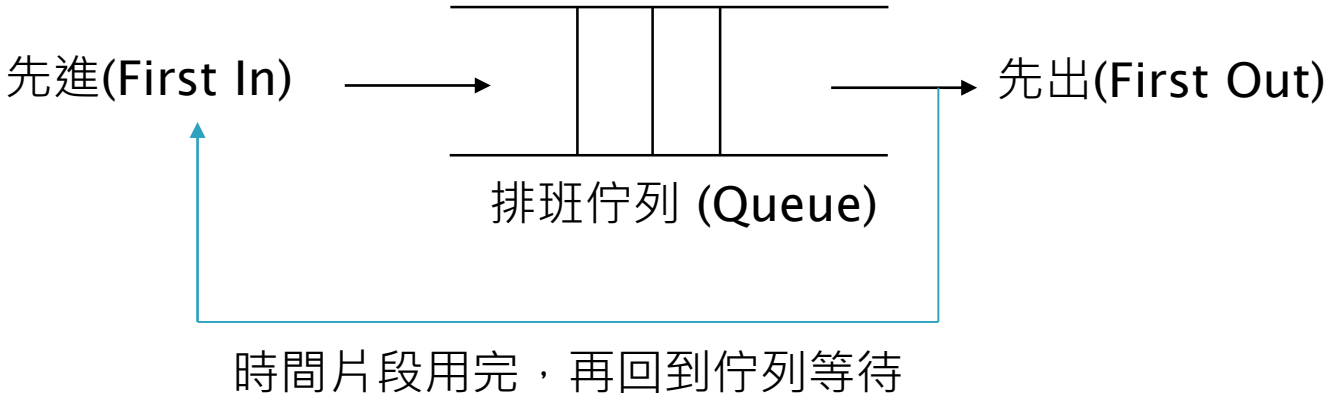


優先權排班法

- 優先權排班法
  - FCFS 的優先權是由「進入就緒佇列的時間」來決定。
  - SJF 的優先權是由「工作的長短」來決定
  - SRTF 的優先權是由「剩餘工作的長短」來決定。
- 只要每個行程進入系統後都能事先計算出它的優先權，排班程式就可以依照優先權來決定其執行的先後次序，如果我們允許在執行中可以重新計算優先權，那麼就可以變成可奪式優先權法。
- 優先權排班法的最大問題就是飢餓 (starvation)，低優先的行程永遠無法被執行的狀態。為了避免飢餓的發生，通常還會加上計時 (aging) 功能。

循環排班法

- 循環 (Round Robin, RR) 排班法可以說是 FCFS 的一個可奪式版本，行程還是依照先到先服務的次序，但是每個行程可以執行的時間是固定的，此固定的單位時間稱為時間量 (time quantum) 或時間片段 (time slice)。



循環排班法(續)

- 三個行程 P_1、P_2 和 P_3，它們完成工作需要的時間分別為 5、2 和 3 毫秒，若時間片段設為1毫秒，那麼使用 RR 排班法的執行甘特圖如圖 5.5。

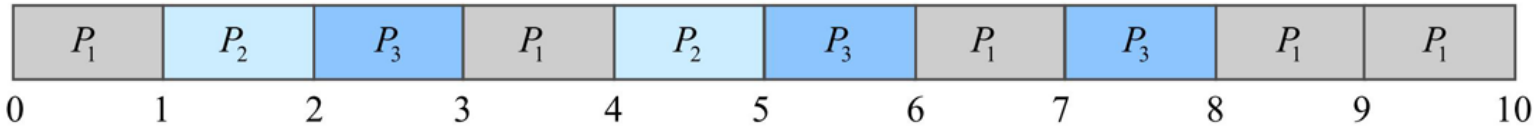


圖 5.5　時間片段設為 1 毫秒的執行甘特圖

- 由上圖我們可以算出 P_1 總共等待了 5 毫秒，P_2 等待了 3 毫秒，而 P_3 等待了 5 毫秒，所以總共等待了 13 毫秒。

循環排班法 (續)

- 如果我們把時間片段設為2 毫秒，那麼執行甘特圖就變成如圖 5.6 了。

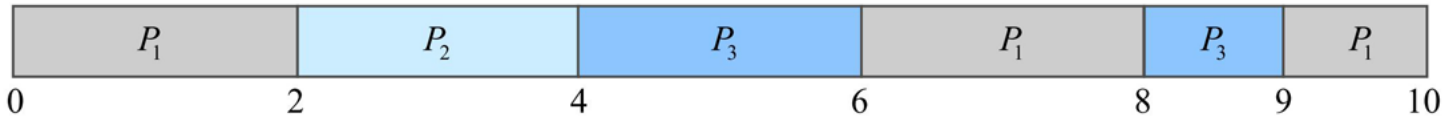


圖 5.6 時間片段設為 3 毫秒的執行甘特圖

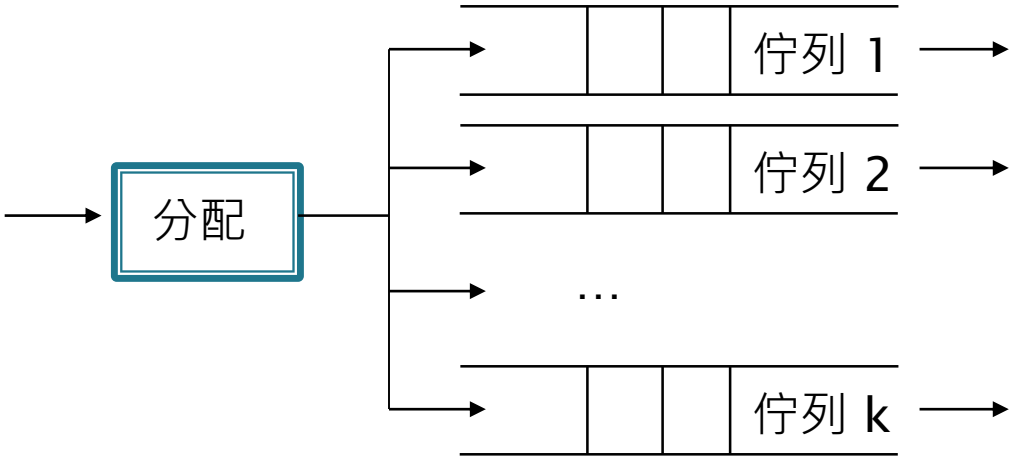
- 同樣的，我們可以算出 P_1 總共等待了 5 毫秒，P_2 等待了 2 毫秒，而 P_3 等待了 6 毫秒，雖然總共還是等待了 13 毫秒，但是各別行程等待的時間還是有所不同，例如 P_2 和 P_3 就跟前一個例子不一樣了。

隨堂練習

- 三個行程 P_1、P_2 和 P_3，它們完成工作需要的時間分別為 7、5 和 3 毫秒，那麼使用 RR 排班法，時間片段分別為 2 和 3 的總等待時間和平均等待時間各為何？

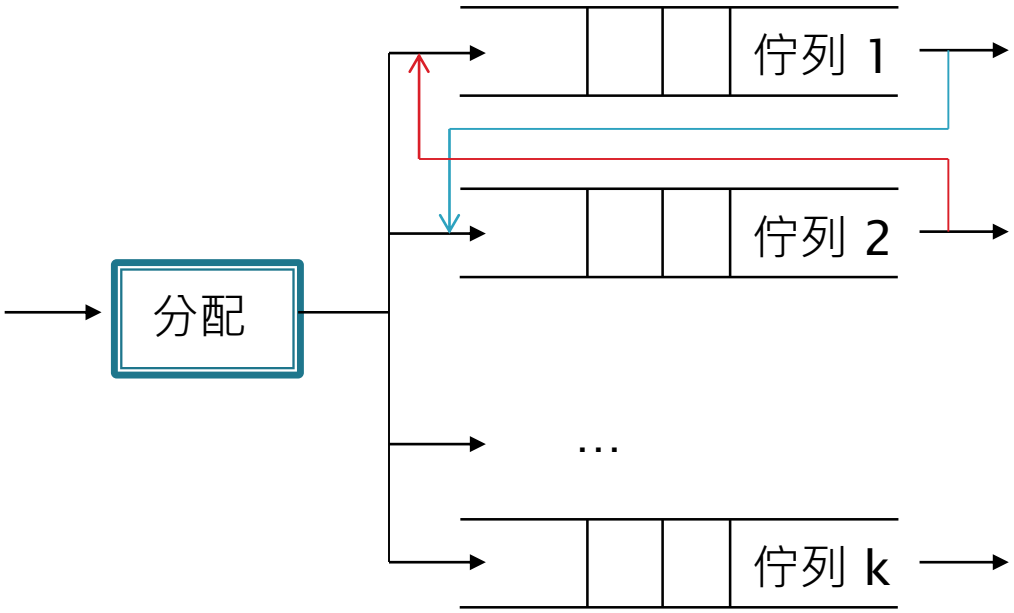
多層佇列排班法

- 多層佇列 (Mutilevel Queue, MQ) 排班法是將優先權的概念應用在就緒佇列裡，例如系統 (system) 行程優於交談 (interactive) 行程，而交談行程又優於批次 (batch) 行程等。
- 在同一個佇列裡的行程則可以再應用 FCFS 或 RR 來排班，使得排班的方法更有彈性。類似優先權排班法，MQ 也可能造成飢餓。



多層回饋佇列排班法

- 多層回饋佇列 (Mutilevel Feedback Queue, MFQ) 排班法是將MQ加入計時功能，避免飢餓產生。很明顯的，MFQ 要比 MQ 更有彈性。因此也是作業系統最常採用的排班法。



記憶體管理

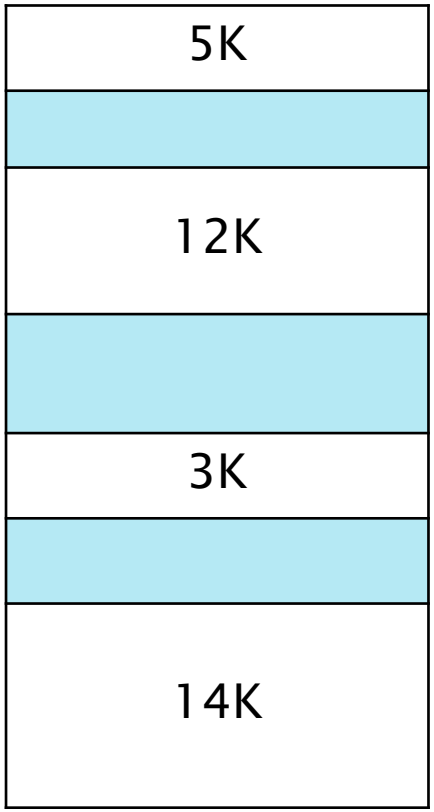
- 當一個行程被執行時，必須要在記憶體裡才行，因此多工系統允許多個行程被執行，它們就必須通通放到記憶體裡，因此記憶體的大小就決定多工的等級 (degree)，也就是允許同時執行工作的數量。
- 行程結束後，即釋放原占用的記憶體，該釋放的記憶體就被回收，通常就稱為垃圾收集 (garbage collection)。
- 記憶體通常被分為二部分，一部分為系統區，專門放置被執行的系統程式和系統資料；另一部分為使用者空間，用來存放使用者行程和資料的地方，而記憶體管理指的就是使用者空間管理。

記憶體管理(續)

- 記憶體配置策略
  - 連續記憶體配置
  - 分段配置
  - 分頁配置

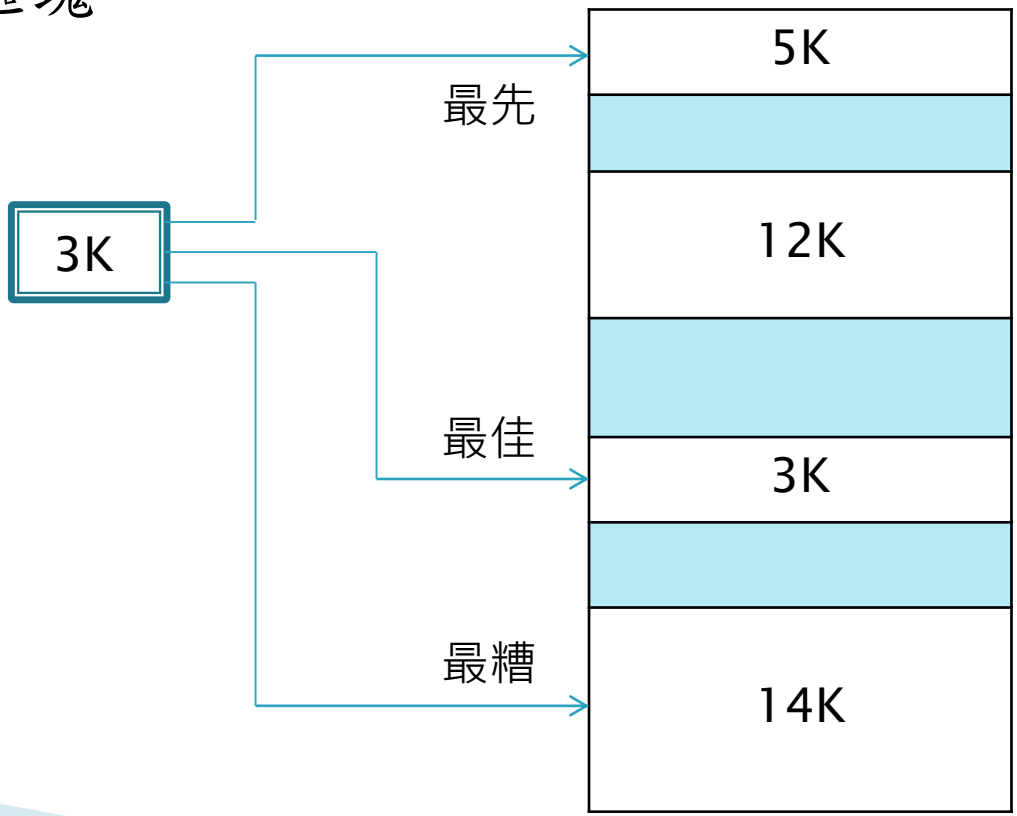
連續記憶體配置

- 這是最簡單的記憶體配置方法，非常適合個人單純的作業系統，剛開始記憶體的初始狀態是一塊完整的自由空間，記憶體的配置是以一個連續空間分配給一個行程為準。
- 連續記憶體配置策略
  - 最先 (first fit) 配置法
  - 最佳 (best fit) 配置法
  - 最糟 (worst fit) 配置法



連續記憶體配置 (續)

- 目前的自由區塊分別是 5K、12K、3K 和 14K，若目前有一個 3K 的行程要置入記憶體，則用最先法會選擇 5K 的自由區塊，若是最佳法就會選擇 3K 的區塊，而最糟法會選擇 14K 的區塊。



連續記憶體配置

- 連續記憶體配置 (Contiguous Memory Allocation) 的垃圾回收通常使用連結串列記載。

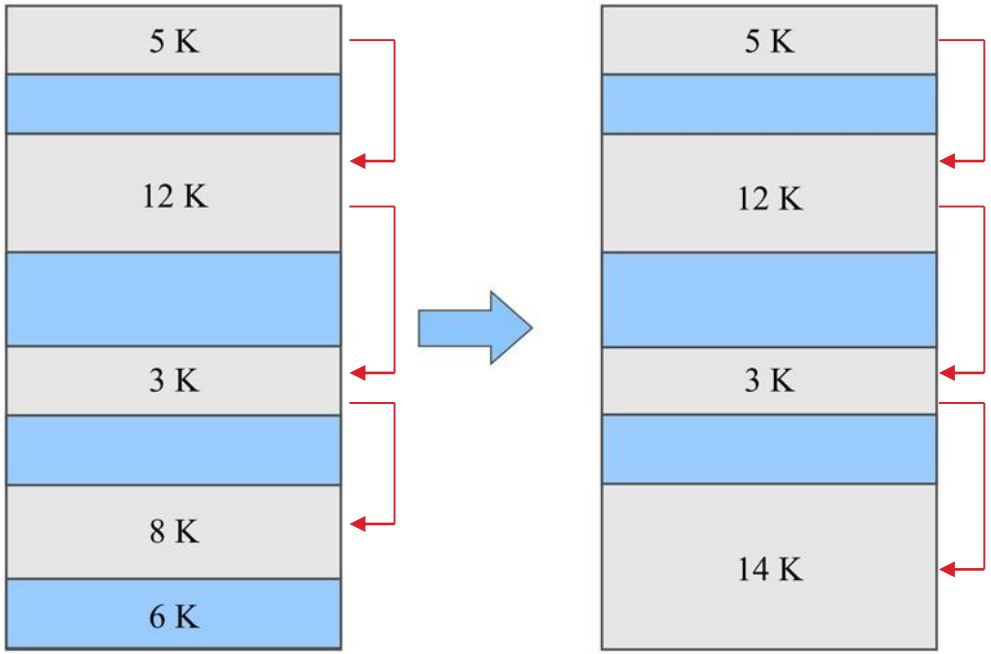


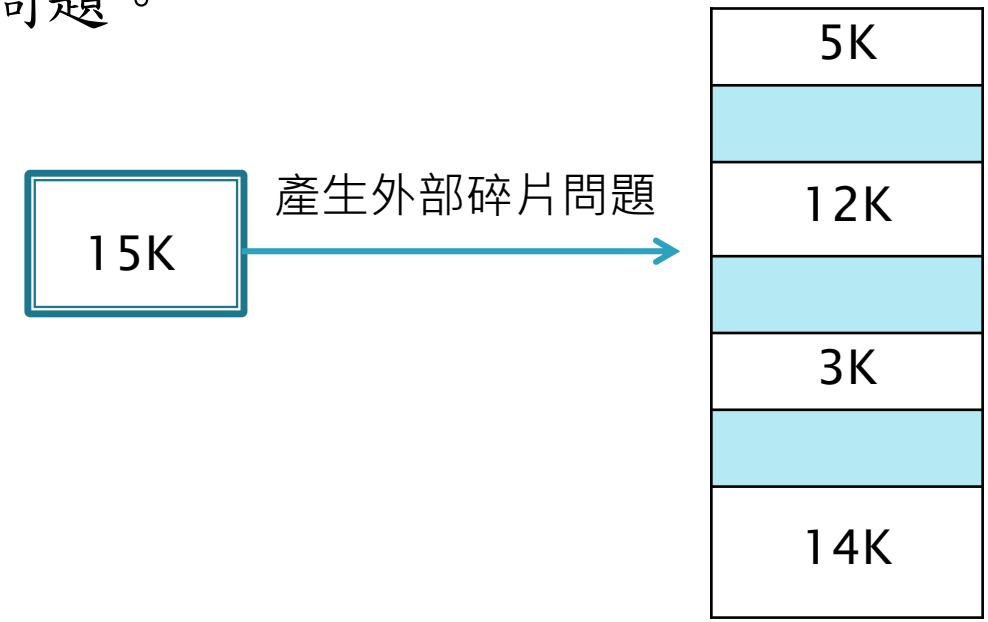
圖 5.7　6K 和其上 8K 的自由空間合併成一個 14K 的自由區塊

連續記憶體配置 (續)

- 記憶體的配置經常會出現碎片 (fragmentation) 的情況
  - 外部碎片 (external fragmentation) 是指未分配給任何行程的自由區塊總和比行程大，但卻無法分配給任何其他的行程使用。
  - 內部碎片 (internal fragmentation) 則是指分配給某一行程，但卻不會被使用的空間。
  - 連續記憶體配置有外部碎片問題

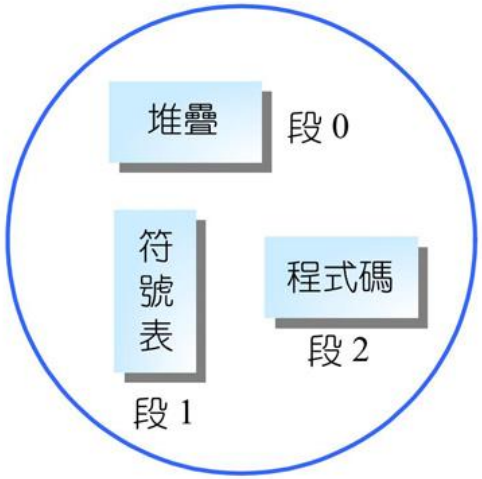
連續記憶體配置 (續)

- 連續記憶體配置有外部碎片問題
- 記憶體重組 (defragmentation)是將執行中的行程移到連續的位置，使得空出的自由空間為一個連續的區塊，可解決外部碎片問題。



分段配置

- 行程的組成可能含有很多部分，例如程式碼、變數、使用的資料結構 (如堆積和堆疊) 和程式庫等，因此將這些部分以段 (segment) 來區分，也就是說把一個行程打散成好多個段放入記憶體，稱為分段配置 (segmentation)。



	起始位置	長度
0	1200	400
1	4000	500
2	2500	1000

分段表

圖 5.8　分段配置的一個例子

分段配置 (續)

- 使用分段配置，一個行程不需放在連續的記憶體上，但是每一片段還是要在一個連續的位置上。
- 需使用額外的表格記錄每一段的起始位置和該分段的大小，此表格稱為分段表 (segment table)，所有的記憶體存取都會透過分段表的轉換，而得出一個正確的記憶體位址。
- 為連續配置的改良，但仍可能會有外部碎片的問題。

分頁配置

- 將記憶體和行程切成一樣大小的區塊，再依照區塊來一一配置，而減少分段法搜尋區塊的程序，這個方法就稱為分頁配置 (paging)，通常在記憶體上的區塊被稱為頁框 (page frame)，而行程被切成的區塊就稱為分頁 (page)。
- 分頁配置也需要一個分頁表 (page table) 將分頁與頁框做成連結來對應到正確的位址。因為每個分頁和頁框的大小一樣，所以分頁表所記載的資訊就相對的簡單。

分頁配置 (續)

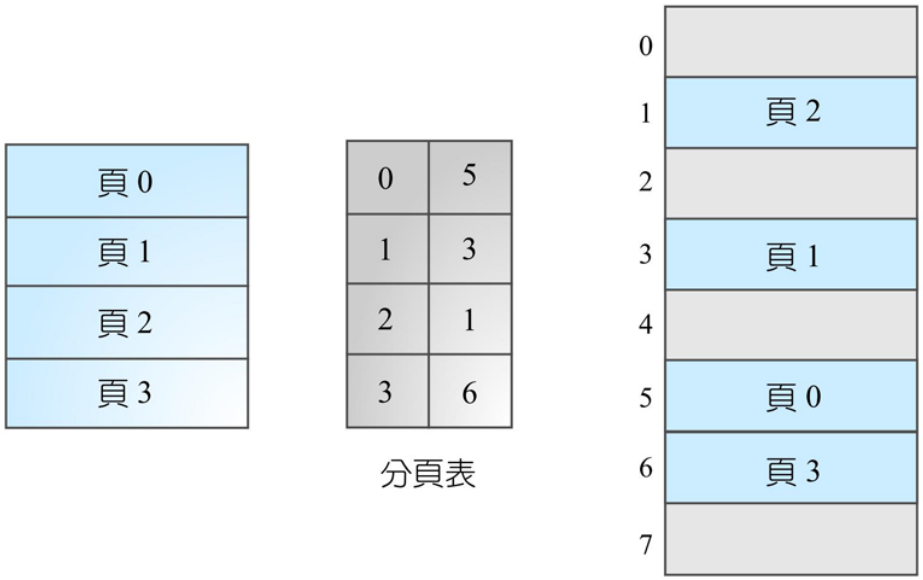


圖 5.9 分頁配置的概念圖

- 分頁的製作比分段還容易，且可靠硬體支援。
- 製作虛擬記憶體較容易。
- 沒有外部碎片的問題，但有內部碎片的問題。

儲存體管理

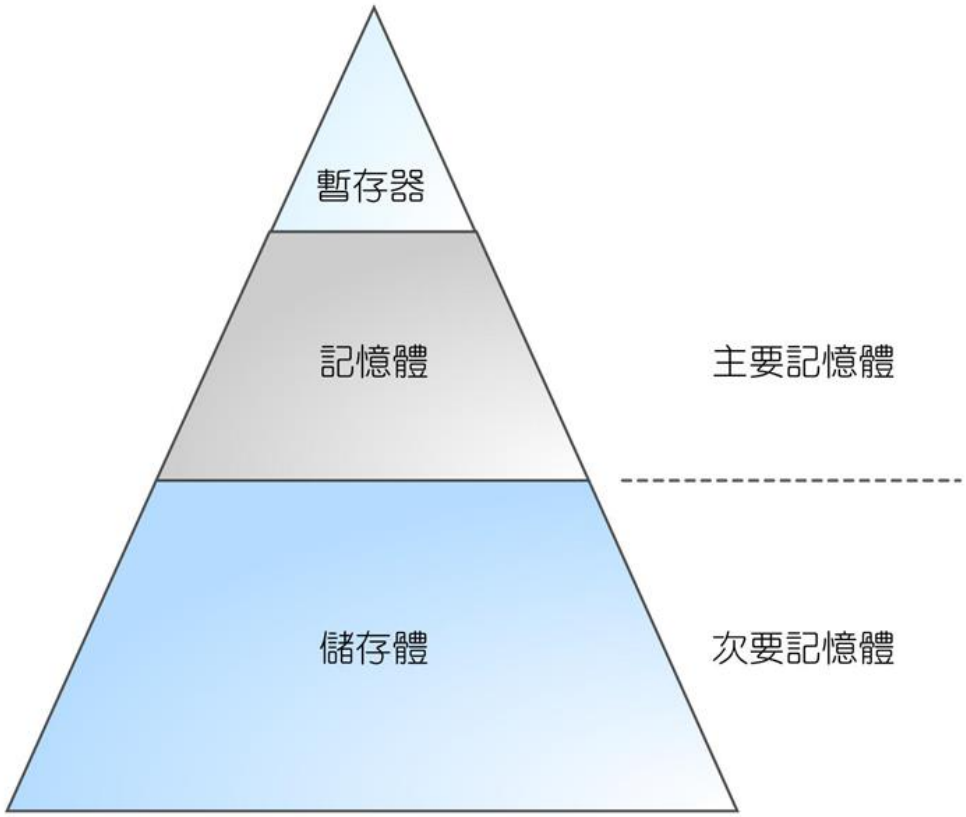


圖 5.10　電腦的儲存架構金字塔

檔案配置

- 檔案系統則是用來提供系統存取檔案用的。
- 這些資訊皆存於儲存體的目錄區 (directory)，和分頁結構相似，檔案的配置也是以一個區塊 (block) 為單位。
- 早期系統的目錄區稱為根目錄，它的空間是固定的，表示可以存放的檔案數目有限(如FAT)。透過叢集串接(cluster chain)，根目錄區可以延伸到資料區，因此檔案沒有數目的限制(如FAT32和NTFS)。

檔案配置 (續)

- 檔案配置法 (allocation method)
  - 連續配置
  - 連結配置
  - 索引配置

連續配置 (Contiguous Allocation)

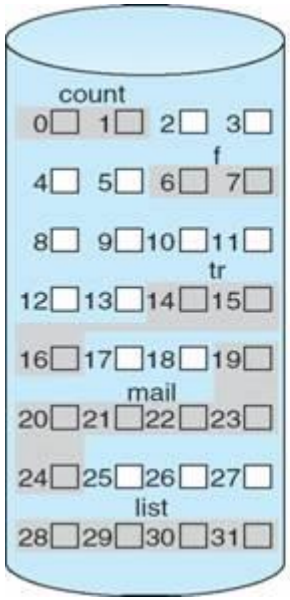
- 將檔案存在連續的區塊內，一旦從目錄區得知檔案的起始位置和大小，就可以到該起始位置 (通常也是某一個區塊的開始位置)，然後讀取連續區塊內的資料，直到所標示的大小為止。
- 有外部碎片問題 (類似連續記憶體配置)。

連結配置 (Linked Allocation)

- 檔案不需要存在連續位置上，從檔案資訊找到第一個區塊後，每個區塊後面會存放下一個區塊的位置，它就像資料結構裡的連結串列 (Linked List)。
- 微軟早期作業系統 MS-DOS 就是使用連結配置來存放檔案，所有磁碟區塊的指標存在一個表格上，這個表格稱為檔案配置表 (File Allocation Table, FAT)。

索引配置 (Indexed Allocation)

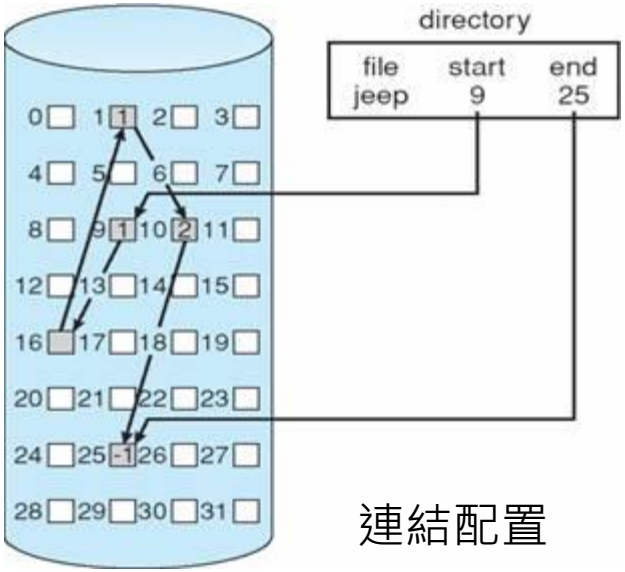
- 概念就像分頁法，每個檔案使用一個類似分頁表的索引區塊 (index block) 作為索引。
- 使用連續配置和連結配置對檔案的存取都是循序的，也就是說第 i 個區塊讀完才能讀到第 $i+1$ 個區塊，因此沒有完全發揮磁碟直接存取的能力。索引配置可以達到直接存取的功能。
- 不會產生外部碎片，但是有內部碎片問題，尤其是小檔案特別嚴重。
- 適合存取大檔案(增加索引區塊的個數)
  - 將索引區塊像連結串列一樣串在一起。
  - 使用多層索引，就像樹狀結構，檔案所指的索引區塊視為第一層索引，它們的指標指到第二層索引，第二層索引區塊指的就是真正的資料區塊。



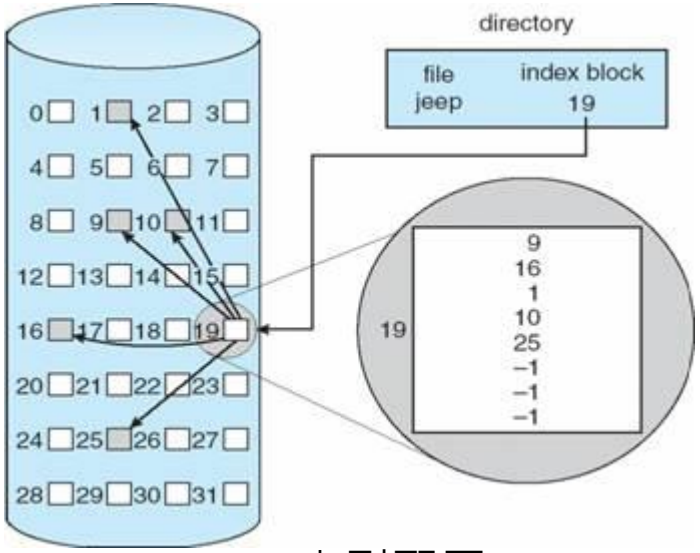
directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

連續配置



連結配置



索引配置

https://www2.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/12_FileSystemImplementation.html

磁碟排班

- 磁碟 (泛指硬碟) 結構： 一般是由數個磁盤所組成，磁盤的二面都可以存放資料，因此每一面都有一個讀寫頭，磁盤以中心軸為中心，劃分出多個同心圓，稱為磁軌 (track)，不同磁盤相同磁軌形成圓柱，稱為磁柱 (cylinder)，每一個磁軌又劃分成數區，稱為磁區 (sector)，磁區就是磁碟存取的最小單位，有時為了快速存取，會將數個連續磁區組成一個邏輯單位，稱為磁簇 (cluster)，它是作業系統存取的最小單位。

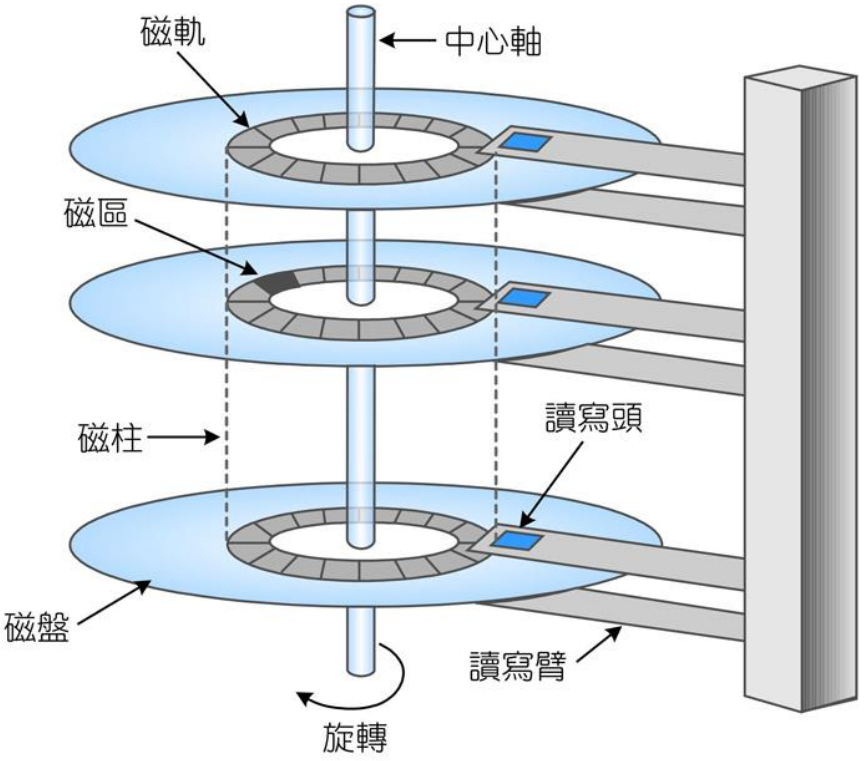
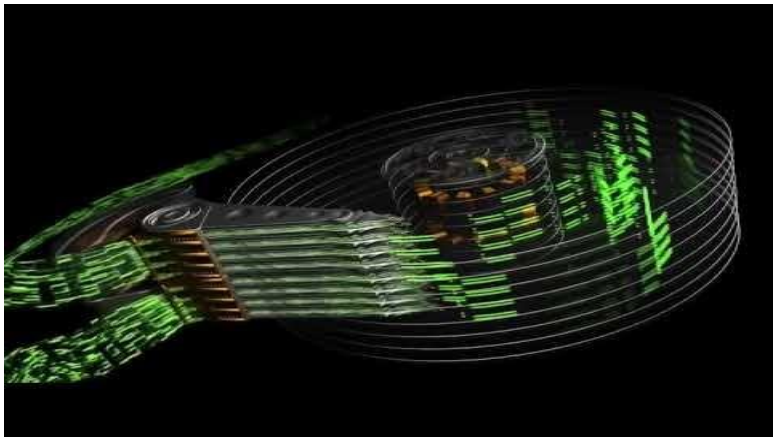


圖 5.11 磁碟的結構

硬碟內部運作方式

硬碟機內部運作



硬碟多讀寫臂運作模式

磁碟排班 (續)

- 讀寫臂伸縮移動到含該磁區的磁軌 (或磁柱) 的時間稱為搜尋時間 (seek time)。
- 磁碟透過旋轉將欲存取的磁區旋轉到讀寫頭下，這個時間稱為旋轉延遲 (rotational latency)
- 當磁區在讀寫頭下方，就可以傳輸資料，這個時間稱為傳輸時間 (transfer time)
- 以搜尋時間最耗時，因為它是屬於機械的移動，所以磁碟的排班，一般都是朝盡量減少讀寫臂的移動次數來改良，而且前提是有很多個磁碟存取的需求進來，這些需求為磁軌或磁柱的編號。

磁碟排班 (續)

- 磁碟排班法則
  - 先到先服務 (FCFS)
  - 最短搜尋時間優先 (SSTF)
  - SCAN
  - C-SCAN
  - LOOK
  - C-LOOK

FCFS 排班

- 先到先服務 (First Come First Serve, FCFS) 是最簡單的排班法，基本上就是沒有任何機制，那個需求先進來，就先處理該需求。
- 假設我們有 0~10 的磁柱，目前位置在 3，依序進來佇列的需求為：5, 9, 2, 6, 1, 7。

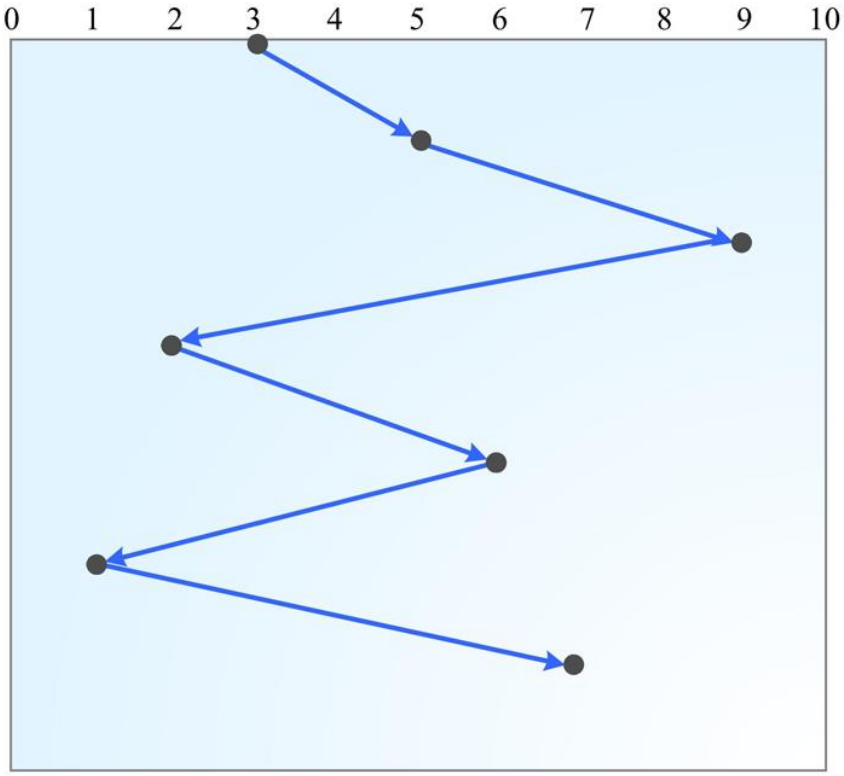


圖 5.12 FCFS 讀寫臂移動的軌跡

SSTF 排班

- 最短搜尋時間優先 (Shortest Seek Time First, SSTF) 和行程排班的 SJF 相似，從目前位置看，那個需求離我最近，我就去服務誰。
- 假設我們有 0~10 的磁柱，目前位置在 3，依序進來佇列的需求為：5, 9, 2, 6, 1, 7。

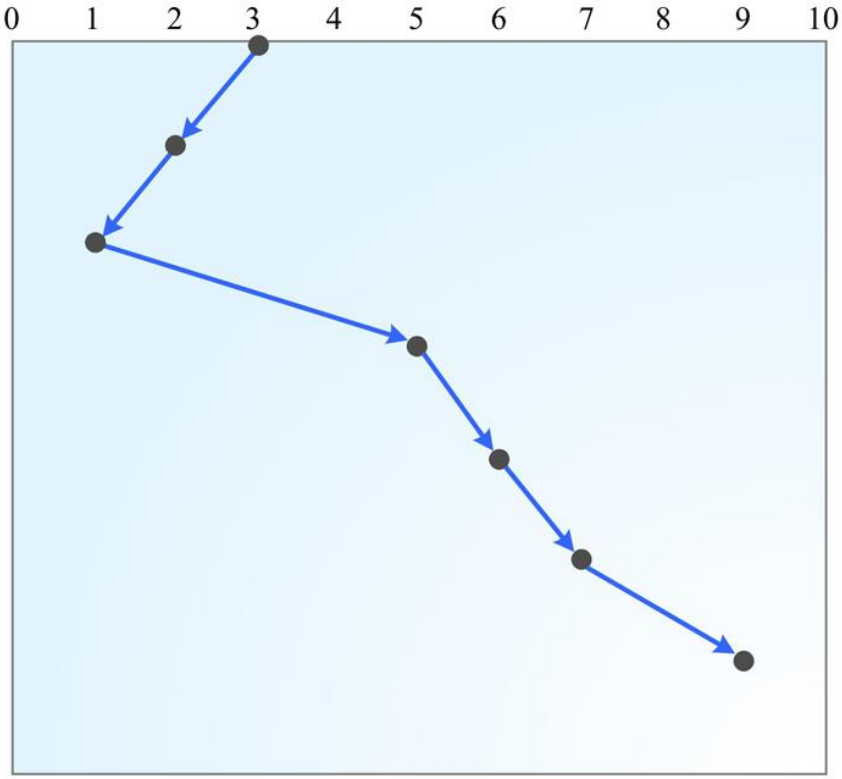


圖 5.13 SSTF 讀寫臂移動的軌跡

SCAN 排班

- SCAN 演算法就像電梯一樣的行為，它有個服務的方向，依照這個方向一直走到底，途中碰到的任何需求，就會馬上服務，到底後再往回走，也是邊走邊服務，如此週而復始，直到所有需求都被滿足為止。
- 假設我們有 0~10 的磁柱，目前位置在 3，往右前進，依序進來佇列的需求為：5, 9, 2, 6, 1, 7。

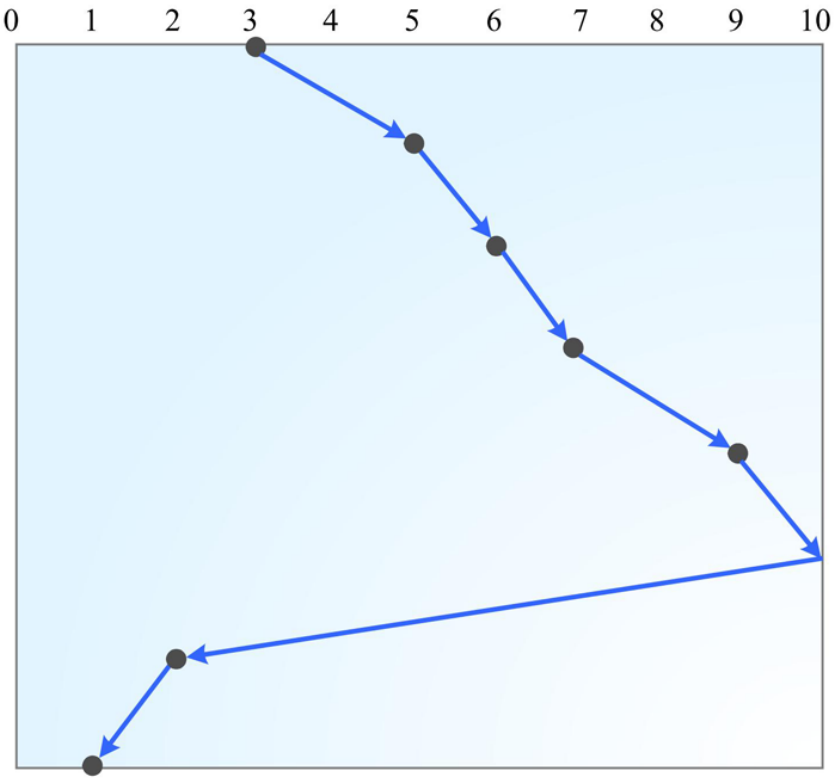


圖 5.14 SCAN 讀寫臂移動的軌跡

C-SCAN 排班

- C-SCAN 是 Circular SCAN 的意思，SCAN 碰到底會往反方向繼續前進服務，而 C-SCAN 碰到底是回到編號 0 的磁柱再重新服務，也就是說 SCAN 是二個方向的服務，而 C-SCAN 是一個方向的服務。
- 假設我們有 0~10 的磁柱，目前位置在 3，往右前進，依序進來佇列的需求為：5, 9, 2, 6, 1, 7。

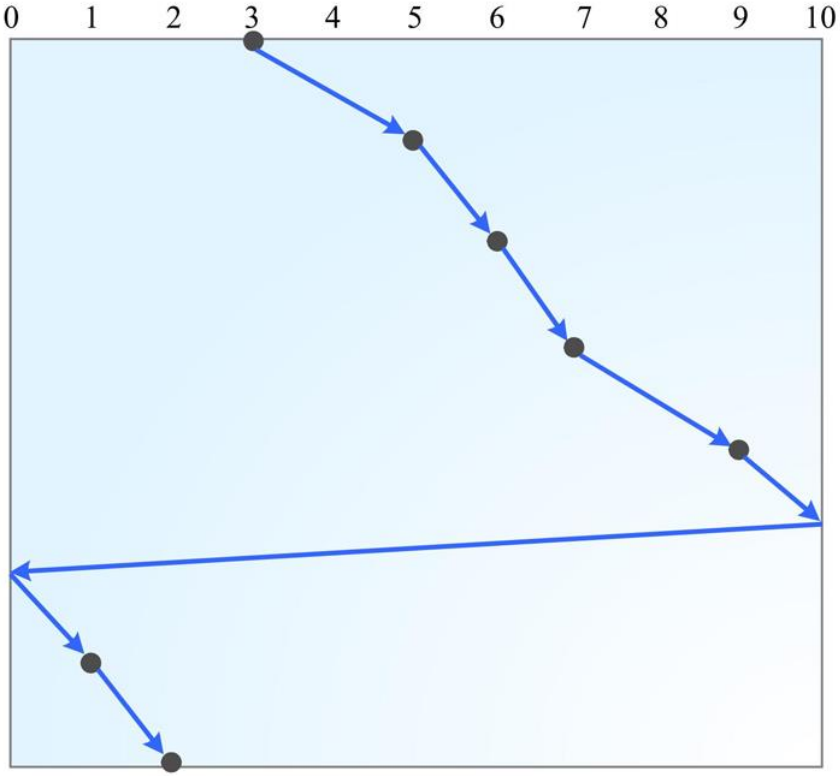


圖 5.15 C-SCAN 讀寫臂移動的軌跡

LOOK 排班

- LOOK 演算法改善 SCAN 一定要走到底的問題，如果在走到底之前能夠先看一下需求佇列，若前面沒有需求就不用到底了，趕快折回去辦正事要緊，因此可以避免多走不需要的路。
- 假設我們有 0~10 的磁柱，目前位置在 3，往右前進，依序進來佇列的需求為：5, 9, 2, 6, 1, 7。

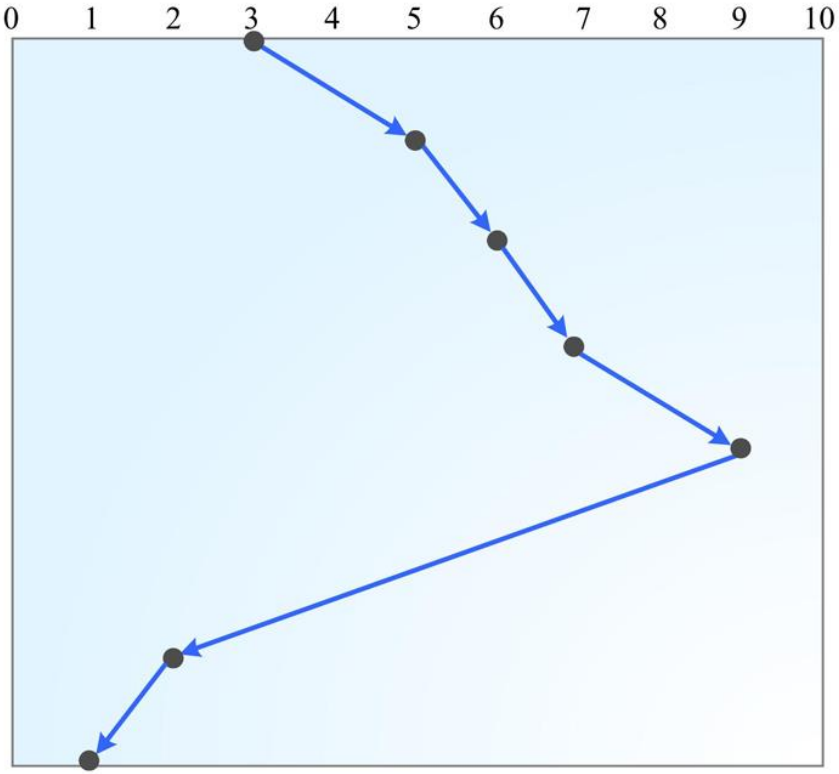


圖 5.16 LOOK 的軌跡圖

C-LOOK 排班

- C-LOOK 演算法改善 C-SCAN 一定要走到底和回到頭的問題，如果在走到底之前能夠先看一下需求佇列，若前面沒有需求就不用到底了，折回到起始點前先看一下需求佇列，因此可以避免多走不需要的路。
- 假設我們有 0~10 的磁柱，目前位置在 3，往右前進，依序進來佇列的需求為：5, 9, 2, 6, 1, 7。

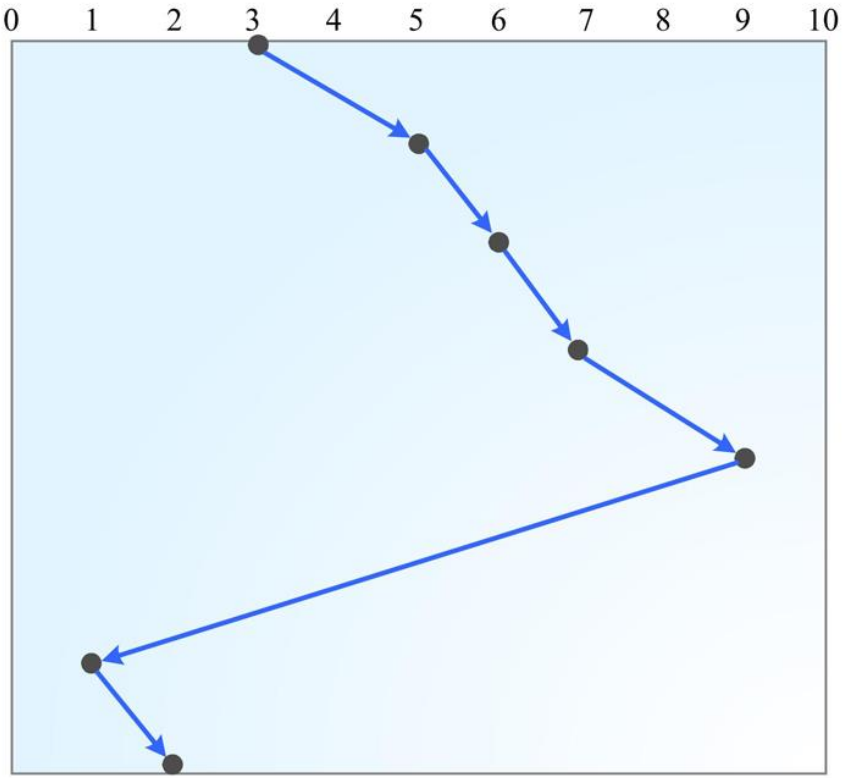


圖 5.17　C-LOOK 的軌跡圖

結 論

- 行程管理、記憶體管理和儲存體管理是作業系統的三個重要的核心設計。
- 另外一個發展趨勢就是物聯網 (Internet of Things)，未來的機器可能都要上網互相聯繫，因此就有必要設計一個簡單的作業系統當作它們聯繫的介面。