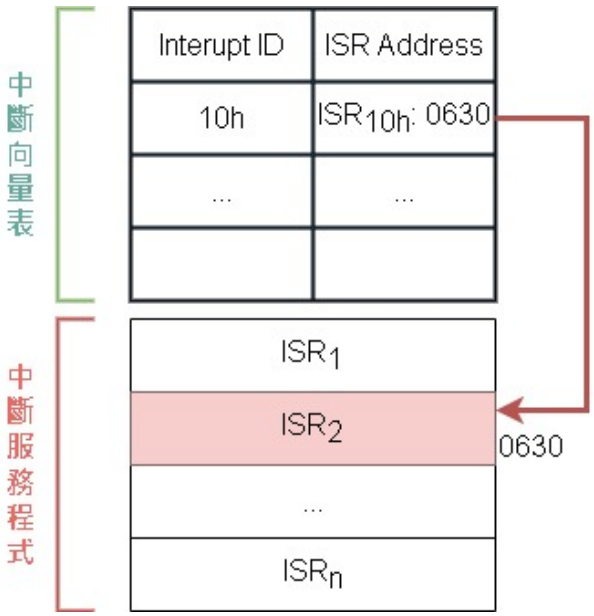
# 電腦系統架構



# 電腦系統的操作

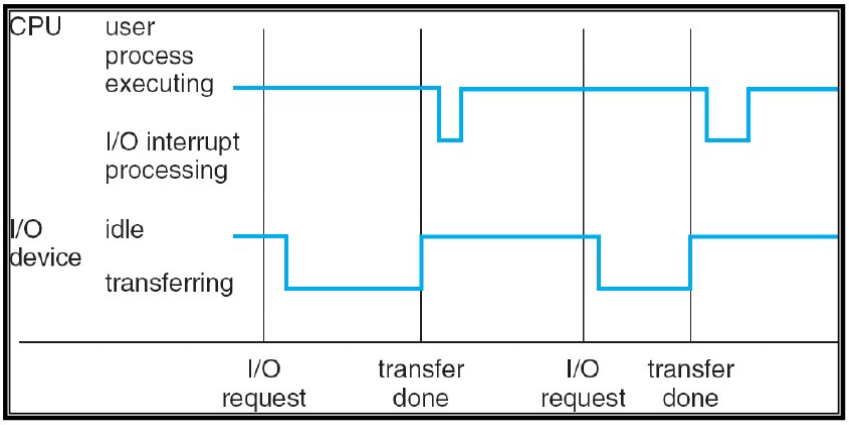
- **Device Controller** 負責一個特定型態的裝置
- I/O 設備和 CPU **同時執行**與**競爭 Memory**
- Device Controller 有自己的 **local buffer(暫存器)**
- 電腦系統中發生一個事件時，通常是由硬體或軟體產生**中斷**來通知
- 近代作業系統是**中斷驅動式(Interupt Driver)**

# 中斷




- OS Area(Memory)內會存在**中斷向量表**和**一組中斷服務程式**
  - 中斷向量表 Interupt Vector
  - 中斷服務程式 Interupt Service Routine(ISR)

## 中斷發生時的處理步驟




1. OS 要求 CPU 暫停目前 Process 的執行，同時保存當時的狀態到 Stack
2. 根據 Interupt ID 查詢 Interupt Vector 可以找到相對應的 ISR 起始位址
3. 跳到 ISR 的起始位址，系統執行 ISR
4. ISR 執行完畢，將控制權交回 OS
5. 恢復原先中斷前的 Process 執行 (實際由 CPU Scheduling 決定)

## 中斷的種類

### 分類法一

#### *External Interupt*

- **CPU 以外**的設備引發
  - e.g. Machine check, I/O Complete, Device Error

#### *Internal Interupt*

- **CPU 本身**引發
  - e.g. illegal instruction, overflow, 運算除以 0

#### *Software Interupt*

- User Process 執行過程中，若需要 OS 提供服務，**由 User Process 發出中斷**通知 OS 提供所指定的服務，也稱為 **Trap(陷阱)**
  - e.g.**system call**

### 分類法二

#### *Interupt*

- **硬體**引發的中斷
  - I/O 設備發出" I/O Complete"中斷

#### *Trap*

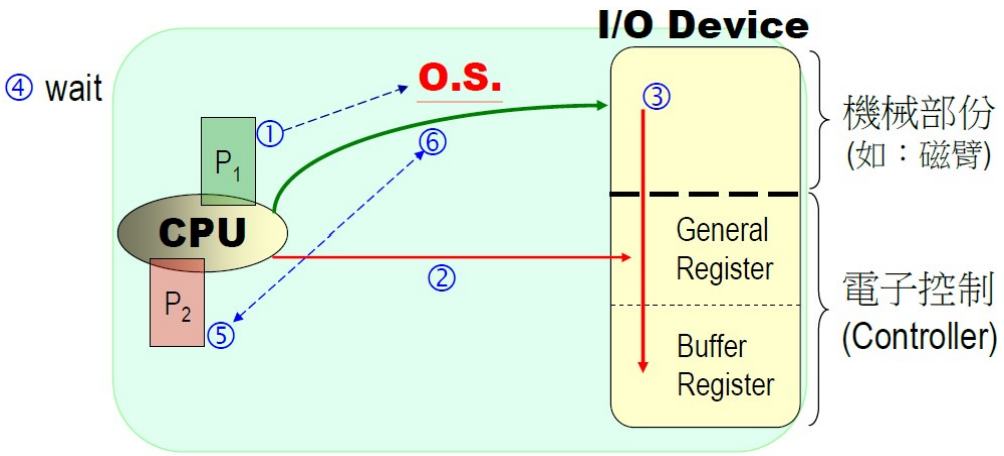
- **軟體**引發的中斷
  - User Process 需要 OS 提供服務時發出
  - 錯誤的數學運算發生 (e.g. 運算除以 0)

# CPU 與 I/O 的運作方式

## Polling I/O (詢問式 I/O)

- 也稱為 Busy Waiting I/O, Programmed I/O
- **General Register** 供 CPU 設定 I/O 命令，及接收命令
- **Buffer Register** 資料在傳送時，暫存資料

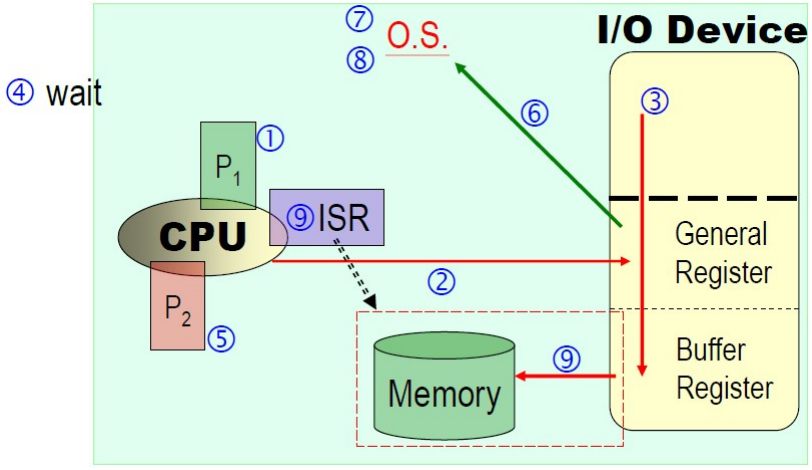
## Polling I/O 執行步驟




1. 工作 P1 發出 I/O Request
2. 由 OS 透過 CPU 來設定 I/O command to general register
3. I/O 運作
4. P1 wait for I/O complete
5. CPU 切給 P2 執行
6. **CPU 不斷詢問 I/O Device 運作是否完成**

- CPU 需**耗費大部分時間來監控 I/O 運作的執行**
- **對 Throughput(產能)無益**

## Interupt I/O (中斷式 I/O)

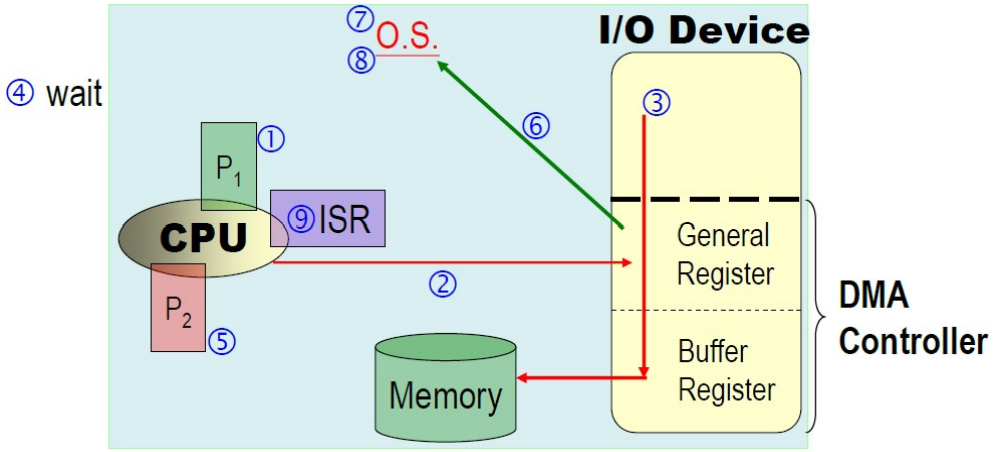



- **執行 Polling I/O 步驟 1 ~ 5**

1. I/O 完成後，Controller 會發出 I/O Complete Interupt 通知 OS
2. 暫停並保存目前工作 P2
3. OS 會根據 Interupt ID 查詢 Interupt Vector，取出對應的 ISR 位址
4. 執行相對應的 ISR
5. ISR 完成後 OS 通知 P1: I/O Request Complete，將 P1 從 Wait State 改為 Ready State

6. 恢復 P1 執行? (依據 CPU 排班選擇 Process)

## Direct Memory Access (DMA) 直接記憶體存取

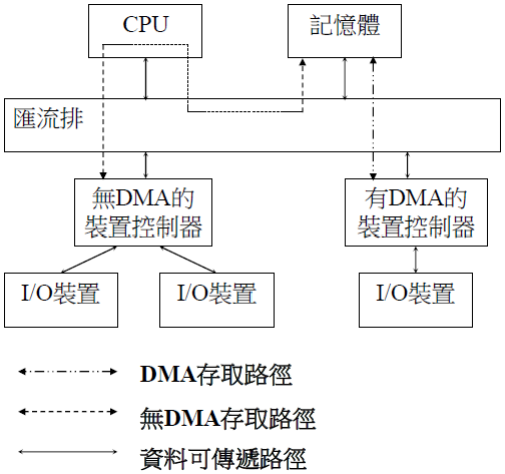



- 負責 **Memory 與 I/O 之間的資料傳輸**
- 傳輸過程不需要 CPU 參與或監督
- 用在高速 **Block-Transfer I/O Device**(e.g. Disk)
- **DMA 發出中斷的時間點與 Interupt I/O 不同**

### CPU 設定 DMA Controller 的運作有哪些

- I/O command (e.g. r/w)
- Physical Device Location (e.g. Track, Sector)
- Memory Location
- Counter (表示傳輸量的大小)
  - **傳輸量達到之後才會發出中斷**

### 有無 DMA Controller 的區別




### CPU 與 DMA 間對 Memory 的運作方式

- 以 **Interleaving 交替**的方式運行，**Cycle Stealing (週期偷取)**
- 機器指令的執行週期

1. **IF:** Instruction Fetch 指令提取 - 從**記憶體**抓取指令
2. **DE:** Decode 解碼 - 透過 CPU(CU, Control Unit) 解碼
3. **FO:** Fetch Operand 運算元提取 - 從**記憶體**抓取運算元
4. **EX:** Execution 執行 - CPU(ALU) 執行
5. **WM:** Write Result to Memory 將結果寫入**記憶體**

- 會使用記憶體的 Stage
  - 必須使用記憶體 **IF**
  - 非必須使用記憶體 **FO, WM**
- DMA Controller 可以去 **steal FO, WM** 的時間進行記憶體與 I/O 之間的傳輸
  - 不影響對記憶體的存取和指令的執行

### CPU 與 DMA 衝突處理

- CPU 與 DMA Controller 同時爭取 Memory，造成 Memory Conflict
- OS 會給 DMA Controller 較高的優先權
  - 優先服務**資源要求較少**的行程(由 OS 來判斷)，可以得到**較少的平均等待時間**
    - CPU 對 Memory 需求很高
    - DMA 對 Memory 需求較少，只有在有需要時才執行
- **分配資源通則**
  - I/O Bound Job 對 CPU 有較高的優先權
  - CPU Bound Job 對 I/O 設備有較高的優先權
  - 例外：**Real Time System**，要給 **CPU Bound Job** 較高的優先權

## 儲存體結構

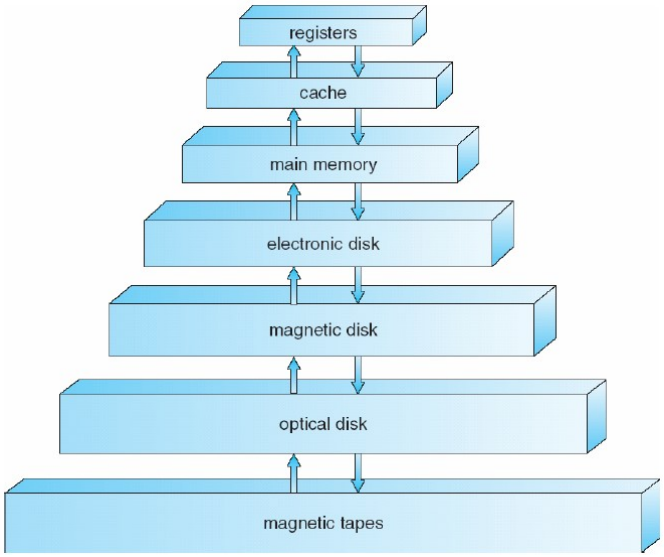
### 主記憶體

- CPU 唯一能夠存取的外接儲存區域
- 容量小
- 具**揮發性 (Volatile)**

### 輔助記憶體

- 為主記憶體的擴展，具**非揮發性 (Nonvolatile)**
- 容量大

# 儲存體裝置的階層 Storage Hierarchy




## 暫存器

- **Program Couter**(PC 程式計數器)
  - 暫存下一條指令所在之 Memory 位址
- **Instruction Register**(IR 指令暫存器)
  - 暫存由 Memory 中取出的指令，供 Control Unit 解碼
- **Memory Data Register**(MOR 記憶體資料暫存器)
  - 暫存欲取出或存入 Memory 的 data 或 intermediate results (中間結果)
- **Memory Address Register**(MAR 記憶體位址暫存器)
  - 暫存欲取出或存入 Memory 的 data 或 intermediate results (中間結果) 的位址
- **Process Status Word**(PSW 行程狀態字元)
  - 紀錄 ALU 執行指令後的狀態 (e.g. +/-, 0, overflow?, etc)
- **Base Register / Limit Register**(基底暫存器/限制暫存器)
  - Base Register 紀錄 Program 執行的起始位置 (程式在 Memory 的起始位址)
  - Limit Register 紀錄 Program 所需的記憶體大小

## 快取記憶體 (Cache Memory)

- **目的**改善 CPU 對主記憶體的存取速度
- **作法**
  - 將記憶體中常被存取的區域內容，儲存在 Cache Memory 中
  - CPU 會先到 Cache Memory 尋找指令或資料
  - 如果 Hit(命中)，則無需到 Memory 中存取，否則才會到 Memory 中存取
- **Cache Memory Hit Ratio 高 --> 效能佳**
  - Cache Memory 的設計目的是用來**節省資料搜尋的時間**
  - 速度跟**資料命中率**相關，因此容量加倍不代表速度加倍
- 硬體系統中，快取記憶體分為兩種

    - L1 Cache 內建在 CPU
    - L2 Cache 在 CPU 之外(PCB) ???
    - L3 Cache ???
    - L1 比 L2 速度快，CPU 會先從 L1 開始搜尋，找不到再搜尋 L2，最後才到 RAM 搜尋

### *Cache Memory 更新策略*

- **Write Through**
    - Cache 更新，立刻對主記憶體更新
    - **優點** Cache 和主記憶體內容一致
    - **缺點** 耗時，喪失使用 Cache Memory 的好處(尤其是 Write 指令很頻繁時)
- **Write Back**
    - Cache 內容要被置換出去(Swap Out)時，才更新內容到主記憶體的內容
    - **優點** 節省時間
    - **缺點** Cache 跟主記憶體內容可能不一致
- 在 Multiprocessor (多重處理器)的環境下，會有**資料一致性問題**，在 Distributed System(分散式系統)是常見的問題

### 主記憶體

| **RAM** | **ROM** |
| --- | --- |
| Random Access Memory | Read Only Memory |
| 資料可隨意存取 | 資料唯讀 |
| 揮發性 斷電資料消失 | 非揮發性 斷電資料仍存在 |
| 容量大 | 容量小 |

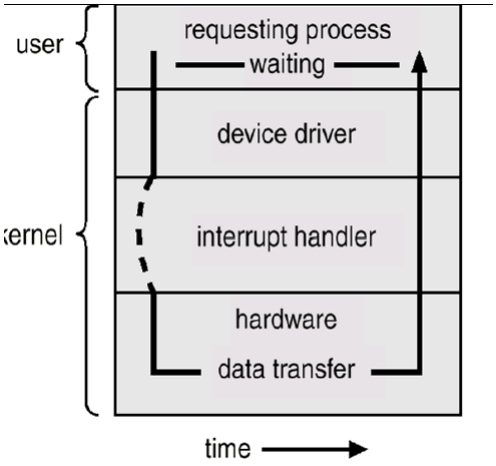
### 磁碟

- 由控制器決定資料傳輸
- 電腦將命令傳送到控制器的 General Register，由控制器操作硬體完成 I/O 動作
- 將資料從磁碟移到 Buffer Register，在從 Buffer Register 將資料傳送到電腦
- 磁碟存取時間由下列三個時間加總
    - **尋找時間**(最耗時)
    - **旋轉潛伏時間**
    - **傳輸時間**

## I/O 結構

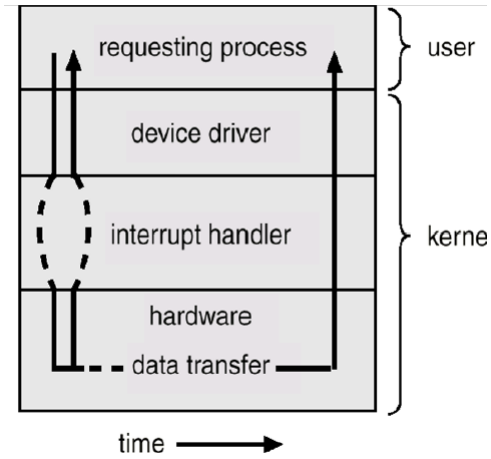
- 當 User Process 發出 I/O Request 後，系統的控制權多久會交還給 User Process?

## Synchronous I/O Structure (同步 I/O 架構)




- **I/O 完成運作後**才會將控制權交還 User Process
- Busy Waiting Loop 或 Wait 指令等待
- **優點** 一段時間內**只有一個 I/O 請求產生**，當 I/O Complete 中斷發生，OS 立刻就知道是哪個 Device 發出
- **缺點** 不支援 I/O 並行處理

## Asynchronous I/O Structure (非同步 I/O 架構)




- 立刻將控制權交還給 User Progress，不需等待 I/O 運作完成
- 可以有多個 I/O Request 同時發生
- OS 必須有一個**Device Status Table**紀錄各種 Device 的位址和 Device 的執行狀況

# 硬體保護 (Hardware Protection)

- 前題
  - **Dual-Mode Operation 雙模式運作**
  - **Privileged Instruction 特權指令**

# Dual-Mode 雙模式運作

## Monitor Mode 監督模式

- **只有 os 可以運行**，可執行 System Process，User Program 不允許運行
  - ISR
  - I/O request
  - 記憶體管理
  - CPU Scheduling
- Supervisor Mode, System Mode, Kernal Mode
- 只有這個 Mode **才有權利執行特權指令**

## User Mode 使用者模式

- User Program 允許被執行
- 不能執行特權指令，會引起 **illegal instruction error**，產生錯誤中斷，OS 強迫 Process 中止

## Dual-Mode 目的

- 保護系統不受 User Program 破壞
  - 特權指令
- 避免 User Program 不會互相干擾

## 硬體對 Dual-Mode 製作的支援

- 硬體需要**提供 Mode Bit(模式位元)**用來區分兩種模式
  - 0 代表 Monitor Mode
  - 1 代表 User Mode

# 特權指令

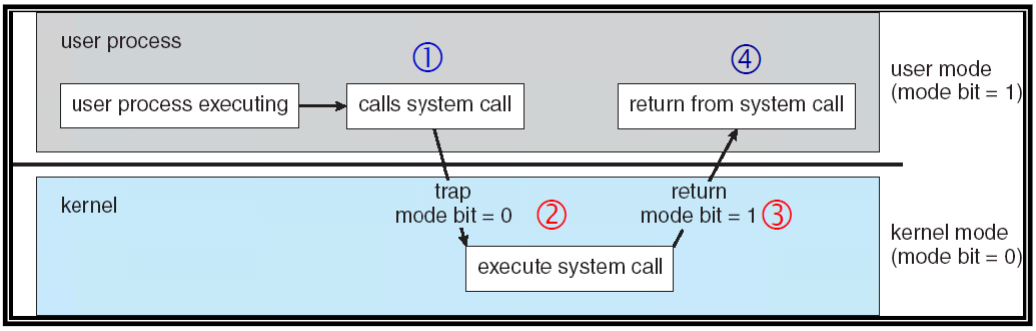
- **I/O 指令**
  - 避免 I/O 被 User Program 占用
- **記憶體管理**有關的暫存器之**修改指令**
- **Timer** 設定有關的指令
  - 與 Scheduling 相關
- **Enable/Disable Interupt** 指令
- **系統停止 Halt** 指令
- 使用者模式改變到監督模式的指令

# I/O 保護

- 防止 User Program **直接使用 I/O Device**
- 必須**提供 Dual-Mode 運作**
- 所有 **I/O 指令都設為特權指令**
- 必須保護 **Interupt Vector 和 ISR 所在的 Memory Area**

## 執行流程

1. 發出 I/O Request，即 System Call，會伴隨一些 Trap，轉換 Mode
   - **Trap 通知 OS 要求一個服務**
   - **System Call 通知 OS 要求什麼服務**
2. 執行相對應 I/O 服務
3. I/O 回傳結果給 OS
4. OS 在將結果回傳給 User Program




## 若 User Program 可更改 Interupt Vector

- Interupt Vecotr 可被改為指向 User Program
  - 導致 User Program 在 Monitor Mode 下執行

## 若 User Program 可更改 ISR

- ISR 內容可執行 User Program
  - 導致 User Program 在 Monitor Mode 下執行

# 記憶體保護 Memory Protection

- 保護執行中的程式或 OS 不會被其他程式干擾
- 保護 Monitor Area
  - 保護 Monitor(OS) 所在區域不被 User Program 修改
- 保護 User Area
  - 防止 User Program 企圖更改其他 User Program 的記憶體資料

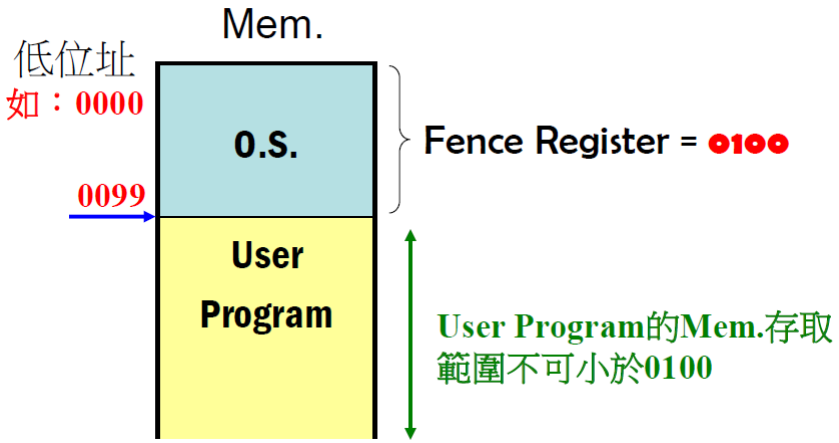
## Dynamic Binding

- 決定程式執行的起始位置
  - i.e. 程式要在記憶體的哪個地方開始執行

- 當程式執行的起始位置決定了，程式內的資料或變數之記憶體位置也就決定了
- Binding 的時期
  - **Compiling Time** 編輯時期
  - **Loading Time** 載入時期
  - **Execution Time** 執行時期
- Dynamic Binding: 在執行期間才決定程式執行的起始位址，表示在執行期間可以任意變更其起始位址

## Monitor Area 的保護

### *方法一Fence Register 界限暫存器*

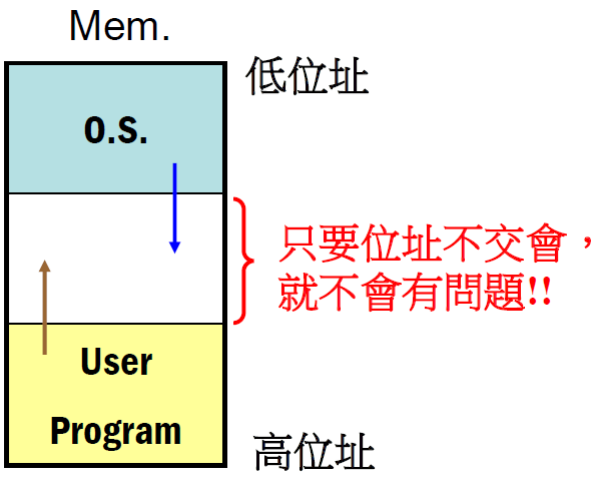



- 用來記錄 **os 的大小** (Limit)
- User Program 的記憶體存取範圍不能在 OS 的記憶體範圍
- 須將**修改或設定 Fence Register 的指令**設為特權指令
- 如果 OS 所在的區域大小動態改變，會造成 User Program 的執行起始位址改變，需要靠 Dynamic Binding **重新定址**
- **缺點** User Program 的效能變差

### *為什麼OS Area 大小會改變*

- OS 錯誤處理程序，但不會一次全部在入記憶體
  - 當有一個錯誤發生，在 OS 記憶體中查不到對應的程序，就會利用 Dynamic Loading 了方式從 Disk 中載入相對應的處理程序，所以 OS 的大小會改變

*方法二 Monitor Area 與 User Area 往反方向增長*




- Monitor Area 由高位往低位擺
- User Area 由低位往高位擺

### 各個 User Area 的保護

- 使用 Register
  - **Base Register**
  - **Limit Register**
- **Base Register** 基底暫存器
  - 紀錄 User Program 的起始位址
- **Limit Register** 限制暫存器
  - 記錄 User Program 的大小
- **須將修改或設定 Base / Limit Register 的指令設為特權指令**

## CPU 保護

- 防止 User Program **長期占用 CPU** 不釋放
- OS 會規定一個 User Program 使用 CPU 時間的合理最大值
  - Timer 隨著 Progres 的執行遞減
  - Timer 為 0 時會發出 **Time Out 的 Interupt** 通知 OS 強迫 User Program 放棄 CPU
- **須將 Timer 的設定指令設為特權指令**

### Timer 用途

- **CPU Protection**
- **Time Sharing System**- Round Robin Scheduling
- DMA Controller **傳輸量(Counter)大小的設定**
- 系統時間