作業系統結構

作業系統服務

- **程式執行**
  - OS 必須能夠將程式載入記憶體執行，並以正常方式終止運行
- **I/O 運作**
  - OS 必須提供介面讓 User Program 間接執行 I/O
- **檔案處理**
  - 讀、寫、建立、刪除檔案
- **通訊**
  - 行程之間交換資訊，**共享記憶體**、**訊息傳遞**
- **錯誤偵測**
- **資源分配**
- **記帳**
  - 紀錄每個使用者用了多少資源、哪些資源
- **保護**
  - 保護 OS 或行程間不相互干擾

一般的系統元件

- 建構作業系統，通常會將系統**分割成較小的部分**進行
- 核心模組(Kernel)
  - ***Process Management** 行程管理
  - ***Main Memory Management** 主記憶體管理
  - **Storage Management** 儲存體管理
    - File Management 檔案管理
    - I/O System Management I/O 系統管理
    - Secondary Management 輔助記憶體管理
  - **Protection System** 保護系統

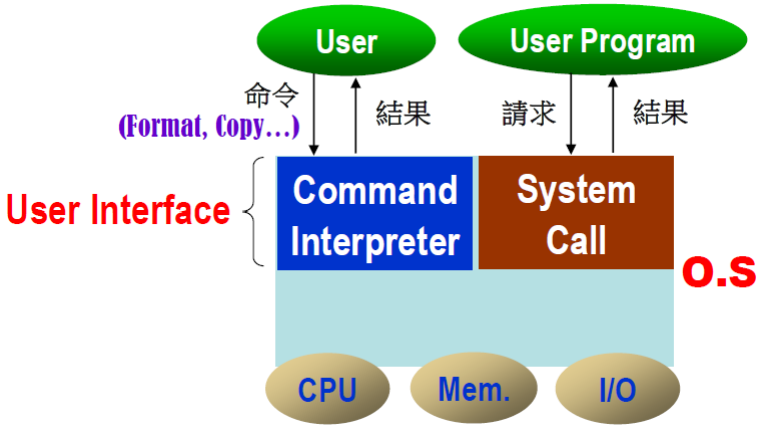
Process Management 行程管理

- Process VS. Program
  - Program **尚未載入記憶體的程式碼，被動的實體**
  - Process **已經載入到記憶體的 Program，主動的實體**
- 行程需要某些特定的資源已完成其工作
  - CPU 時間
  - 記憶體
  - 檔案
  - I/O 裝置

主記憶體管理

- CPU 唯一可以**直接定址與存取**的大型儲存裝置
  - 指令必須在主記憶體中，CPU 才能執行他

OS User Interface



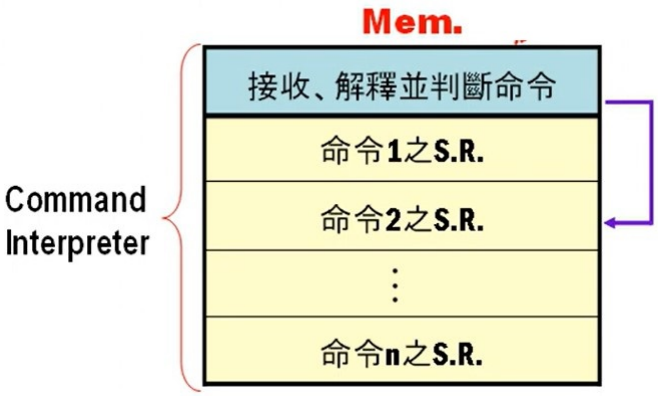
- 所有程式(包括 OS)**User Interface**
- OS 是服務 **User** 和 **User Program**
  - **Command Interpreter --> User**
  - **System Call --> User Program**

Command Interpreter

- User 和 OS 的溝通介面
- 功能
  - **接收**使用者 Input 的命令 (User Command)
  - **解釋並判斷命**令是否正確
    - 不正確：**Bad Command** 訊息
    - 正確：啟動**對應的 Service Routine (Command Routine)**
  - 結果呈現給 User

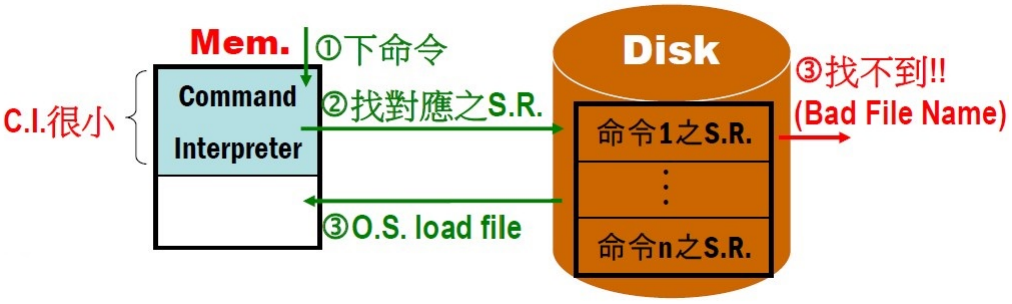
Command Interpreter 是否要包含 Service Routine?

Service Routine 為 Command Interpreter 一部分



- 優點
  - 命令執行速度快
- 缺點
  - 命令數量受限 Command Interpreter 大小
  - 無法增刪命令，彈性度低

Service Routine 與 Command Interpreter 分開



- Service Routine 以**檔案**型式儲存在 **Storage System**(i.e. Disk)
- Command Interpreter 作為 **Loader**
- 優點
  - Service Routine 不會占用大量的記憶體空間
  - 命令增刪容易，且不影響 OS Limit
  - 不同 User 可以自定義 User Environment
- 缺點

- 執行速度慢

Command Interpreter 與 OS Kernal Module 的關聯程度

與 Kernal 合併 緊密結合

- e.g.**Windows**
- 缺點 要變更 Command Interpreter，OS 必須一起變更

與 Kernal 分開 鬆散結合

- e.g.**UNIX**的 Shell 跟 Kernal 是分開的
- Shell 可以任意更改，不會影響到 Kernal，即 User Interface 的變更不會影響到 OS 的 Kernal
- 每個 User 可以建立自己的 Shell
- 缺點 相同的命令格式，在不同的 User Environment 會有不同的解釋，可能造成誤用

System Call

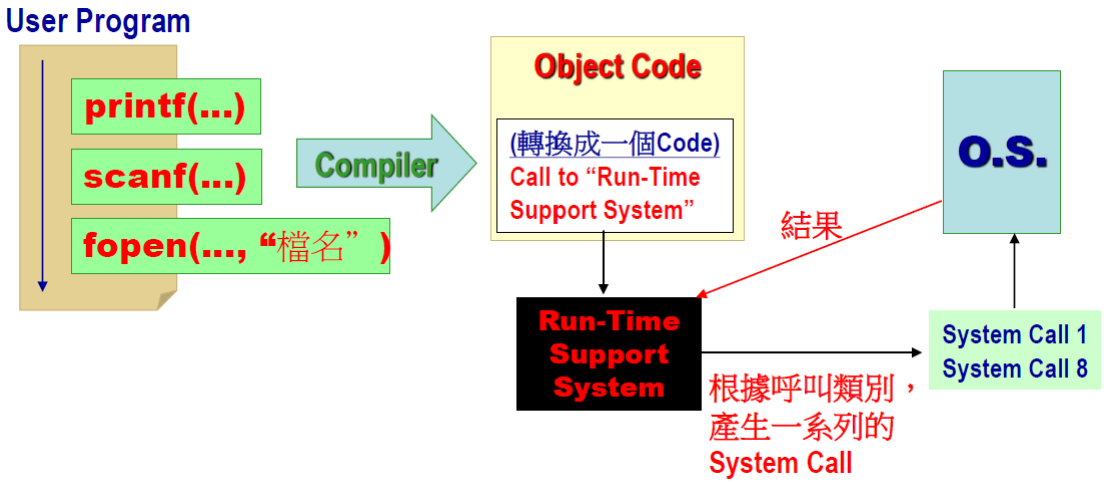
作為 User Program 和 OS 之間的溝通介面

- 當 User Program 執行時需要 OS 提供 Service，則透過 System Call 通知 OS，由 OS 執行相對應的 Service Routine
- e.g. I/O Service
- System Call 會伴隨一個**Trap**，將**User Mode 轉為 Monitor Mode**

System Call 分類

- **Process Control**行程控制
- **File Management**檔案管理
- **Device Management**裝置管理
- **Information Maintenance**資訊維護
- **Communication**通訊

User 通常看不到 System Call



- 透過**In-Line**或**Run-Time Support System**

- **In-Line -->** Compiler 直接將 System Call 加入 Object Code
- **Run-Time Support System -->** Compiler 在 Object Code 加入一個 Code 去呼叫 **Run-Time Support System**
  - Run-Time Support System 根據呼叫類別產生一系列 System Call 給 OS
  - OS 將結果回傳給 Run-Time Support System

System Call 的參數如何傳遞給 OS

- 方法二和方法三較不會限制欲傳遞的參數數量或長度

方法一 利用 Register 儲存參數

- 優點：速度快
- 缺點：不適用參數個數較多的情況

方法二 將參數利用 Table 或 Block 的方式儲存在 Memory 中

- 利用**一個 Register 紀錄 Table 或 Block 的起始位址**傳給 OS
- 優點 適用於參數個數較多的情況
- 缺點 速度慢

方法三 利用 System Stack

- 要存放參數時，將參數 Push 到 **System Stack**，在由 OS 從 Stack 中 Pop 取出參數
  - Stack 可用 Memory 或其他硬體(e.g. Cache)實現

MicroKernel 微核心

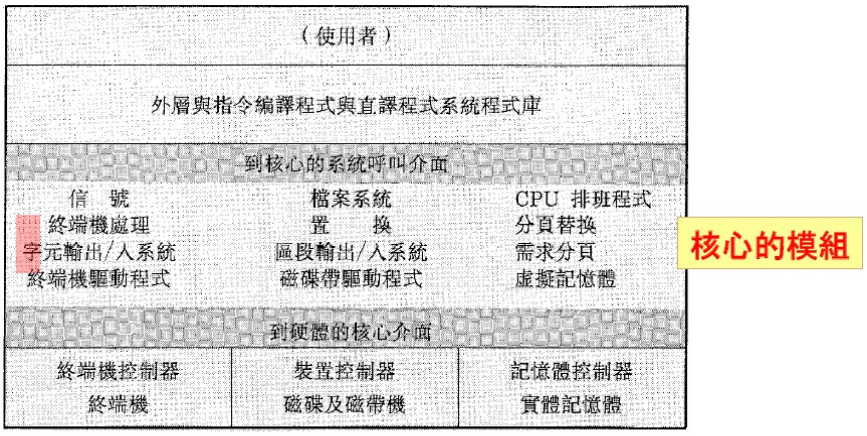


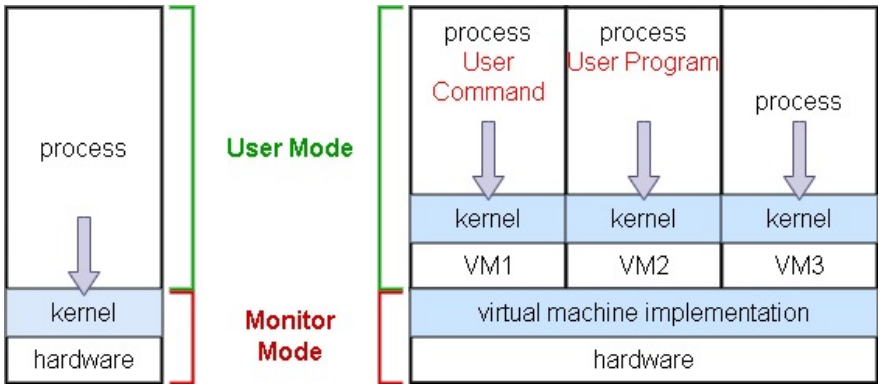
圖 3.7 UNIX 系統結構

- 減少非必要模組，交由 **System Program** 或 **User Program** 來製作，得到**較少模組樹的 Kernel**
- MicroKernel 保留的功能
  - **Basic Process Management**
  - **Basic Memory Management**
  - **Process Communication (e.g. Message Passing)**
- 微核心主要提供在 User Mode 的**Client User Program**和**其他 Server** 之間的通訊

    - 類似 Client Server 架構
        1. User App 用到過去放在 Kernel 的功能模組
        2. Kernel 到 System Library 調出相對應的 Service Routine 執行
        3. Service Routine 完成後將結果傳給 Kernel
        4. Kernel 再將結果傳給 User App
- 優點
    - **OS 容易擴充**
        - 所有 System Service 都是加入到 **User Mode** 執行，不影響 Kernel
        - OS 易於在不同硬體平台轉移
    - **安全性與實用性**
        - 大部分 System Service 在 User Mode 執行，Fail 也不會影響系統其他部分

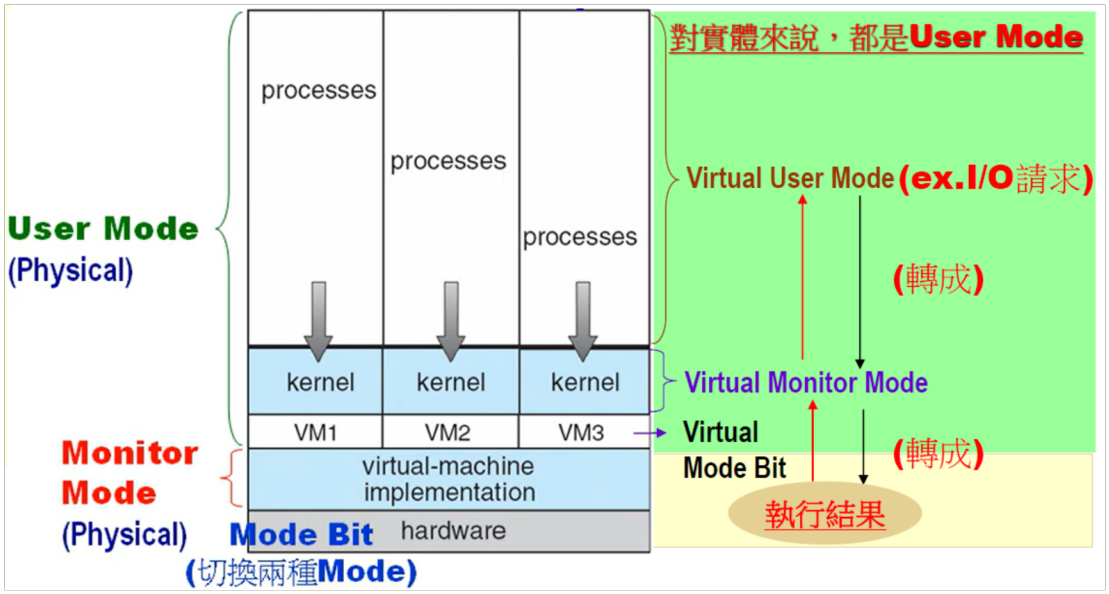
Virtual Machine 虛擬機器

- 透過**軟體模擬**，創造一份與**底層硬體一模一樣的功能介面**
    - 利用 **CPU Scheduling** 技術創造出**多顆 CPU** 的效果
    - 透過 **Virtual Machine** 技術，**擴大 Memory Space**
    - 透過 **Spooling** 技術，提供多套 I/O Device，達到 **I/O 共用**的目的
- **概念：** 每個使用者都認為擁有一整部電腦，且獨享電腦上的資源，可以隨時重新開機
    - 對系統而言，VM 上的 OS 相當於執行中的 User Program
    - 效能低於真正的系統

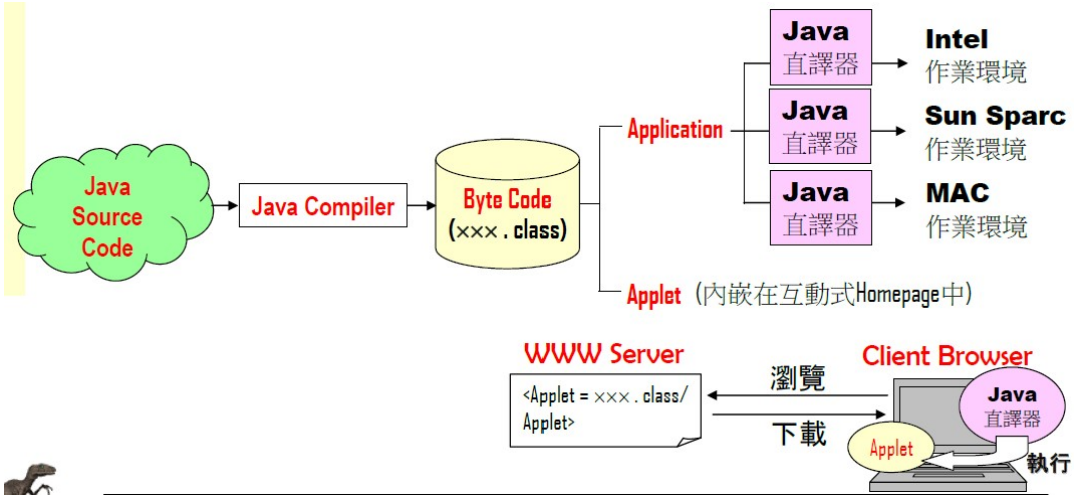


- VM 的軟體是在 Monitor Mode 下執行
- 優點
    - 對 **OS 測試與開發**是良好的平台工具
        - 測試 OS 在 VM 上執行，其他 User 和 User Program 仍可在其他 VM 或實體機器上運行，不會中止
        - VM 是 **Isolated**，所以 OS 開發或測試過程中的不當錯誤，不會對實體系統造成危害
    - VM 具有**安全性**
    - 同一部(實體)機器上**可以執行多套不同 OS**
- 缺點
    - 製作極為困難(要精確複製底層硬體運作非常困難)

    - VM 之間非常 Isolated，不易 Communication 和 Resource Sharing (需借助 Virtual Message Passing 技術)
- VM製作困難的原因
    - 需要從Virtual User Mode切換成Virtual Monitor Mode在切換到Physical Monitor Mode
    - 切換過程困難

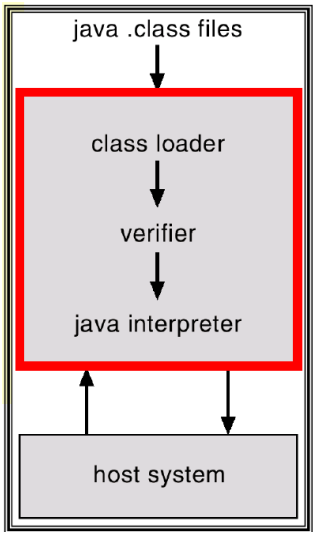


Java Virtual Machine



- 透過編譯器將 Java 程式轉為 byte-code(位元碼)，**與平台無關**
- 透過**直譯器**解譯為能夠在系統上執行的**位元碼**
    - 翻譯一行指令，立即呼叫一行硬體的指令來執行
- Applet 內嵌在互動式 Homepage
    - Client Browser 瀏覽網頁時，會將 byte-code 下在下來，在透過直譯器解譯
    - 減輕 Server 負擔
- byte-code 最大優點是**可跨平台執行**
    - 前題: 平台需支援 **Java Interpreter** 或 **JVM**
- Java 程式事先將原始碼編譯成 byte-code，再將 byte-code 交由**直譯器**或**瀏覽程式**執行
    - **直譯器**或**瀏覽程式**就是 Java 虛擬機器 (JVM)

- byte-code
  - 可是為 JVM 的**Object Code**
  - **機器獨立性**，可跨平台執行
  - **高度可攜性**
- Java 虛擬機指的是在作業系統或瀏覽器上執行的一種程式，可以**解讀 Java 的 byte-code**，並在作業系統的幫助下執行 byte-code



- JVM 包含
  - class loader 類別載入器
  - class verifier 類別檢查器
    - over flow
    - under flow
    - no pointer
  - Java interpreter 直譯器
- JVM 藉由**Garbage Collection**自動執行記憶體回收
- **Just In Time Compiler (JIT)** 提高程式執行速度
  - JVM 多重防護措施確保系統安全穩定，但造成執行效能低落
  - **JIT compiler** 將 Byte Code 即時翻譯成 Target Machine 上的 Object Code，改變 Java Interpreter 的效率
  - JIT compiler 著重於**減少翻譯時間**，**產生有效率的可執行碼**
    - 最佳化
    - 翻譯過的程式碼不須再翻譯一遍

系統呼叫的類型

行程的控制 Process Control

- 正常結束，中止程序(end, abort)
- 載入，執行(load, execute)
- 建立行程，終止行程(create process, termniate process)

- 取得行程屬性，設定行程屬性(get process attributes, set process attributes)
- 等待時間(wait for time)
- 等待事件，顯示事件(wait event, signal event)
- 配置及釋放記憶體空間(allocate and free memory)

檔案的管理 File Management

- 建立檔案，刪除檔案(create file, delete file)
- 開啟，關閉(open, close)
- 讀出，寫入，重新定位(read, write, reposition)
- 獲取檔案屬性，設定檔案屬性(get file attributes, set file attributes)

裝置的管理 Device Management

- 要求裝置，釋放裝置(request device, release device)
- 讀出，寫入，重新定位(read, write, reposition)
- 獲取裝置屬性，設定裝置屬性(get device attributes, set device attributes)
- 邏輯上的加入或移出裝置(logically attach or detach devices)

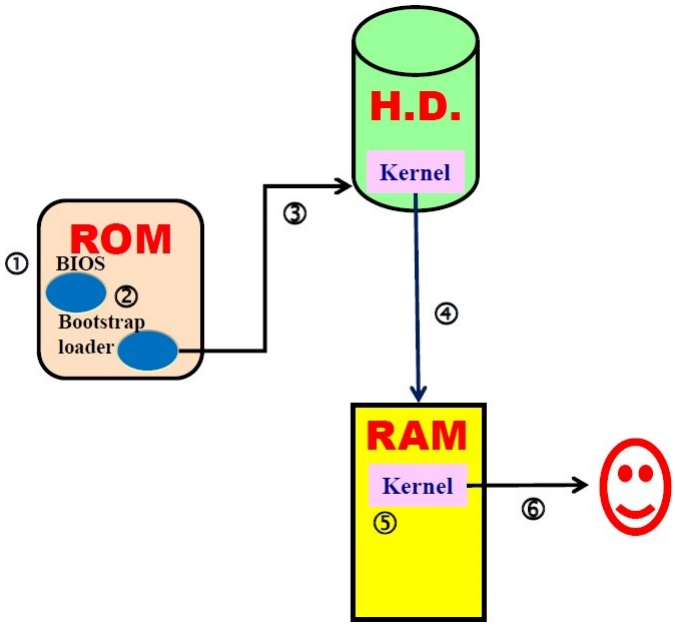
資訊維護 Information Maintenance

- 取得時間或日期，設定時間或日期(get time or date, set time or date)
- 取得系統資料，設定系統資料(get system data, set system data)
- 取得行程，檔案或裝置的屬性(get process, file or device attributes)
- 取得行程，檔案或裝置的屬性(set process, file or device attributes)

通信 Communication

- 建立檔案，刪除檔案(create ,delete communication connection)
- 傳送，接收訊息(send, rerceive messages)
- 傳輸狀況訊息(transfer staatus information)
- 連接獲分離遠程裝置(attach or detach remote devices)

開機程序



1. 執行 **BIOS** 以進行各硬體裝置測試和初始設定
2. 執行**靴帶載入程式 (Bootstrap Loader)**
3. 由 Bootstrap Loader 找到 OS 的 Kernel 在哪裡
4. 將 Kernel 載入 RAM (Monitor Mode)
5. 利用 Kernel 執行系統初始化工作
6. 初始化完畢，系統控制權轉移到 User Mode

BIOS (Basic Input / Output System)

- 電腦開機時必須調用的內建程式
- 功能：
  - 提供電腦個硬體周邊裝置的中段服務
  - 對各周邊裝置進行初始設定與基本檢測工作 (即：電腦啟動自我檢測 Power On Self Test, POST)，保證電腦能正常運作
    - e.g. 確認實體記憶體 RAM 大小和各硬體組件，ex. 硬碟、鍵盤、顯示器、磁碟機...等設備是否存在或可以正常運作
    - 若發現設備有問題就會發出特定的警報聲音

Bootstrap Loader

- 小型程式，於開機時執行
- 存放在 ROM
- 用來找到 OS Kernel，使系統可以順利完成開機過程