# OS_110_CH2

## OS services

### User Interface

- CLI ( Command Line Interface )
  - 用命令的方式操作
  - Shell : Command-line interpreter 解釋器
    - ex. CSHELL, BASH
    - Shell 不僅用於處理單行的指令，也有內建程式語言的功能
- GUI ( Graphic User Interface )
  - 透過圖形

### Communication Models

- 透過 memory 溝通
- 分為兩種
  - message passing
    - 透過 system call 將資料複製到 kernel 後，對方再到 kernel 拿
  - shared memory
    - 透過 system call 分配一記憶體空間 shared memory，雙方都可以讀取

## OS-Application Interface

### System Calls

- The OS interface to a runing program
- 是 software interrupt，所以一定是 kernel mode
- assembly-language instructions 彙編語言指令
- ADD $s1 $s2 $s3

### API

- 使用者用程式 API 去執行 System call

- 普遍實施為一個 library
- ex. ls, dir, cd
- API 本身不會有 interrupt，是裡面呼叫的 System call 才會有 interrupt

## Example

- User program

```c
printf("%d", exp2(int x,int y));
```

- Interface

```
int exp2(int x, int y);
i.e. return the value of x * 2^y
```

- Library

```
lmp1: int exp2(int x, int y) {for ...}
lmp2: ...
lmp3: ...
```

## API

- Win32 API for Windows
- POSIX API for POSIX-based system
  - Portable Operating System Interface for Unix
- Java API for the Java virtual machine

API → System Call → OS

## Why use API?

- Simplicity 簡單
- Portability 可移植性
- Efficiency 有效率

## System Calls: Passing Parameters

- 有三種模式

    - registers
    - table in memory
        - 用 pointer
    - stack

# OS Structure

## User goals and System goals

- User
    - easy to use
    - easy to learn
    - reliable
    - safe
    - fast
- System
    - easy to design
    - easy to implement
    - easy to maintain
    - reliable
    - error-free
    - efficient

## Simple OS

- 分層頂多 1 或 2 層
- 不安全、難加強

## Layered OS

- 分了很多層，低層依賴高層
- 但不能跳層呼叫
- 較簡單除錯跟維持
- 低效率、難定義層級

## Microkernel OS

- 程式碼越少越好
- 從內核移動到 user space
- 有了 module，subsystem，kernel 負責 module 的溝通
- message passing
- 可靠性非常高
- 效能更差

## Modular OS

- 現今最常用的
- 一樣有 Subsystem，全部都在 kernel space
- 溝通就可以不用 message passing
- 可以加入自己寫的 module

## Virtual Machine

- 可以在原 OS(1) 上，再建立一個 OS(2)
- Difficult to achieve due to **critical instruction**
- 難以實現**關鍵指令**
- 當 OS(2) 要執行 system call 時，會向 OS(1) 呼叫 interrupt，但 OS(1) 是在 user model 接收到的，所以會發生錯誤
- 解決辦法，就是 OS(1) 幫 OS(2) 攔截 system call，再幫他執行一次，所以實際上執行的 OS(1)
- 有些 CPU 會寫說有支援，就是說除了有 dual mode 以外，還有一個 vm mode
- 可以提供完整的保護系統資源，個別 OS 不會互相影響

- 對於系統兼容性高，可以在同一電腦上有多系統使用

- 對於 OS 研究跟開發是一個相當好的載具

- 雲端運算崛起

## Full Virtualization

- 個別 OS 都不知道其他人的存在
- vmware

## Para-virtualization

- 有一個 OS 知道其他所有 OS 的存在
- Xen

## Java Virtual Machine

- 就像是執行在一個 VM 裡面一樣
- 執行在一個虛擬環境下，所以不會去影響其他系統內的程序
- Just-In-Time 使得表現提升