

# Operating Systems 作業系統

## **OPERATING-SYSTEM STRUCTURES** 作業系統架構

# 作業系統結構

- 系統組成
  - 行程管理
  - 主記憶體管理
  - 檔案管理
  - I/O系統管理
  - 輔助記憶體管理
  - 網路
  - 保護系統
  - 命令直譯程式系統
- 作業系統服務
- 系統呼叫
- 系統結構
- 虛擬機器
- 系統設計
- 摘要

# 系統組成 (System Components)

- 龐大的作業系統可分為幾個較小的**模組**。
- 有的作業系統可能只有實作出其中的幾個模組或部分模組
  - 每個作業系統的**應用領域**不盡相同
  - 系統設計師必須要以**需求**為導向來選擇適當的組成模組。

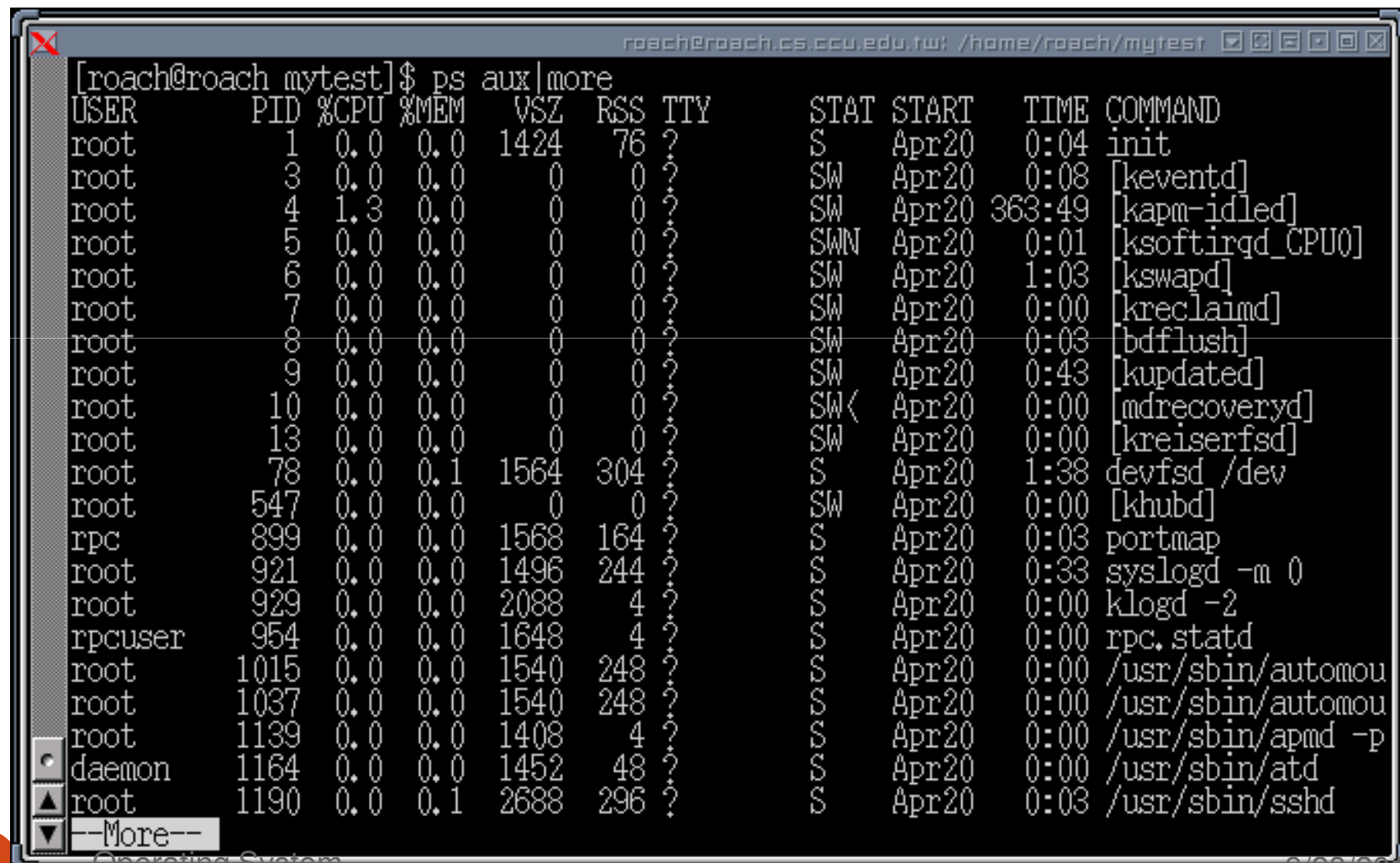
# 行程管理 (Process Management)

- **程式**與**行程**間的關係
  - **程式**被載入到**記憶體**中執行之後就稱為**行程**。
  - **程式**平常只是被動地存在儲存裝置中，並沒有權力去使用系統上的資源。
  - 成為**行程**後，就有主動的控制權去執行程式碼、並使用各種資源來完成工作。
- 行程是電腦系統中的**基本工作單位**

## 行程管理 (續)

- 作業系統對行程管理有下列幾項目標：
  - 管理**使用者**和**系統行程**的**建立**與**結束**。
  - 管理**使用者**和**系統行程**的**暫停**與**再開始**。
  - 提供行程間**同步**的機制。
  - 提供行程間**通訊**的機制。
  - 提供**處理死結**的機制。

# Linux 的 ps 指令



```
[roach@roach mytest]$ ps aux|more
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  1424    76 ?        S      Apr20    0:04 init
root         3  0.0  0.0      0     0 ?        SW     Apr20    0:08 [keventd]
root         4  1.3  0.0      0     0 ?        SW     Apr20 363:49 [kpm-idled]
root         5  0.0  0.0      0     0 ?        SWN    Apr20    0:01 [ksoftirqd_CPU0]
root         6  0.0  0.0      0     0 ?        SW     Apr20    1:03 [kswapd]
root         7  0.0  0.0      0     0 ?        SW     Apr20    0:00 [kreclaimd]
root         8  0.0  0.0      0     0 ?        SW     Apr20    0:03 [bdflush]
root         9  0.0  0.0      0     0 ?        SW     Apr20    0:43 [kupdated]
root        10  0.0  0.0      0     0 ?        SW<    Apr20    0:00 [mdrecoveryd]
root        13  0.0  0.0      0     0 ?        SW     Apr20    0:00 [kreiserfsd]
root         78  0.0  0.1  1564   304 ?        S      Apr20    1:38 devfsd /dev
root        547  0.0  0.0      0     0 ?        SW     Apr20    0:00 [khubd]
rpc         899  0.0  0.0  1568   164 ?        S      Apr20    0:03 portmap
root        921  0.0  0.0  1496   244 ?        S      Apr20    0:33 syslogd -m 0
root        929  0.0  0.0  2088     4 ?        S      Apr20    0:00 klogd -2
rpcuser     954  0.0  0.0  1648     4 ?        S      Apr20    0:00 rpc.statd
root       1015  0.0  0.0  1540   248 ?        S      Apr20    0:00 /usr/sbin/automou
root       1037  0.0  0.0  1540   248 ?        S      Apr20    0:00 /usr/sbin/automou
root       1139  0.0  0.0  1408     4 ?        S      Apr20    0:00 /usr/sbin/apmd -p
daemon     1164  0.0  0.0  1452    48 ?        S      Apr20    0:00 /usr/sbin/atd
root       1190  0.0  0.1  2688   296 ?        S      Apr20    0:03 /usr/sbin/sshd
--More--
```

# 主記憶體管理 (Memory Management)

- 主記憶體是一塊很大的**陣列**
  - CPU 和**週邊裝置**透過**記憶體位址**共同使用主記憶體。
- 主記憶體也是 CPU **唯一**能夠**直接存取**到的**儲存裝置**。
- 複雜的作業系統爲了提高 CPU 的使用率，允許多個程式同時被載入到主記憶體中執行
  - 作業系統必須小心決定程式應該要被載入到主記憶體的哪一塊空間。
- 不同的系統會使用不同的記憶體管理方式
  - 大型的工作站
  - 嵌入式系統

# 主記憶體管理 (續)

- 主記憶體管理所需要管理的事項如下：
  - 記錄哪一塊記憶體位址被哪一個程式使用
  - 決定程式應該載入到哪一塊記憶體空間
  - **分配**和**回收**記憶體空間



# 檔案管理 (File Management)

- 每種儲存裝置都有不同的**儲存方式**和**物理特性**。
- 爲了讓資料的存取不受實體裝置的差異所影響，作業系統定義了一個**邏輯儲存單位**，以及將檔案分類的抽象概念
  - 檔案
  - 目錄
- 作業系統對檔案管理有下列幾個項目：
  - **檔案**的建立與刪除
  - **目錄**的建立與刪除
  - 提供**管理**檔案和目錄的能力，例如複製、移動、和更改檔名等

# I/O系統管理 (I/O System Management)

- 作業系統的任務之一就是建立**使用者**與**週邊裝置**之間的**友善介面**。
- 作業系統是透過**驅動程式**來將複雜的週邊設備隱藏起來，使用者只要合法地使用驅動程式介面，就可以驅動週邊設備。
- 另一個 I/O 重要的概念
  - **裝置即檔案**
- I/O 系統管理包括了：
  - 記憶體管理部分，包含了**緩衝**、**快取**與**週邊並行**（spooling）
  - 裝置驅動程式的**介面**
  - 特定硬體裝置的**驅動程式**

# 輔助記憶體管理 (Secondary-Storage Management)

- 電腦系統中提供了**輔助記憶體**來支援主記憶體
  - 因為主記憶體主要的**缺點**就是斷電後裡面的資料會全部消失。
- 輔助記憶體的速度相當慢，要如何讓存取輔助記憶體的動作有效率便相當地重要
- 作業系統在輔助記憶體管理中負責下列幾個部分：
  - 管理**未使用**的空間
  - 分配空間給需要的檔案
  - 磁碟排程

# 網路 (Networking)

- 網路將許多電腦連結成**分散式系統**
  - 電腦並不**共享**記憶體、週邊設備或是計時器
  - 每一台電腦都有自己的記憶體及計時器，這些電腦之間用各種線路來聯繫
- 在分散式系統中有各式各樣的電腦
  - 連接這些電腦的線路就通稱為通訊網路
- 分散式系統的**好處**
  - 增加計算的速度
  - 資料共享
  - 提高系統可靠度

# 保護系統 (Protection System)

- 作業系統必須提供一些保護機制以提高系統的**可靠度**
  - 行程**不能任意修改**別人甚至是作業系統的**記憶體空間**。
  - 作業系統必須防止**同時存取**一個檔案所會發生的問題。
  - 使用**計時器**來避免一個行程佔用 CPU 的時間太長。
  - 讓使用者無法直接存取週邊設備的控制暫存器，只能透過**系統呼叫**來取得使用權。

# 命令直譯程式系統

- 在電腦系統中，除了作業系統以外，還需要其他重要的系統程式幫忙才能執行使用者所下的命令。
  - 命令直譯程式
  - 圖型使用者介面

# 作業系統結構

- 系統組成
- 作業系統服務
- 系統呼叫
- 系統結構
- 虛擬機器
- 系統設計
- 摘要

# 作業系統服務

- 下列幾項服務是一般作業系統都會提供的：
  - 程式執行
  - I/O 操作
  - 檔案管理
  - 通訊
  - 錯誤偵測
- 還有一些服務是用來提高系統的效能
  - 資源分配
  - 統計
  - 保護



# 作業系統結構

- 系統組成
- 作業系統服務
- 系統呼叫
  - 行程控制
  - 記憶體管理
  - 檔案操作
  - 裝置管理
  - 訊息維護
  - 行程間通訊
- 系統結構
- 虛擬機器
- 系統設計
- 摘要

# 系統呼叫 (System Call)

- 系統呼叫是**使用者行程**與**作業系統**間的**介面**
  - 讓使用者行程可以使用週邊設備
  - 讓使用者行程得到較高的權限來處理工作
- 系統呼叫發生時
  - 產生**陷阱**中斷進入**系統模式**
  - 作業系統取得**控制權**
  - 判斷使用者行程請求的是哪一種系統呼叫
  - 執行使用者行程所需要的服務

# 行程控制 (Process Control)

- 行程**不正常結束**的情況有兩種
  - 使用**系統呼叫**，讓正在執行的行程不正常地結束。
  - 行程做出**不恰當的行爲**，引發例外中斷而使行程不正常地結束。
- 透過系統呼叫，行程可以**載入**或是**執行**其他的程式。
- 對新行程作某些控制
  - 像是設定**優先權**或是**執行時間的最大值**等
- 有些情形會用到讓行程等待的系統呼叫
  - 產生新行程後，等待新行程執行完畢後再取得控制權
  - 讓使用者行程**等待一段指定的時間**後，再繼續執行未完成的工作
  - 讓使用者行程等待一個**事件**或是**訊號**的發生再繼續執行

## 行程控制 (續)

- 行程控制類基本的系統呼叫：
  - 正常及不正常地結束行程的執行，如 `exit()`, `abort()`。
  - 載入並執行一個新的行程，如 `execve()`。
  - 建立新的行程及終止行程，如 `fork()`, `exit()`。
  - 取得及設定行程屬性如 `getpriority()`, `setpriority()`。
  - 等待一段時間，如 `nanosleep()`。
  - 等待事件或是發出訊號，如 `wait()`, `kill()`。

# 記憶體管理 (Memory Management)

- 作業系統必須要提供系統呼叫，**動態地**將記憶體**分配**給**行程**使用。
- 系統呼叫要能將記憶體**歸還**給**作業系統**。
- 作業系統要有管理記憶體的系統呼叫
  - 如鎖定部分**分頁**不作置換
  - 重新對映虛擬記憶體位址
- 還需要提供共享記憶體的系統呼叫。
- 以下是記憶體管理類基本的系統呼叫：
  - 配置及釋放記憶體，如 `mmap()`, `munmap()`。
  - 管理記憶體，如 `mlock()`, `munlock()`。
  - 共享記憶體，如 `shmat()`, `shmdt()`。

# 檔案操作 (File Manipulation)

- 使用者可以透過**系統呼叫**，以檔名來**建立**或**刪除**一個檔案。
- 可以透過系統呼叫打開檔案，並且對檔案做一連串的讀寫，或是**修改**檔案的**屬性**。
- 作業系統也要提供改變檔案讀寫位置的系統呼叫。
- 不再使用這個檔案時，系統呼叫也要能將**檔案關閉**。
- 作業系統還要提供一套類似的系統呼叫來**管理目錄**。

## 檔案操作 (續)

- 以下是檔案管理類基本的系統呼叫：
  - 建立與刪除檔案，如 `create()`, `unlink()`。
  - 開啓檔案與關閉檔案，如 `open()`, `close()`。
  - 讀取檔案與寫入檔案，如 `read()`, `write()`。
  - 改變檔案讀寫的位置，如 `lseek()`。
  - 取得檔案屬性與設定檔案屬性，如 `stat()`, `chmod()`。

# 裝置管理 (Device Management)

- **檔案**與**裝置**有著十分相似的特性，因此處理檔案用的系統呼叫大部分也可以供裝置使用。
- 許多作業系統將檔案與裝置合成檔案裝置結構。
- 以下是裝置管理類基本的系統呼叫：
  - 要求裝置與釋放裝置，如 `open()`, `close()`。
  - 讀取裝置與寫入裝置，如 `read()`, `write()`。
  - 改變讀寫位置，如 `lseek()`。
  - 取得裝置屬性及設定裝置屬性，如 `ioctl()`。
  - 加入與移除裝置，如 `create()`, `unlink()`。



## 訊息維護 (Information Maintenance)

- 作業系統會將所有在系統中執行的**行程**以及**相關的資料**都記錄下來。例如：
  - 系統時間
  - 上線人數
  - 剩餘的記憶體空間
  - 系統的硬體配備
  - 作業系統的版本
- 使用者可以透過系統呼叫將相關的訊息回報給使用者。

## 訊息維護 (續)

- 以下是訊息維護類基本的系統呼叫：
  - 取得時間日期與設定時間日期，如 `gettimeofday()`, `settimeofday()`。
  - 取得系統資料與設定系統資料，如 `sched_setparam()`, `sched_getparam()`。
  - 取得行程、檔案及裝置的屬性，如 `getpid()`, `getppid()`, `ioctl()`。
  - 設定行程、檔案及裝置的屬性，如 `setuid()`, `ioctl()`。

# 行程間通訊 (Interprocess Communication)

- 兩種常見的行程間通訊模式

- **訊息傳遞模式**

- 訊息傳遞模式每次傳遞訊息**都需要經過作業系統**，因此比較沒有效率。
    - 行程間就**不需要考慮同步**的問題。

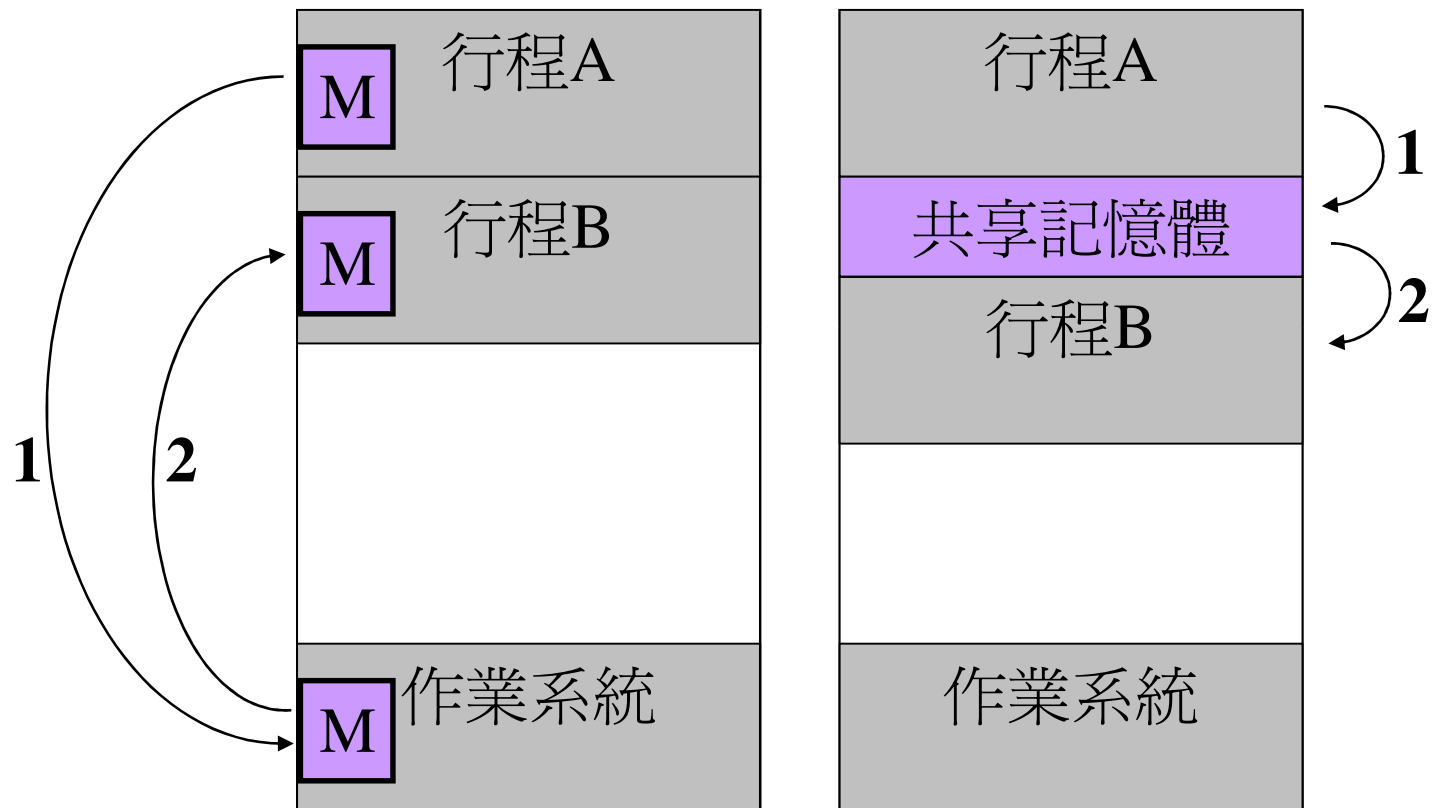
- **共享記憶體模式**

- 存取速度會比訊息傳遞模式快。
    - 必須視情況加入同步的機制。

## 行程間通訊 (續)

- 以下是行程間通訊類基本的系統呼叫：
  - 建立通訊與停止通訊，如 `open()`, `close()`。
  - 收送訊息，如 `read()`, `write()`。
  - 收送狀態及資料，如 `exit()`, `wait()`, `waitpid()`。

# 行程間的兩種通訊模式



# 作業系統結構

- 系統組成
- 作業系統服務
- 系統呼叫
- 系統結構
  - 簡單結構
  - 分層方法
  - 微核心
- 虛擬機器
- 系統設計
- 摘要

# 系統結構 (System Structure)

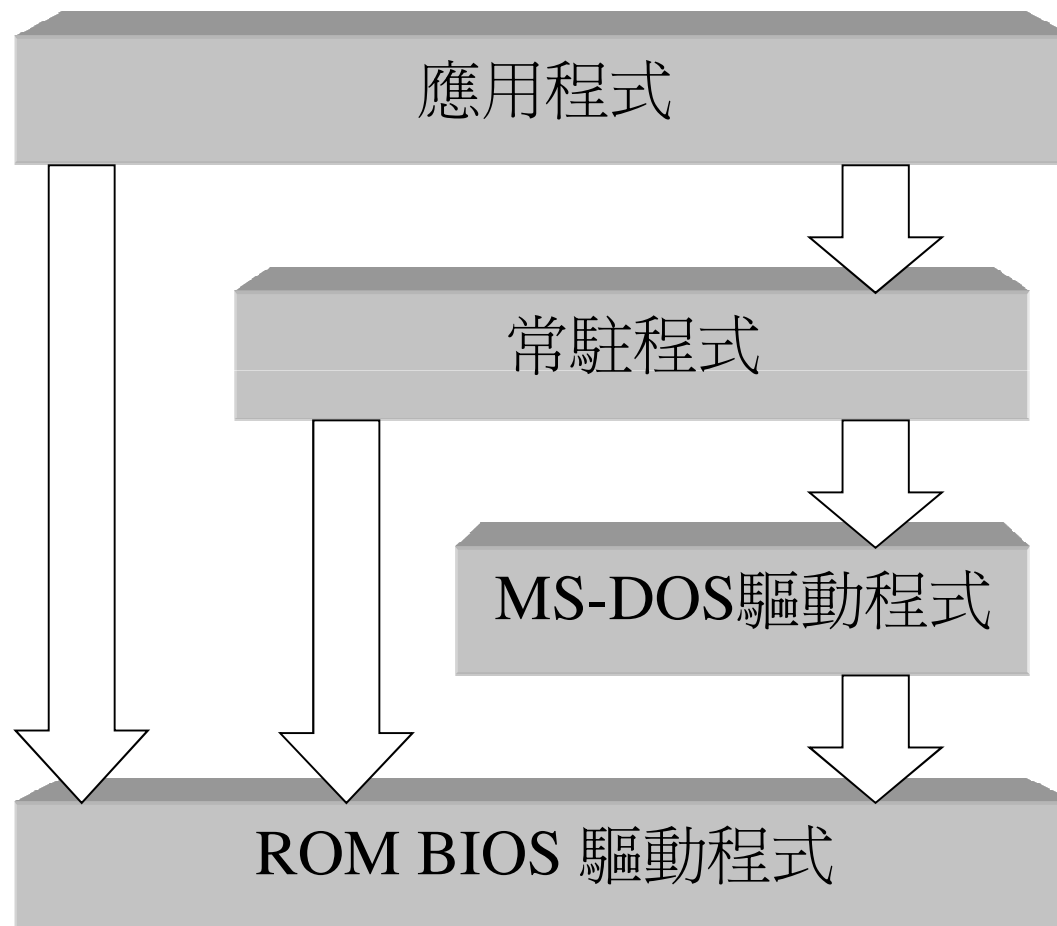
- **維護**與**修改**作業系統是一件龐大又複雜的工程，需要好好規劃。
- 常見的方法是
  - 將作業系統**模組化**
  - 每個模組都要經過周詳的**定義**
  - 形成一個有組織的**系統結構**

# 簡單結構 (Simple Structure)

- MS-DOS
  - 是以建立一個簡單的小系統為目標來實作。
  - MS-DOS 希望以最簡單的方法來提供使用者最多的功能。
  - MS-DOS 並沒有進一步地模組化，其各層程式都可以直接與底層溝通。
  - MS-DOS 的設計者可能沒料想到這個作業系統居然如此地受歡迎，否則要是 MS-DOS 的功能介面及層次分得很好，它可能到現在還受到廣泛使用。



# MS-DOS 系統結構



# 簡單結構

- Unix
  - Unix 系統架構分為**核心**及**系統程式**兩個部份。
  - 隨著 Unix 的改變與擴充，核心又被分成幾個**介面**和**驅動程式**。
  - 新版本的 Unix 以有更適切的硬體支援來設計，因此作業系統可以被分為更小塊、也更適當的模組。
  - 這種方式可以讓作業系統對電腦硬體與應用程式作較佳的控制。
- 一些常用的技術來輔助作業系統的模組化
  - 由上而下的作法

# Unix 系統結構

|                                     |                                    |                                |
|-------------------------------------|------------------------------------|--------------------------------|
| 使用者                                 |                                    |                                |
| 命令直譯程式<br>編譯器和直譯器<br>系統函式庫          |                                    |                                |
| 核心的系統呼叫介面                           |                                    |                                |
| 訊號<br>終端機處理<br>終端機驅動程式<br>字元 I/O 系統 | 檔案系統<br>置換<br>區塊I/O系統<br>磁碟和磁帶驅動程式 | CPU排程<br>分頁替換<br>需求分頁<br>虛擬記憶體 |
| 硬體的核心介面                             |                                    |                                |
| 終端機控制器<br>終端機                       | 裝置控制器<br>磁碟、磁帶、光碟                  | 記憶體控制器<br>記憶體                  |

# 分層方法 (Layered Approach)

- 分層方法是將作業系統**模組化**的方法之一。
  - 例如一個系統若是分爲 N 層的話，最底層（第零層）是硬體，而最高層（第 N 層）是使用者介面，作業系統也是其中一層。
- **分層方法**最主要的好處就是**模組化**，這個方法讓除錯更加容易。
- 分層方法的**缺點**是執行起來會比較**沒有效率**
  - 在每一層往下**尋求服務**時，都會需要**傳遞些資料**或是**參數**，因此系統中的層次越多，對系統的額外負擔也會越大。

# 微核心 (Microkernel)

- 微核心就是很小的核心
  - **巨大核心**內不必要的模組盡量移出，改由**使用者層**或是**函式庫**來處理這些服務。
  - 核心內只剩**最重要的服務**。
- 微核心的**優點**是作業系統**容易擴充**與**維護**。

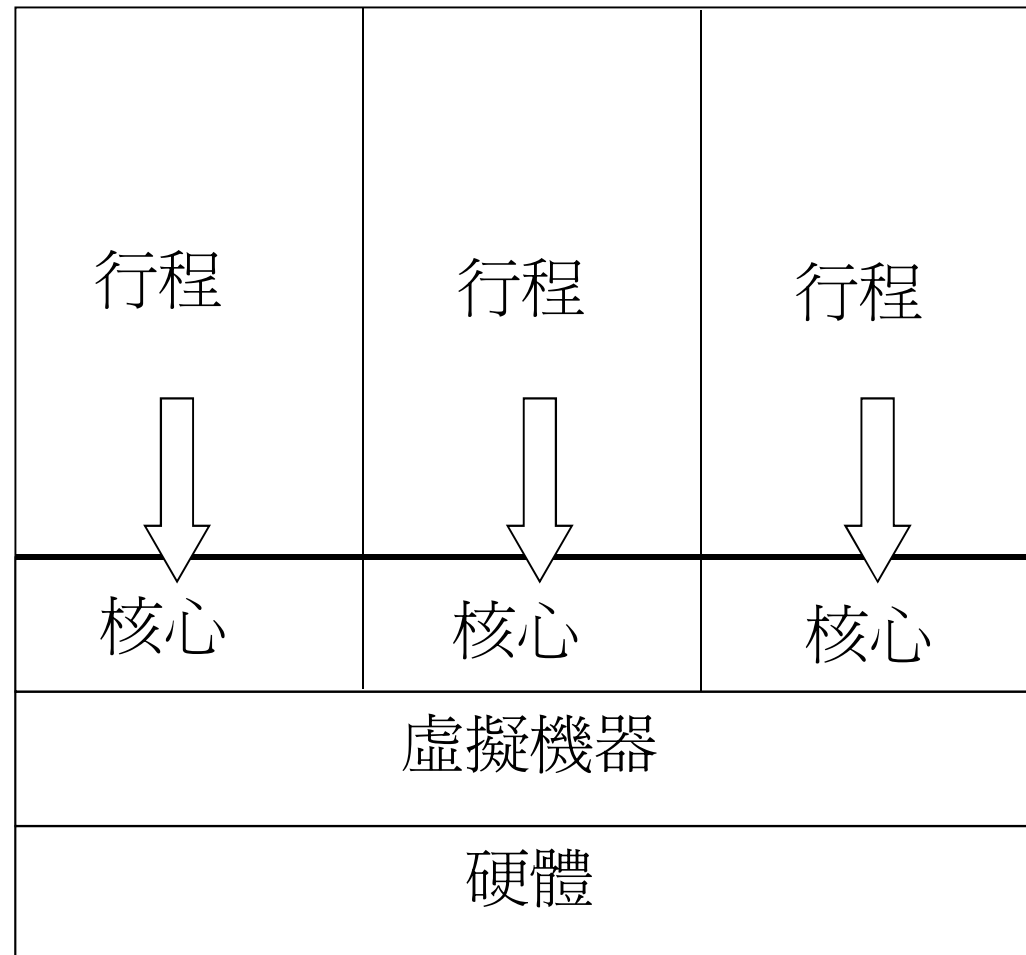
# 作業系統結構

- 系統組成
- 作業系統服務
- 系統呼叫
- 系統結構
- 虛擬機器
  - 實作
  - 優點
- 系統設計
- 摘要

# 虛擬機器 (Virtual Machine)

- 虛擬機器最主要的概念是希望每個在電腦系統中的**使用者**都**認為**自己擁有一整部電腦，而且這部電腦上的資源都由**一個人獨享**，甚至隨時可以重新開機。
- 虛擬機器是以**分享電腦**中的**實體資源**來達成每個使用者都擁有一部真實電腦的**假象**
  - 利用**排程**讓行程分享 CPU 資源，使得行程以為自己擁有 CPU。
  - 利用**週邊並行**和**檔案系統**的作法，提供虛擬的讀卡機和虛擬的印表機。
  - 使用者的終端機則可以提供虛擬機器的控制台。
- 虛擬機器**主要的困難**是發生在**磁碟系統**中
  - 可以在實體磁碟上切割出數個虛擬磁碟供給虛擬機器使用。

# 虛擬機器的系統層次





# 實作

- 虛擬機器要能夠**模擬**出**實體機器**所提供的功能給使用者。
- 虛擬機器和實體機器主要的不同在於執行的時間差異：
  - 虛擬機器在處理 I/O 時是用週邊並行的方式，所以可以節省一些時間。
  - 虛擬機器必須要直譯每個指令，這又會浪費一些時間。

## 優點

- 系統資源可以得到**完全的保護**。
- 虛擬機器也是個很好的**研究**與**發展**作業系統的工具。
- 虛擬機器能讓不同架構的硬體能夠相互執行彼此的程式。

# 作業系統結構

- 系統組成
- 作業系統服務
- 系統呼叫
- 系統結構
- 虛擬機器
- 系統設計
  - 設計目標
  - 機制和策略
  - 實作
- 摘要

# 系統設計 (System Design)

- 在設計一個系統之前，最重要的是先**定義**好系統的**目標**與**規格**。
- 系統的設計會與所選擇的硬體與系統的類型有關。
- 每種系統都有不同的特性以及需要支援的硬體。
- 設計的問題**並沒有一個最佳解法**。

# 設計目標

- 系統設計的需求基本上分為兩類
  - 使用者的需求
  - 系統的需求
- 如何定義作業系統的要求並沒有一定的答案
  - 隨著應用的環境不同，作業系統的需求也會不同。
- 有些一般性的原則可以提出來供作參考
  - 軟體工程

# 機制和策略

- **機制**是指**做一件事的目標或方向**。
- **策略**是指**做這件事的具體方法**。
- 機制與策略分開的例子
  - 微核心
    - 微核心系統的**機制**就是**保留最少的核心功能**
    - **策略**就是以**核心模組**或是**使用者的程式**提供**系統服務**。
  - Linux 作業系統的排程
    - 它的**機制**是以**優先權來排程**。
    - **策略**是系統與使用者都可以**對權重作調整**。

# 實作

- **傳統**的作業系統都是以**組合語言**寫成的。
- **現今**的作業系統也可以用**高階語言**和**組合語言**混合來寫：
  - Linux, MicroSoft ?
- 高階的程式碼來撰寫系統的好處
  - 減少程式開發的時間
  - 程式碼更精簡
  - 容易瞭解
  - 方便除錯
  - 編譯器可以最佳化
  - 容易移植

# 實作 (續)

- 高階語言所實作出來的作業系統有下列**缺點**
  - 執行速度較慢
  - 需要較大的執行空間
- 其他影響速度因素
  - 資料結構
  - 演算法
- 系統設計師要能找出系統效率的瓶頸並想辦法增進
  - 監督系統的運作情形
  - 加入額外的程式來計算與偵測系統的行爲
  - 資料分析
  - 統計出系統各部分的執行效率



# 摘要 (1)

- 系統組成
  - 作業系統可以依照提供給使用者服務的功能性，將系統分為幾個部分。
- 系統結構
  - 尋求較好的系統結構，讓作業系統能夠簡單地被實作出來。
  - 能夠有系統地除錯。
- 虛擬機器
  - 行程擁有獨占整個電腦資源的錯覺。
  - 虛擬機器非常適合用來進行作業系統的開發與除錯。

## 摘要 (2)

- 系統設計
  - 將機制與策略分開對於作業系統設計的彈性有相當大的影響
    - 一點策略的改變可能都會影響到基本的機制運作。
    - 如果採用一般性的機制並且與策略分開設計，只需要稍微修改就可以將系統換成新的策略。
- 實作作業系統
  - 對硬體相關和瓶頸的部分使用組合語言來加強。
  - 其餘的部分則使用高階語言配合正確的資料結構及演算法。