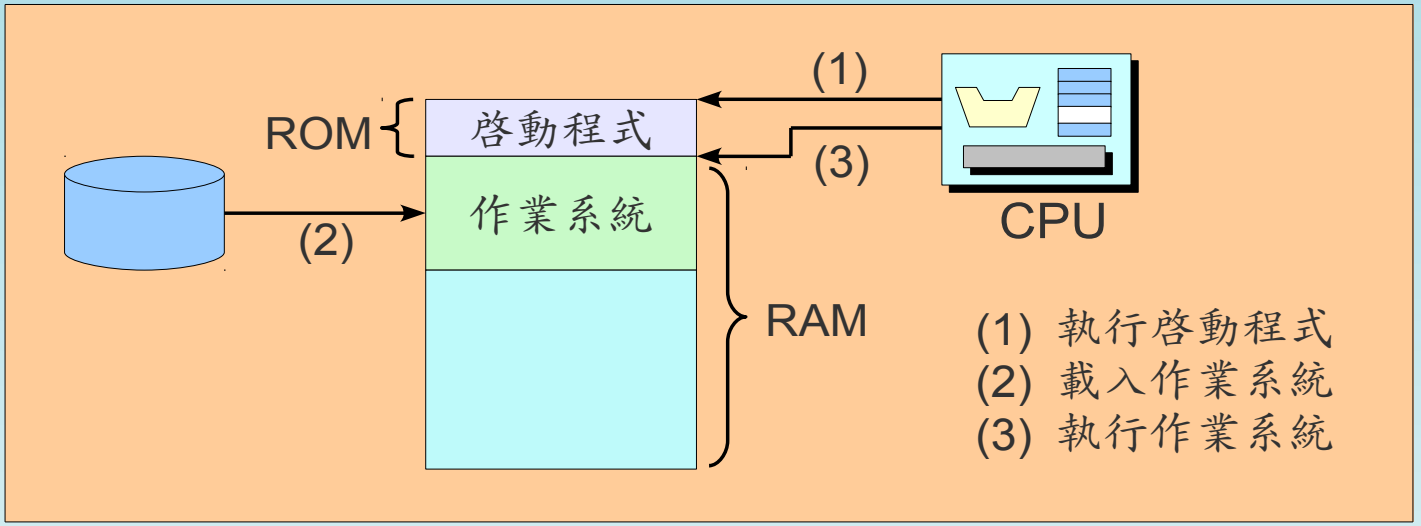
第7章 作業系統

- 作業系統(Operating systems, O.S.)
  * 電腦硬體和使用者(人或應用程式)之間的介面
  * 管理電腦系統硬體及軟體資源的程式
    # 資源：CPU、記憶體、輸出入設備...
  * 協助其他程式執行的程式
- 載入及執行作業系統
  * 執行其他程式時，作業系統將該程式載入記憶體然後執行
  * 作業系統本身的載入及執行
    # 啓動程式(Bootstrap)存在 ROM 中，開機時程式計數器(PC)預設爲啓動程式第一個指令位址，因此首先執行啓動程式

啓動程式負責將作業系統載入記憶體，並將 PC 設爲作業系統的第一個指令位址

作業系統開始執行



(a) 作業系統演進

- 批次系統(Batch system, 1950)

  * 批次作業系統控制大型電腦(Mainframe)，使用打孔卡爲輸入、行列印表機爲輸出、磁帶機爲次級儲存體

* 要執行的程式稱爲工作(Job)
* 電腦以一個一個或一組一組的方式執行工作：批次處理
  # 程式設計師將一疊打孔卡(程式及資料)送至作業室請求執行工作：無法控制電腦，與電腦也沒有互動
  # 作業員處理(讀取)打孔卡並執行程式
  # 如果執行成功，執行結果送給程式設計師；反之，送出錯誤報告
* 電腦資源在工作執行時被佔據
* 作業系統僅確保所有資源會從一個工作轉到下一個工作

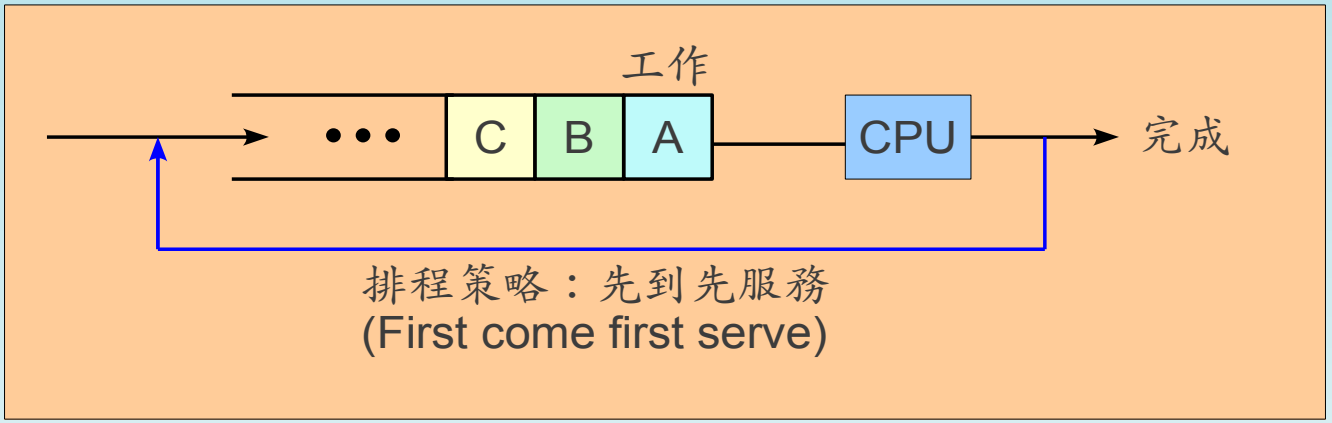
- 分時系統(Time-sharing system)
  * 多重程式(Multiprogramming)：同時執行許多程式
    # 資源可以分享，例如：某工作在執行列印時，CPU 即可讓其他工作使用 → 資源使用效率較高
  * 分時(Time-sharing)：系統資源被不同的工作分享

CPU 在各個工作輪流切換，因此許多工作可以同時進行

每一個工作可分配到一部分時間：時隙(Time-slot)

分時機制非常快速，使用者無法察覺(例如：時隙 ≈ 25 ms)

排程(Scheduling)：將資源配置給不同的程式，決定哪個程式在什麼時候該使用哪個資源



- 個人系統(Personal system)

  * 單一使用者作業系統，例如：DOS

- 平行系統(Parallel system)

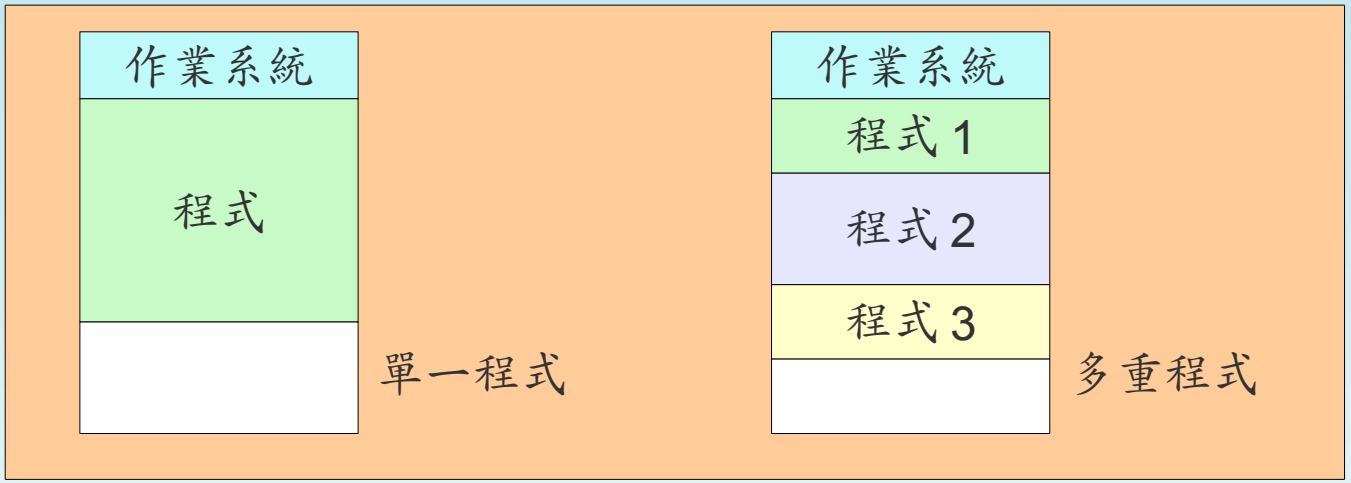
  * 同一部電腦有多個 CPU：眞正並行(True concurrency)

- 分散式系統(Distributed system)
  * 一個工作可以在由網路連結的許多系統上執行
  * 程式、資料、及資源均爲分散
- 即時系統(Real-time system)
  * 可在指定時間內完成任務的作業系統
- 現代作業系統的任務
  * 管理：記憶體、行程、設備、檔案、及使用者介面

(b) 記憶體管理者(Memory manager)

- 記憶體管理者：使記憶體的使用更有效率
- 單一程式(Monoprogramming)
  * 大部分記憶體配置給單一程式及其資料，小部分給作業系統

* 程式執行完畢，記憶體再配置給下一個工作
* 記憶體管理者：載入及執行程式，再將記憶體配置給下一個程式
* 問題：CPU 及記憶體的使用效率低
  # 整個程式需存在記憶體中：程式大於記憶體容量則無法執行
  # 同一個時間僅能執行一個程式
  # 當程式在存取輸出入設備時，CPU 處於閒置狀態



- 多重程式(Multiprogramming)
  * 同一時間可以有多個程式存在記憶體：程式並行執行，CPU 在程

式間快速切換

* 主要技術：

非置換(Nonswapping)：分割(Partitioning)及分頁(Paging)

置換(Swapping)：需求分頁(Demand paging)及需求分段(Demand segmentation)

- 非置換(Nonswapping)：程式在執行期間留在記憶體中，不被置換

* 分割(Partitioning)

記憶體分爲可變長度的區域(分割)，每一個分割存放一個程式

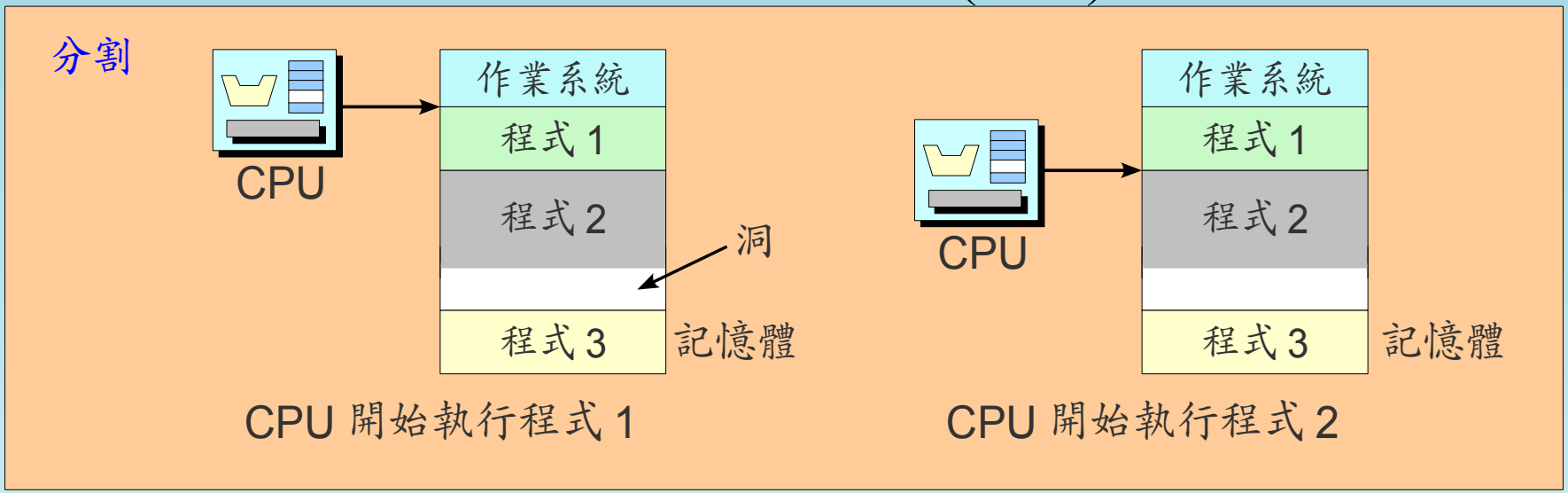
CPU 在這些程式間切換：從一個程式開始執行，直到碰到輸出入要求或時隙使用完畢，然後切換到下一個程式

問題

- 記憶體管理者必須事先決定分割區域的大小

- 如果分割太小，有些程式因無法載入而無法執行

- 如果分割太大，記憶體會產生「洞」(Hole)：分割容量大於程式



* 分頁(Paging)

記憶體分爲同樣大小的區段，稱爲框(Frame)

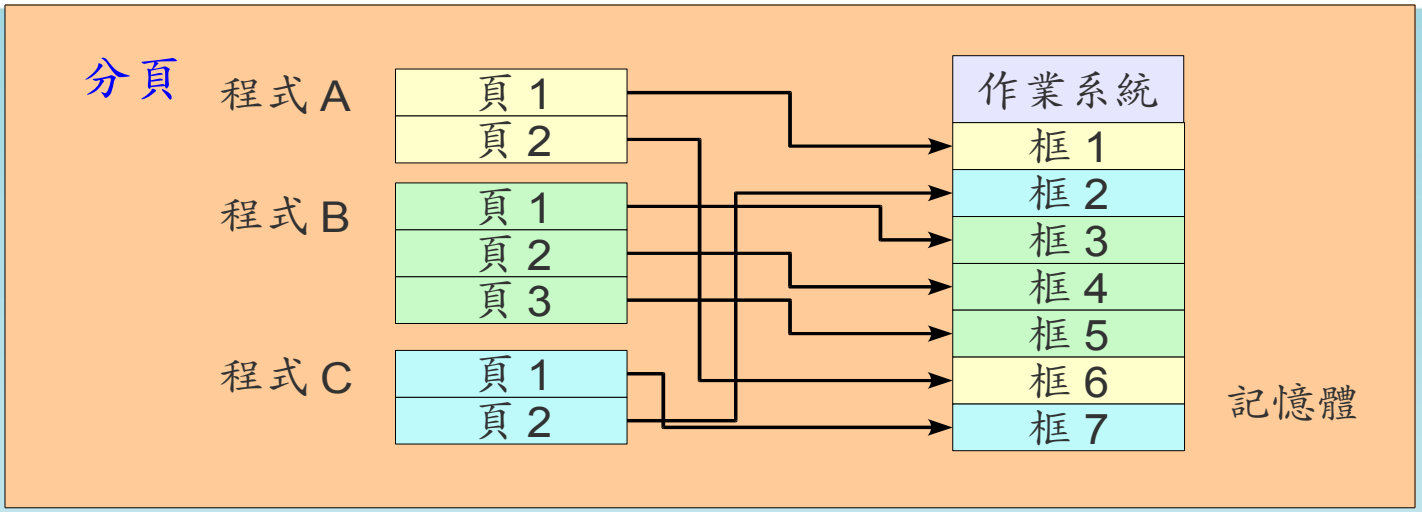
程式分爲同樣大小的區段，成爲頁(Page)

框的大小＝頁的大小，程式中的一頁載入記憶體中的一框

程式的各頁無需存放在記憶體中連續的框

整個程式必須載入記憶體

優點：記憶體的使用較有效率



- 置換(Swapping)：程式執行期間，可以在記憶體及磁碟之間交換

  * 需求分頁(Demand paging)

    # 類似分頁，但程式分頁可以被載入記憶框、被執行、或被其他頁所置換

    # 程式無需全部存放在記憶體中

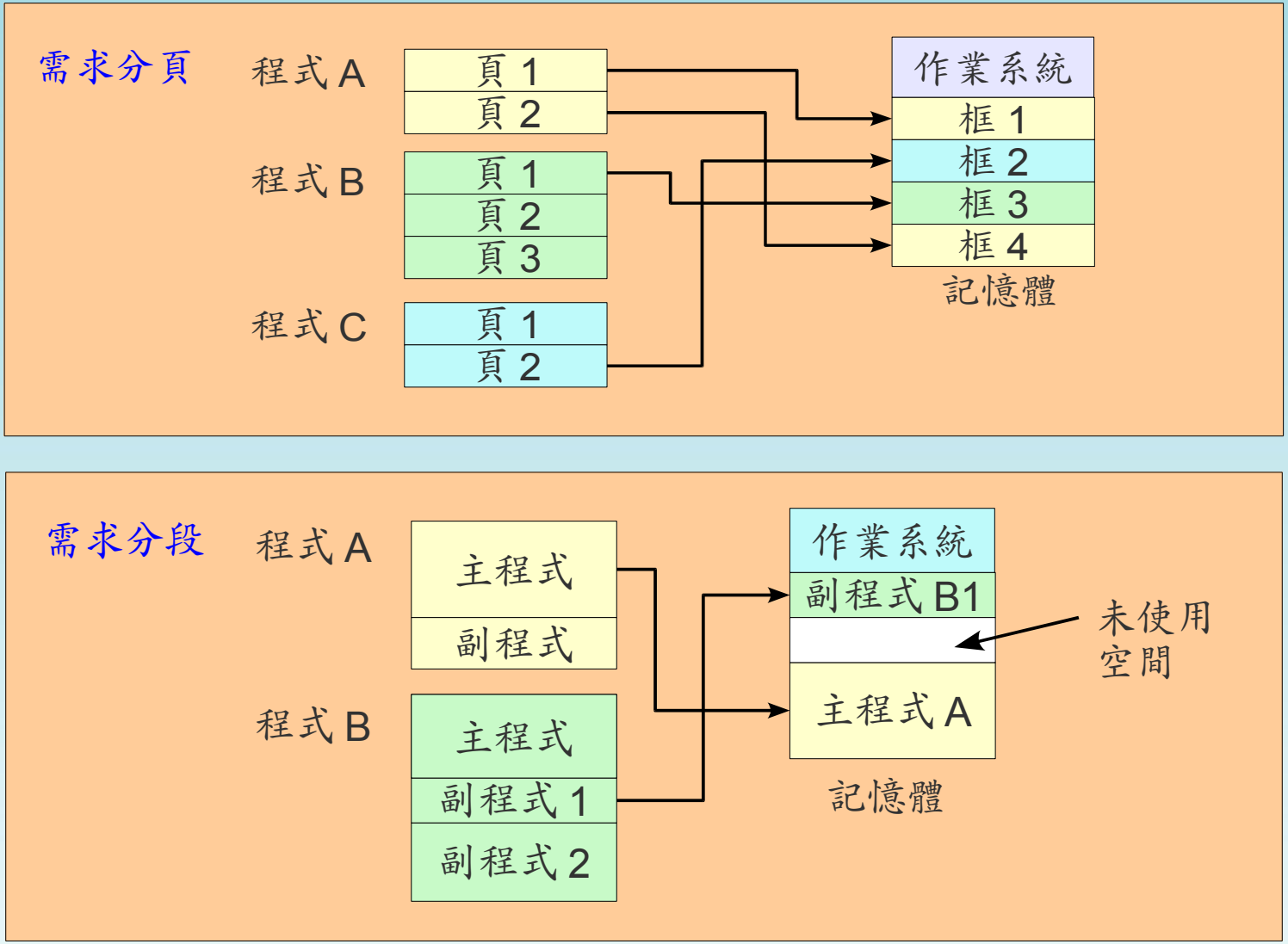
  * 需求分段(Demand segmentation)

    # 程式分爲邏輯單元，稱爲區段(Segment)

      → 依據程式設計師的對於模組(Module)的觀點：主程式、副程式...

    # 程式區段可以被載入記憶體、被執行、或被其他區段或其他程式置換

程式的所有區段無需全部存放在記憶體中



* 需求分頁及分段(Demand paging and segmentation)

程式分爲區段(Segment)：模組(Module)

模組分爲頁(Page)

記憶體分爲框(Frame)

一個模組的各頁可以被載入記憶框、被執行、或被其他頁置換

程式所有頁無需全部存放在記憶體中

- 虛擬記憶體(Virtual memory)

  * 執行中的程式可以部分存放在記憶體中、其餘存放在磁碟中：程式可以比記憶體大 → 記憶體看起來比實際要大

    # 例如：記憶體容量爲 10 MB，共有 10 個程式，每個 3 MB，所有程式同時執行時，記憶體看起來有 30 MB 的容量

  * 現今作業系統無論使用需求分頁、需求分段、或兩者混用，均有虛擬記憶體功能

(c) 行程管理者(Process manager)

- 行程管理者：使 CPU 的使用更有效率

- 程式(Program)、工作(Job)、與行程(Process)

  * 程式：程式設計師所撰寫一群靜止的指令

    # 儲存在磁碟、磁帶、或光碟中

    # 可以變成工作，也可以不用變成工作

  * 工作：程式從被選擇要執行直到執行完畢，就成爲「工作」

    # 在這段期間，一個工作可能被執行，也可能不被執行(等待)

      - 執行狀態：被 CPU 執行中

      - 等待狀態：等待載入記憶體、等待 CPU 執行、或等待輸出入事件

    # 作業系統控管工作，但不控管程式

    # 當一個工作結束執行(正常或異常)，就再次成爲程式

    # 每個工作都是一個程式，但並非每個程式都是一個工作

* 行程：正在執行的程式

亦即：已經開始執行但還沒結束的程式，或已經存在記憶體的工作

→ 所有正在等待的工作中，被選出並載入記憶體中的工作

可能正由 CPU 執行中，或正在等待 CPU 時間

每一個行程都是一個工作，但並非每個工作都是一個行程

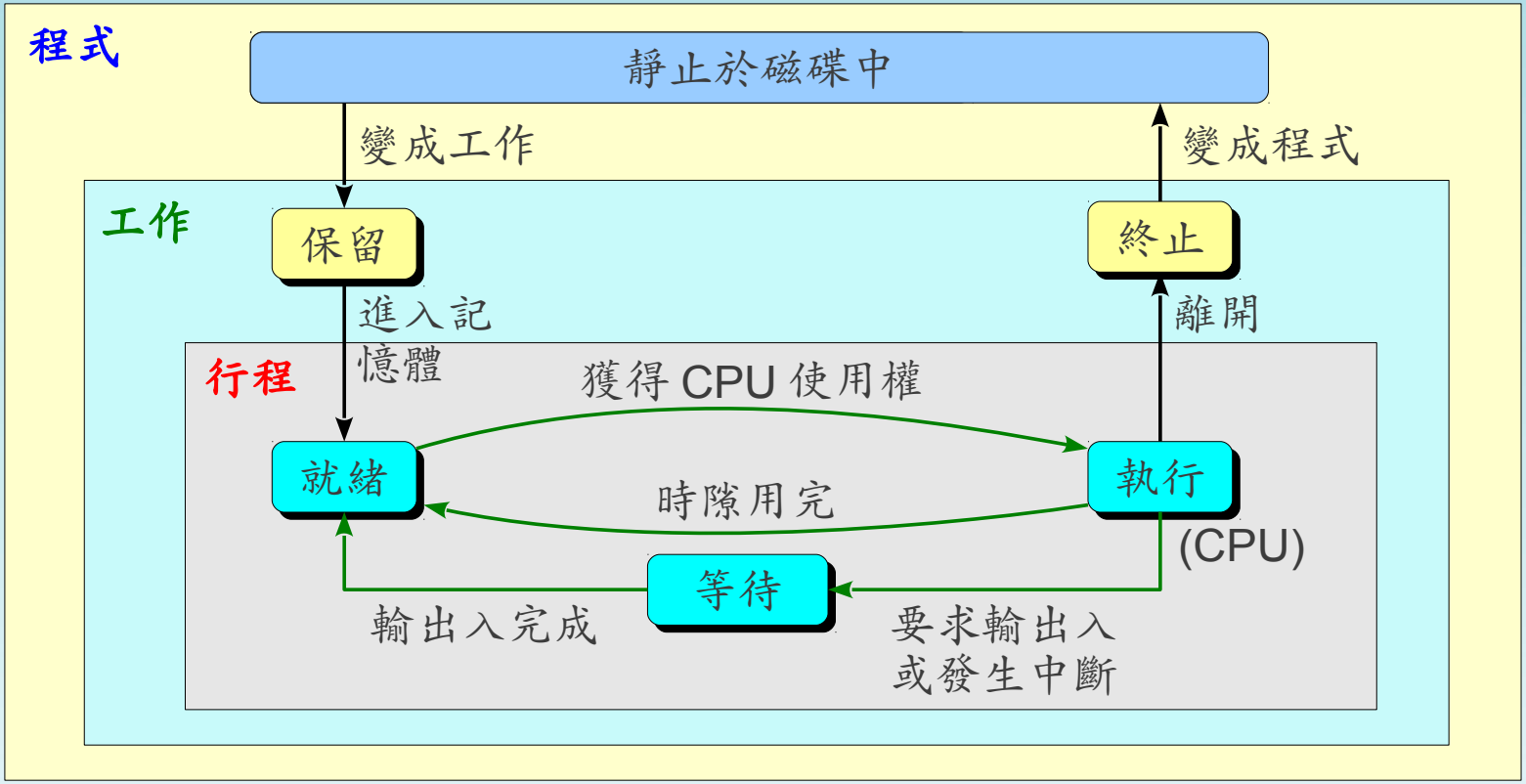
- 狀態流程圖(State diagram)

→ 描述程式、工作、及行程的各種狀態及其之間的關係圖

* 當程式被作業系統選擇預備執行，即成爲工作：進入保留狀態(Hold state)
* 當工作被載入記憶體，即成爲行程：進入就緒狀態(Ready state)
* 就緒的行程取得 CPU 使用權：進入執行狀態(Running state)
* 執行中的行程使用完時隙(Time slot)：進入就緒狀態
* 執行中的行程要求輸出入：進入等待狀態(Waiting state)

* 等待中的行程完成輸出入：進入就緒狀態

* 執行中的行程結束：進入終止狀態(Terminated state)



- 排程器(Scheduler)

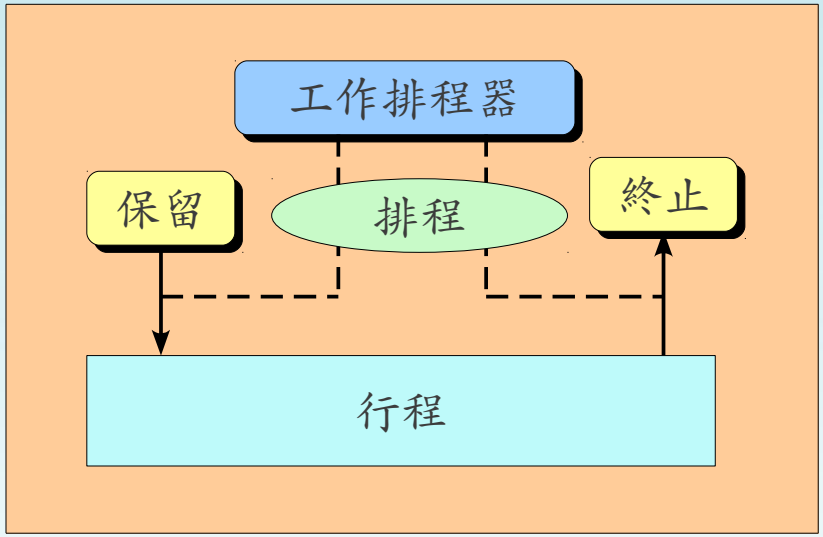
* 行程管理者使用工作排程器(Job scheduler)及行程排程器(Process scheduler)將工作或行程從一個狀態移到另一個狀態

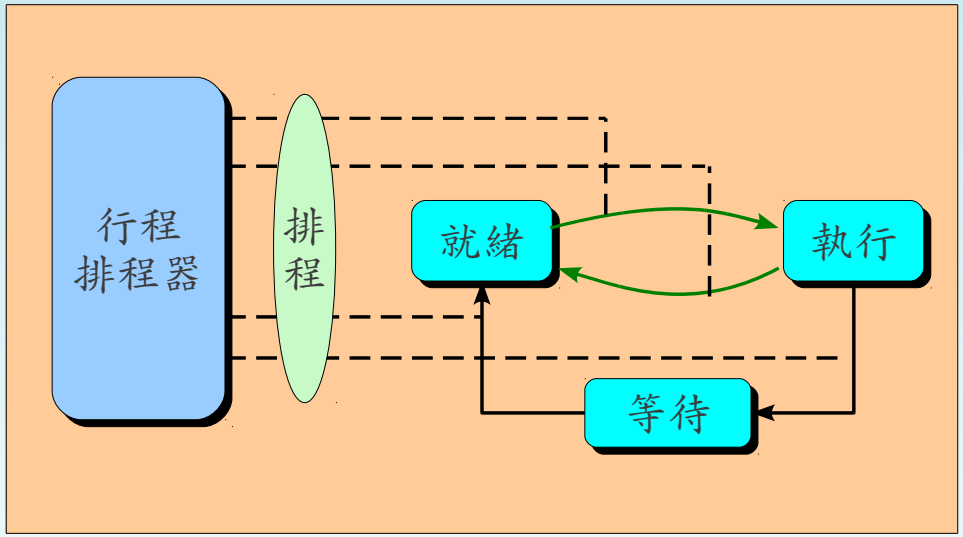
工作排程器：產生或終止一個行程

- 將一個工作從保留狀態移至就緒狀態
- 將一個行程從執行狀態移至終止狀態

行程排程器：將一個行程從一個狀態移到另一個狀態

- 從就緒到執行、從執行到就緒、從執行到等待、從等待到就緒





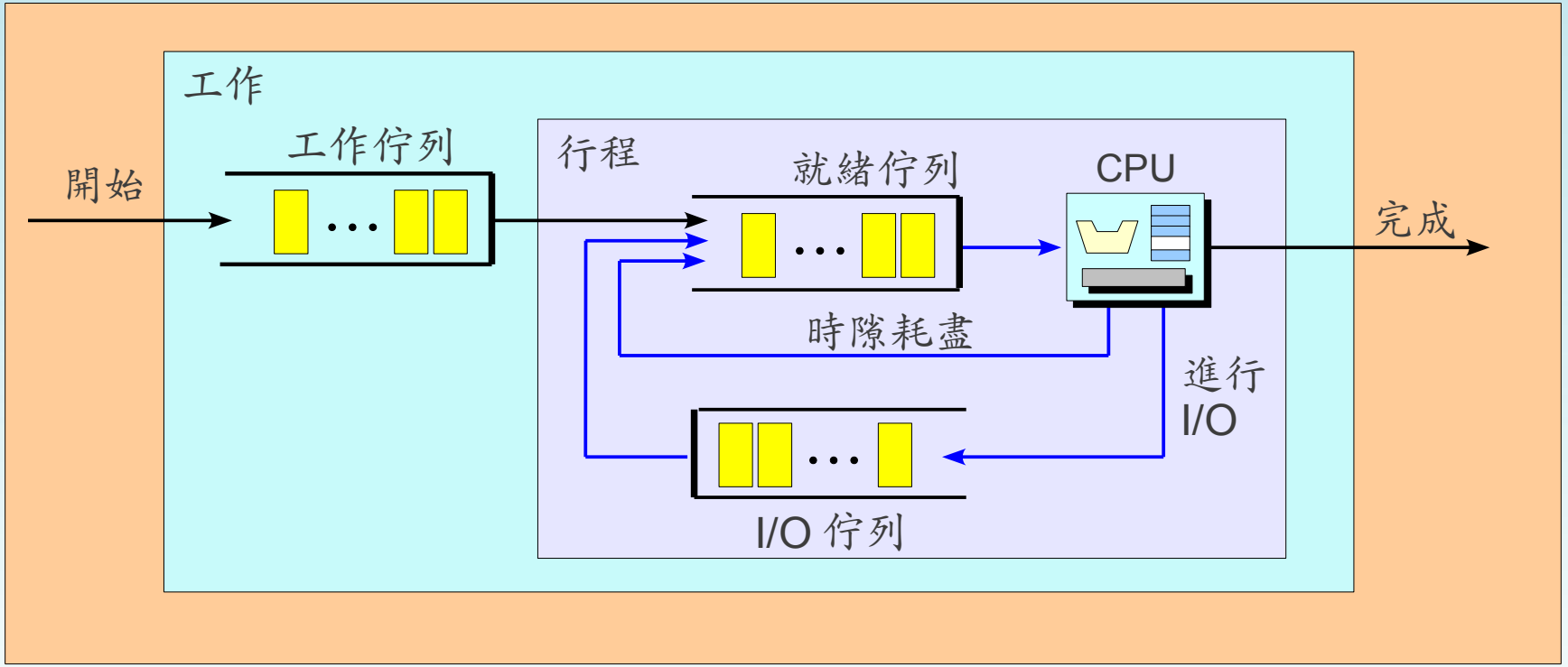
- 佇列排程器(Queuing)

  * 工作或行程彼此競爭電腦資源：CPU、記憶體、輸出入設備、資料...

    → 工作或行程需要排隊等待，作業系統需要排隊策略

  * 行程管理者設置佇列(Queue)，利用不同策略來排序工作或行程

    # 先進先出(First-in-first-out, FIFO)、最短工作優先(Shortest length first)、最高權優先(Highest priority first)...



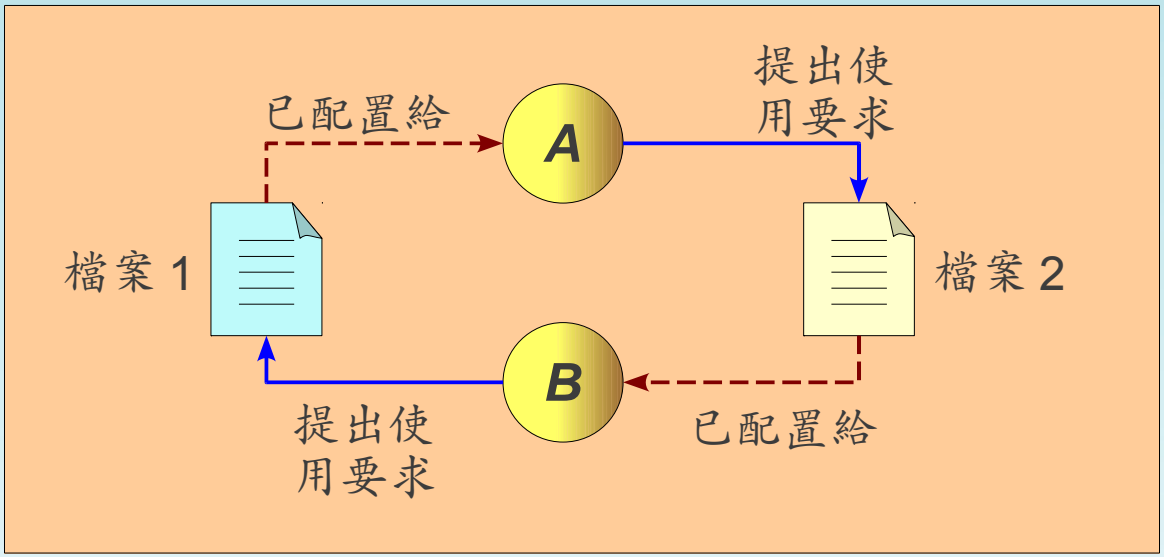
- 死結(Deadlock)：一個行程等待某項資源，但該資源永遠無法取得

  * 例如：

    # 行程 *A* 正在使用檔案 1，而且在取得檔案 2 前不會釋放檔案 1

    # 行程 *B* 正在使用檔案 2，而且在取得檔案 1 前不會釋放檔案 2

    # 檔案不能分享 → 死結發生



  * 死結發生的 4 個必要條件(Necessary conditions)

    # 互斥(Mutual exclusion)：一個資源僅能由一個行程所使用(互相排斥)

    # 資源佔用(Resource holding)：一個佔用某資源的行程正等待被其他行

程所佔用的資源

不可強佔(No preemption)：作業系統不能臨時重新配置資源，使某個行程強佔其他行程的資源

循環等待(Circular waiting)：所有行程與資源形成一個迴路(Loop)

* 上述條件並非充分條件(Sufficient condition)

若發生死結，則 4 個條件必定成立

4 個條件均成立，未必會產生死結

只要某個條件不成立，就不會發生死結

* 死結解決方案

→ 策略：防止一個或多個條件發生，例如：

一個行程必須擁有所需的所有資源後才能開始執行

限制一個行程能擁有某個資源的時間

- 飢餓(Starvation)：一個行程可能因長期無法取得資源而不能執行(並

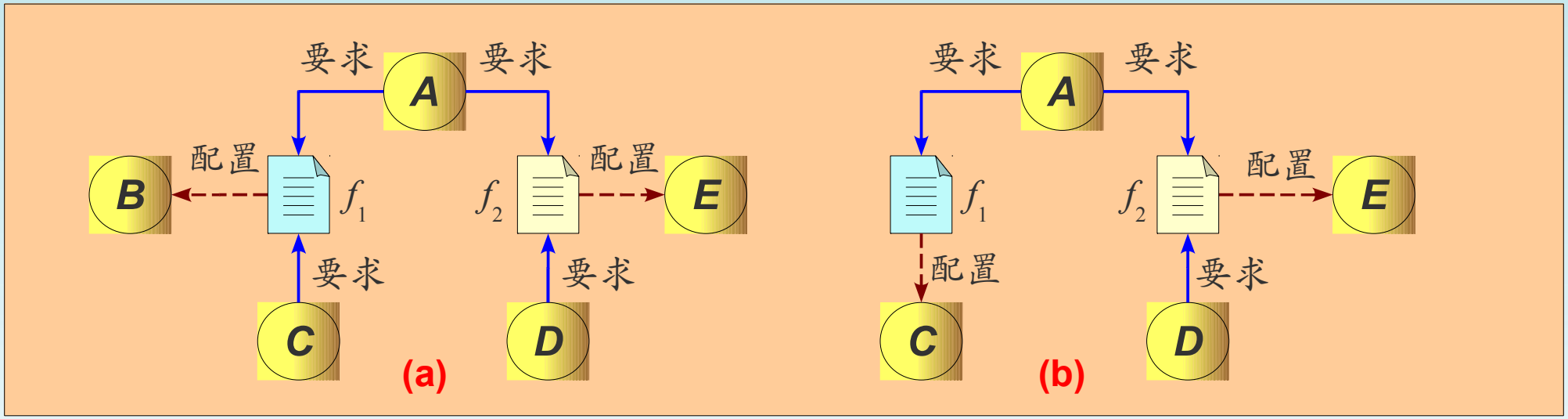
非完全無法取得，但一直沒有機會取得)，例如下圖：

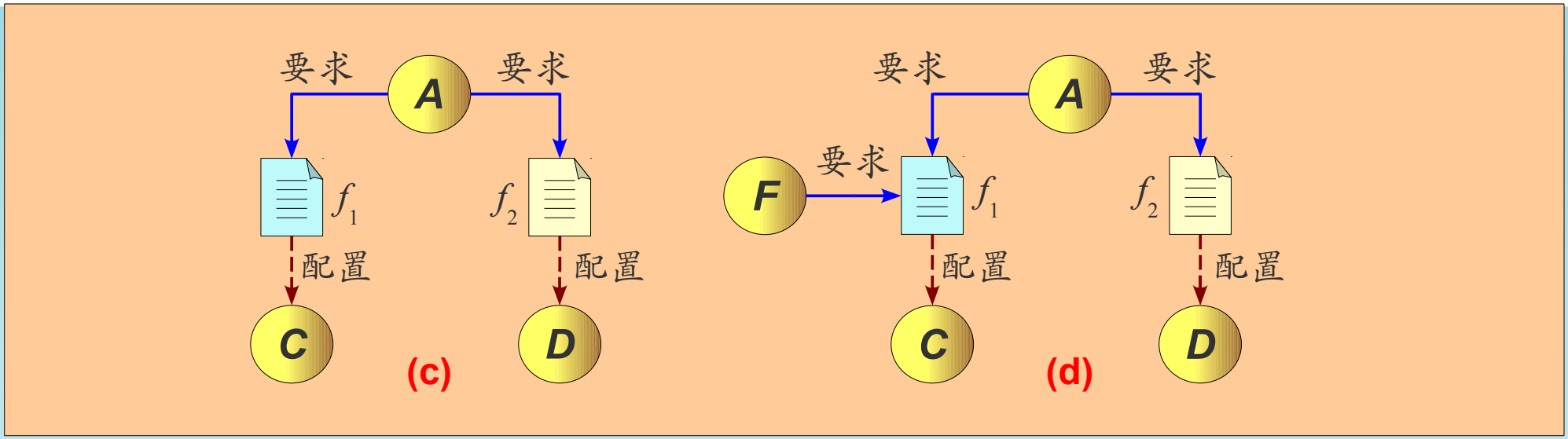
(a) 初始狀態：行程 A 同時要求資源 f_1 及 f_2，行程 B 擁有 f_1，行程 C 要求 f_1，行程 D 要求 f_2，行程 E 擁有 f_2

(b) B 結束並釋放 f_1，A 無法使用 f_1 因爲 f_2 無法取得 → f_1 配置給 C

(c) E 結束並釋放 f_2，A 無法使用 f_2 因爲 f_1 無法取得 → f_2 配置給 D

(d) 一個新的行程 F 可能開始執行而且要求 f_1 ... → A 可能持續飢餓





(d) 設備管理者、檔案管理者、使用者介面

- 設備管理者(Device manager)：使輸出入設備的使用更有效率
  - * 持續監控輸出入設備，確保功能正常
  - * 檢查輸出入設備是否已完成服務
  - * 對每個輸出入設備設置一個佇列，或共同設置一個或多個佇列
  - * 對於輸出入設備設置不同的存取策略

- 檔案管理者(File manager)：控制檔案的存取
  * 依據檔案的存取控制資訊(Access control information)來管制檔案的存取：讀、寫、修改、刪除、執行...
  * 監督檔案的產生、刪除、及修改
  * 控制檔案的命名
  * 監督檔案的儲存(如何儲存、存在哪裡...)
  * 負責歸檔(Archive)及備份(Backup)
- 使用者介面(User interface)：接受並解譯使用者的指令
  * 工具：命令列介面(Shell)、圖形介面(視窗)

(e) 作業系統現況

- UNIX
  * 1969 年由貝爾(Bell)實驗室開發

* 可攜式(Portable)：作業平台移轉非常容易
  # 原因：以 C 語言撰寫，而非特殊語言
* 有強大的公用程式(Utility)
* 虛擬記憶體(Virtual memory)
* 多使用者(Multi-user)及多重行程(Multi-programming)
* UNIX 結構：核心(Kernel)、介面程式、公用程式、應用程式
  # 核心：作業系統基礎部分 → 記憶體、行程、設備、及檔案管理
  # 介面程式：使用者下達指令之介面
  # 公用程式：支援使用者的標準 UNIX 程式 → 文字編輯器、搜尋及排序程式...
  # 應用程式：作業系統標準安裝不包含，大多由第三方所提供的程式

- Linux
  * 1991 年由 Linus Torvalds 所開發

* 以 UNIX 爲基礎的個人電腦作業系統
* 組成單元：核心、系統函示庫、系統公用程式

- Microsoft Windows, Mac OS
  * 分別由微軟(Microsoft Corp.)及蘋果公司(Apple Inc.)所開發
  * 圖形介面(Graphical user interface, GUI)