

## 作業系統：Overview

### 目錄

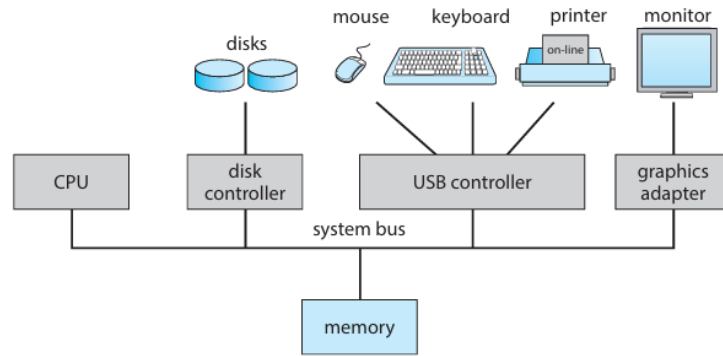
作業系統：Overview .....	1
1. 電腦系統組織 .....	1
2. 作業系統的角色 .....	2
3. 不同角色對 OS 的觀點 .....	2
4. 作業系統提供的服務(Operating-System Services) .....	3
5. 核心(kernel) .....	3
6. Polling（輪詢） .....	3
7. 中斷(Interrupts) .....	4
7.1 以 I/O system 來看 Interrupts(Chapter 12) .....	4
7.2 中斷向量表(Interrupt Vector Table) .....	5
8. 系統呼叫（System Calls） .....	6
9. Direct Memory Access(DMA)(由 Chapter12 補充的，可以在往後面放) .....	7
10. Multiprogramming & Multitasking .....	8
11. 電腦系統架構(Computer-System Architecture) .....	8
11.1 Symmetric Multiprocessing(SMP 對稱多處理) .....	8
11.2 Multicore Systems(多核心系統) .....	8
11.3 NUMA(非一致記憶體存取) .....	9
11.4 叢集系統(Clustered Systems) .....	10
12. 開機流程(Bootstrap Process) .....	10
13. 雙模式(Dual-Mode) .....	11
14. 計時器(Timer) .....	12
15. Linker & Loader .....	12
16. I/O Systems Overview .....	13
16.1 記憶體對應 I/O(Memory-Mapped I/O) .....	13

### 1. 電腦系統組織

現代電腦系統主要涵蓋一個或多個 CPU，多個裝置控制器(device controller)，並透過系統匯流排(bus)連接彼此與記憶體(memory)。每個 controller 責一類裝置(如：磁碟、音訊、圖形)，有些控制器可同時連接多個裝置(例如 USB hub)。

這些裝置控制器(device controller)內部會有緩衝區(local buffer)與特殊暫存器(registers)，負責與其裝置間的資料搬移。而作業系統為每個控制器提供驅動程式(device driver)，讓 OS 能以一致的方式管理不同裝置。

Figure 1：典型的 PC 架構

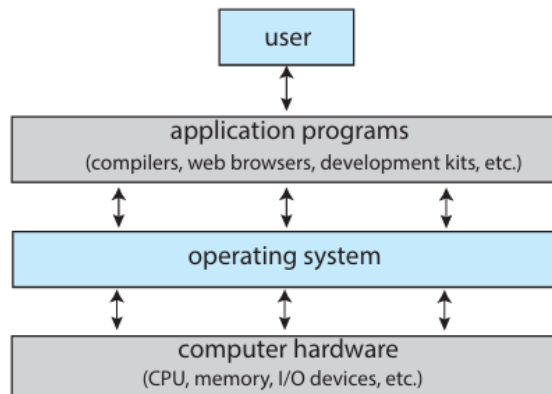


## 2. 作業系統的角色

計算機系統可以粗略分為四個部分：

- 硬體(hardware)：CPU、記憶體與 I/O 裝置，提供資源。
- 作業系統(operating system)：協調硬體與應用程式的使用。
- 應用程式(application programs)：如文字處理器、編譯器、網頁瀏覽器等。
- 使用者(user)：最終使用者。

Figure 2：計算機系統的組成



## 3. 不同角色對 OS 的觀點

對一般使用者(User View)而言，作業系統就是操作電腦的介面，例如透過螢幕、鍵盤與滑鼠來執行應用程式。大多數使用者不會關心系統資源如何被共享或分配。因此，作業系統的設計重點在於提升使用便利性(case of use)，接著才考量效能(performance)與安全性(security)，

對於系統觀點(System View)，作業系統是最貼近硬體的程式管理並分配各種硬體資源，其角色可分為兩大部分：

1. 資源分配者(Resource Allocator)：OS 管理並分配各種硬體資源，例如 CPU 時間、記憶體、儲存空間、I/O 裝置，目標是追求資源使用的效率(efficiency)與公平性(fairness)。
2. 控制程式(Control Program)：OS 控制使用者程式的執行，防止錯誤行為與不當資源使用。尤其是對 I/O 裝置的直接操作，需特別加以管理與保護。

## 4. 作業系統提供的服務(Operating-System Services)

OS 的主要目的之一，是提供一個方便且高效的環境讓程式得以執行。OS 對「使用者」與「程式」都提供一組共通的基本服務。這些服務可以分為「幫助使用者與應用程式的服務」與「幫助系統自身運作效率的服務」：

Table 1：幫助使用者與應用程式的服務

服務名稱	說明
使用者介面 (User Interface)	提供 GUI(圖形介面)、CLI(命令列介面)、或觸控式介面，讓使用者與系統互動。
程式執行 (Program Execution)	載入程式到記憶體並執行它。若程式執行完成或錯誤終止，OS 要能處理結束程序。
輸入/輸出操作 (I/O Operations)	管理所有 I/O 裝置。由於使用者無法直接操作硬體，因此 OS 必須提供中介方法來進行 I/O。
檔案系統操作 (File-System Manipulation)	提供建立、讀寫、搜尋、刪除檔案與資料夾的功能。也包含權限控管(誰可以存取哪個檔案)。
程式間通訊 (Communications)	同一台機器裡讓不同程序之間能夠溝通，可透過「共享記憶體」或「訊息傳遞(message passing)」來完成。
錯誤偵測 (Error Detection)	作業系統要能持續監控錯誤，包括硬體錯誤(如記憶體或網路)、I/O 錯誤(如印表機沒紙)、或程式錯誤(如存取非法記憶體)。有時必須終止程式，有時可回傳錯誤代碼讓程式自行處理。

Table 2：幫助系統自身運作效率的服務

服務名稱	說明
資源分配 (Resource Allocation)	當多個程序同時執行，系統要分配 CPU、記憶體、磁碟等資源。使用排程演算法(如 CPU scheduling)與資源管理方法。
使用記錄與帳務 (Accounting)	系統紀錄每個程式使用了多少資源，例如 CPU 時間、記憶體、磁碟空間等，用來做帳務分析或統計。
保護與安全 (Protection and Security)	確保不同程序間無法任意干擾彼此，也防止非法使用者入侵系統。包括帳號密碼登入、權限控制、防火牆等。

## 5. 核心(kernel)

核心(kernel)是作業系統中永遠在運行的程式，可以分為：

- 系統程式(system programs)：輔助作業系統，但不一定屬於核心。
- 應用程式(application programs)：用戶運行的程式，不參與作業系統運作。

## 6. Polling (輪詢)

Polling 是一種由 CPU 主動查詢裝置狀態的資料傳輸方式。Polling 是主機透過不斷詢問裝置是否準備好來進行資料傳輸的流程。步驟如下：

Table 3：Polling 簡易步驟

主機輪詢 busy bit (狀態暫存器)，直到裝置準備好。
寫入資料 到 Data-out。
設定 command-ready。
裝置控制器接收命令 → 執行 → 清除 busy

## 7. 中斷(Interrupts)

了解完 Polling，就可以來看看 Interrupt，中斷(Interrupt)是 CPU 與外部裝置之間通訊的重要機制，可讓系統即時回應硬體事件，而非不斷輪詢(polling)浪費效能。基本流程如下：

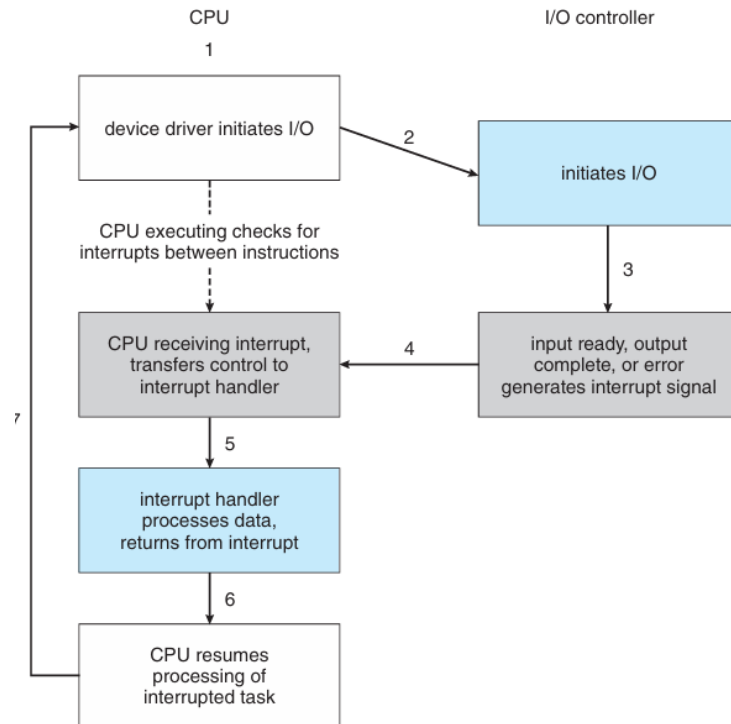
1. CPU 執行使用者程式
2. 裝置控制器完成任務後發出中斷訊號(interrupt signal)
3. CPU 接收到中斷後，暫停目前執行的程式，並根據中斷向量表(Interrupt Vector Table)跳到對應的中斷服務程式(ISR)
4. ISR(Interrupt Service Routine) 處理完事件後，還原先前 CPU 狀態，繼續執行原本的程式

### 7.1 以 I/O system 來看 Interrupts(Chapter 12)

那如果以 1.1 I/O system 來看 Interrupts(Chapter 12)：

1. CPU 啟動 I/O
2. 裝置完成後產生 中斷信號 (interrupt request)
3. CPU 儲存狀態 → 跳轉至中斷處理程序 (ISR)
4. 處理完後返回原本任務

Figure 3：Interrupt-driven I/O cycle



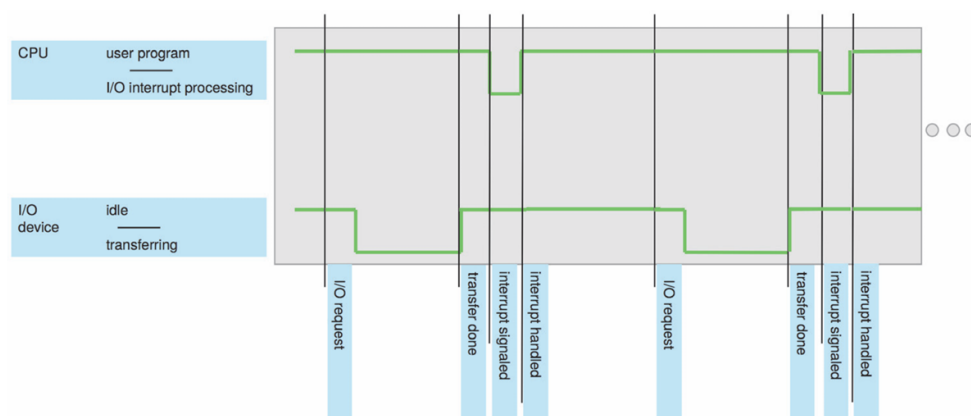
## 7.2 中斷向量表(Interrupt Vector Table)

就是存放 ISR 的地方（是一個指標陣列），讓 CPU 能根據中斷編號快速跳轉處理。中斷類型可以分為：

- Non-maskable：不可屏蔽中斷，無法被忽略，例如硬體錯誤
- Maskable：可屏蔽中斷，可由軟體設定是否暫時忽略

而當多個裝置共用同一條中斷線時，便可以使用中斷連鎖機制(Interrupt Chaining)，即一個 ISR 執行後，主動查詢其他可能發出中斷的裝置，依序處理。

Figure 4：中斷處理的基本流程圖(Interrupt Handling Flow)



8. 系統呼叫 (System Calls)

事件大多透過中斷(interrupt)來通知 OS。其中 Interrupt 可以分為：

- 硬體中斷：例如滑鼠移動、磁碟完成。
- 陷阱(Trap)/ Exception：例如：除以零錯誤、記憶體違規存取)

同時，又有另一種「軟體中斷」：System call(系統呼叫)，是用來請求 OS 幫忙執行特權任務(例如存檔)。

開發者寫程式通常使用的是 API (Application Programming Interface)，例如 read()。而這其實是「包裝好」的 system call，由作業系統提供的函式庫（如 Linux 的 libc）實作。優點為跨平台、簡單好寫，不用煩惱細節。

Figure 5：呼叫 read 時的 System Calls 邏輯

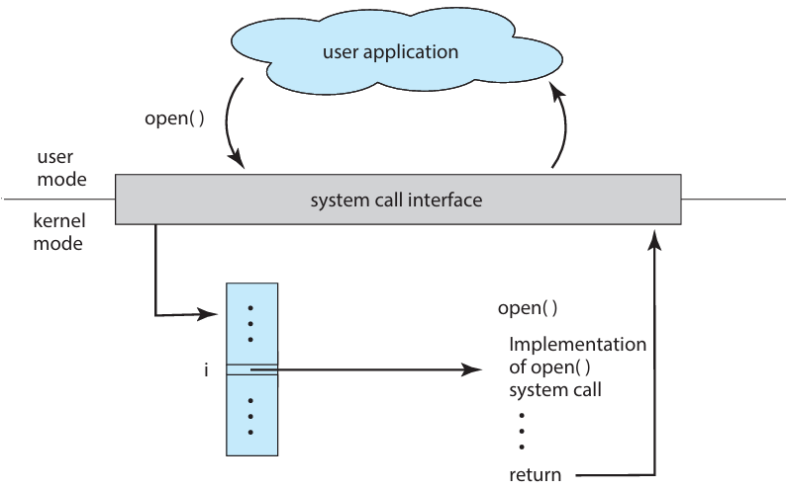


Table 4：System Calls 種類

分類	常見功能	範例 system call
Process Control	建立、終止程序、等待、記憶體分配等	fork(), exec(), exit()
File Management	建立、開啟、讀寫、關閉檔案	open(), read(), write()
Device Management	請求/釋放裝置、讀寫、移動資料	ioctl(), read(), write()
Information	時間、日期、系統資訊、屬性查詢與設定	gettimeofday(), getpid()
Communication	進程之間的訊息交換 (Message/Shared memory)	pipe(), shm_open()
Protection	設定/查詢權限、限制資源使用	chmod(), umask()

對於 System Call 的參數傳遞形式，如果是少量參數就直接用佔存器；如果參數較多，就使用 stack(推入資料)或記憶體區塊(把所有參數放進記憶體，然後只傳這個位置)。

備註：Linux 結合以上兩種：少的用暫存器，多的就用 block。

Table 5：系統呼叫(System Call)vs. 中斷(Interrupt)

特徵	系統呼叫(System Call)	中斷(Interrupt)
----	-------------------	---------------

誰觸發	程式主動請求	硬體或裝置主動發出
時機	程式需要作業系統幫忙時	例如鍵盤輸入、網路封包到達時
控制權轉移	使用者程式請求進入核心	當前執行單元被「打斷」進入核心
是否可預期	可預期	多半不可預期
範例	read(), write(), open()	鍵盤輸入中斷、計時器中斷

## 9. Direct Memory Access(DMA)(由 Chapter12 補充的，可以在往後面放)

在傳統的 I/O 模型中，資料傳輸是由 CPU 控制的：

1. CPU 從裝置讀取一筆資料
2. 再將資料寫入主記憶體
3. 每筆資料都要經過 CPU 操作（造成 overhead）

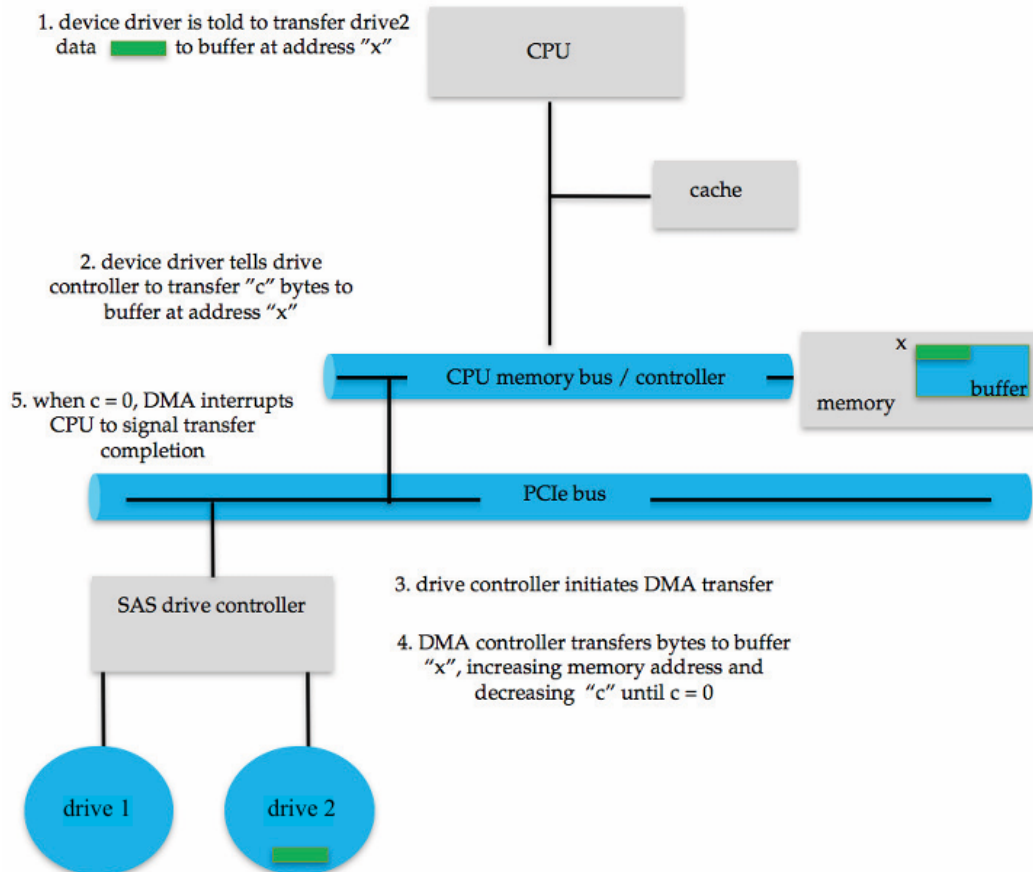
這樣這種方式在處理大量資料（如硬碟檔案、音訊串流）時效率低落。CPU 資源被浪費在搬運資料上，而無法專注處理其他運算任務。而 Direct Memory Access（DMA）是一種由硬體控制器（稱為 DMA 控制器）所主導的資料傳輸機制。它允許 I/O 裝置（如硬碟、音效卡、網卡）直接與主記憶體（RAM）進行資料傳輸，跳過中央處理器（CPU）的介入，以節省 CPU 資源並提高效能。主機只需要事先提供這三個資訊：

- 來源/目的的記憶體位址
- 資料長度（要傳多少）
- 指令區塊（Command Block）：用來描述任務的控制結構

DMA 步驟大致如下：

1. CPU 發出命令給驅動程式（device driver）（假設要從 Drive 2 傳送資料到記憶體中的 x 位址）
2. 驅動程式（device driver）會把指令更詳細的告訴磁碟控制器（drive controller）（要傳送 c bytes 的資料到記憶體位址 x）
3. DMA 控制器進行傳輸
  - I. 將磁碟資料逐步傳送到記憶體的位址 x，
  - II. 每傳一筆就更新記憶體位址（x++），
  - III. 並將剩餘資料量（c）遞減，直到 c = 0。
4. 資料傳完後，DMA 發中斷通知 CPU，這時 CPU 才再次介入做後續處理。

Figure 6 : Steps in a DMA transfer.



## 10. Multiprogramming & Multitasking

- Multiprogramming：多個程式在記憶體，CPU 一次執行一個，等待時切換
- Multitasking：Multiprogramming + 快速切換

## 11. 電腦系統架構(Computer-System Architecture)

現代電腦系統可能只用一個處理器，也可能有上百個處理器，設計和操作系統的支援方式會有所不同。早期電腦大多屬於單處理器系統(Single-Processor Systems)，一個 CPU 核心(core) 執行所有指令與處理所有任務。而現代，多處理器系統(Multiprocessor Systems)則成為現代標準配置。多處理器系統(Multiprocessor Systems)可以進一步細分：

### 11.1 Symmetric Multiprocessing(SMP 對稱多處理)

所有 CPU 共用主記憶體與系統匯流排。每個 CPU 都是平等的，可以處理作業系統和應用程式。優點是運行效率高，N 顆核心最多可同時跑 N 個程序。而缺點是當 CPU 太多時，彼此會爭搶匯流排，導致效能下降。

### 11.2 Multicore Systems(多核心系統)

一顆實體晶片上面有多個核心(cores)，每個核心有自己的暫存器與 L1 cache，並共享 L2 cache。核心之間溝通快、耗能低，是現今最常見架構(如手機、筆電)。



### 11.3 NUMA(非一致記憶體存取)

每個 CPU 有自己的本地記憶體，彼此透過系統互連線互通資料，共享記憶體空間。適合大量 CPU 的擴充 (scalability) (多用於伺服器)。若 CPU0 存取 CPU3 的記憶體，會比較慢(有延遲 latency)→ 需做「CPU 排程」與「記憶體分配」優化來避免延遲

Figure 7：對稱式多處理架構(Symmetric Multiprocessing, SMP)

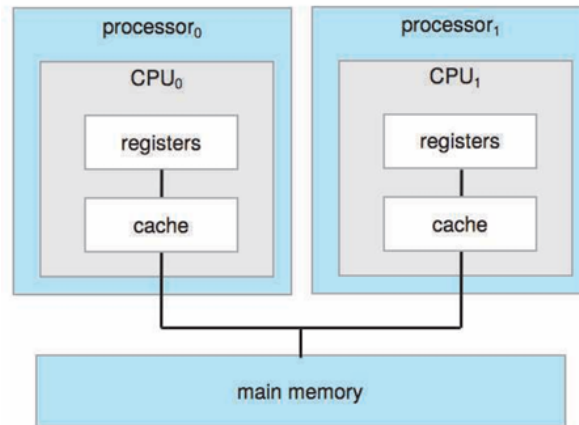


Figure 1.8 Symmetric multiprocessing architecture.

Figure 8：雙核心設計：兩個核心位於同一顆晶片上

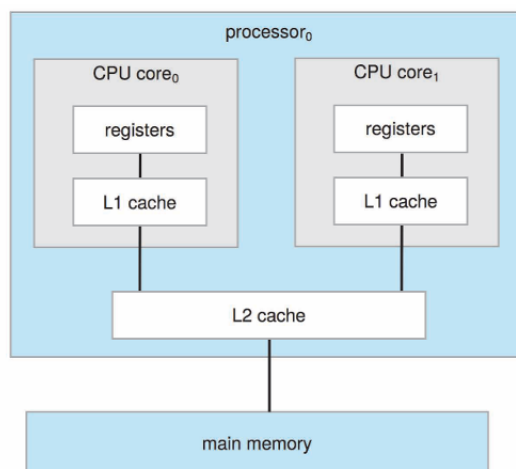
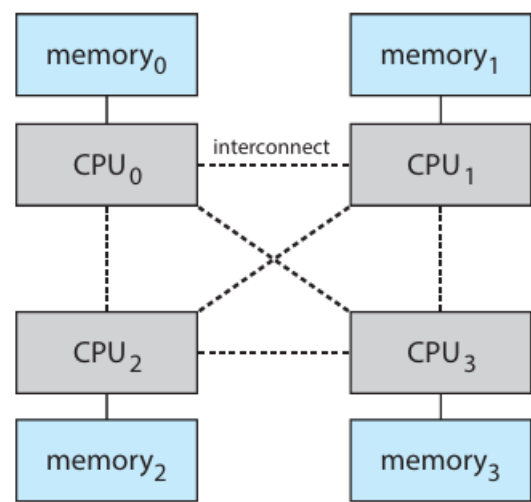


Figure 9：NUMA(非一致性記憶體存取)多處理架構



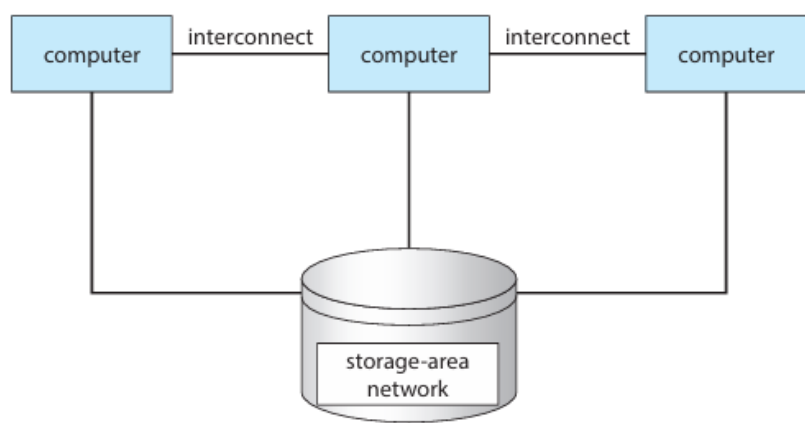
11.4 叢集系統(Clustered Systems)

多台電腦(節點)組成一個群組，透過網路互連(如 LAN 或 InfiniBand)。每台電腦是完整的系統，可有多核心。

Table 6：Clustered Systems 的功能

高可用性 (High Availability)	若有一台電腦掛掉，可以由另一台接管。又稱熱備援(hot standby)/ 對等備援 (symmetric)。
高效能 (High Performance Computing)	每個核心有自己的暫存器與 L1 cache，並共享 L2 cache。核心之間溝通快、耗能低，
共享儲存 (Shared Storage)	彼此之間採用 SAN(Storage Area Network)共用資料。系統則需使用「分散式鎖定管理 (Distributed Lock Manager, DLM)」來避免資料衝突。

Figure 10：叢集系統的一般架構

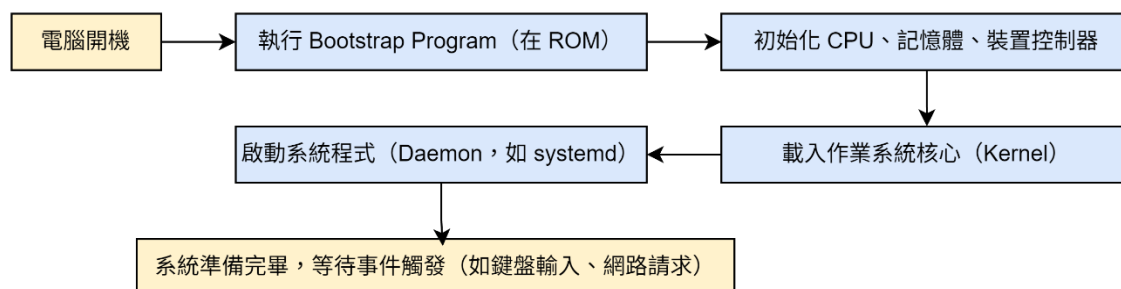


12. 開機流程(Bootstrap Process)

電腦開機時，會執行 bootstrap program(啟動程式)，bootstrap program 會儲存在 firmware(如 ROM)，負責初始化 CPU、記憶體與裝置控制器，然後載入作業系統核心(kernel)。Kernel 開始執行後，會載入系統程式(稱為 daemon，

常駐程式)，例如 Linux 中第一個系統程式是 systemd。系統完成開機後，就等待「事件」(例如：鍵盤輸入、網路請求)。

Figure 11：開機流程



### 13. 雙模式(Dual-Mode)

目的是避免惡意或錯誤程式傷害系統，Dual-Mode 可以分為：

- User mode(使用者模式)：執行使用者程式，受限、無法執行危險指令
- Kernel mode(核心模式)：執行作業系統程式，有最高權限，可直接操控硬體

Figure 12：user mode 與 Kernel mode 的交互形式

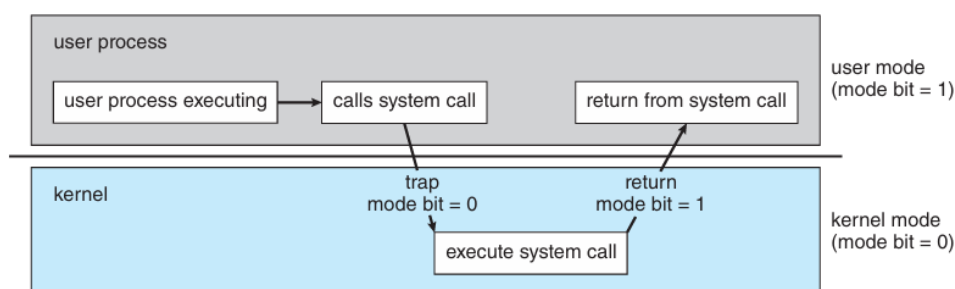
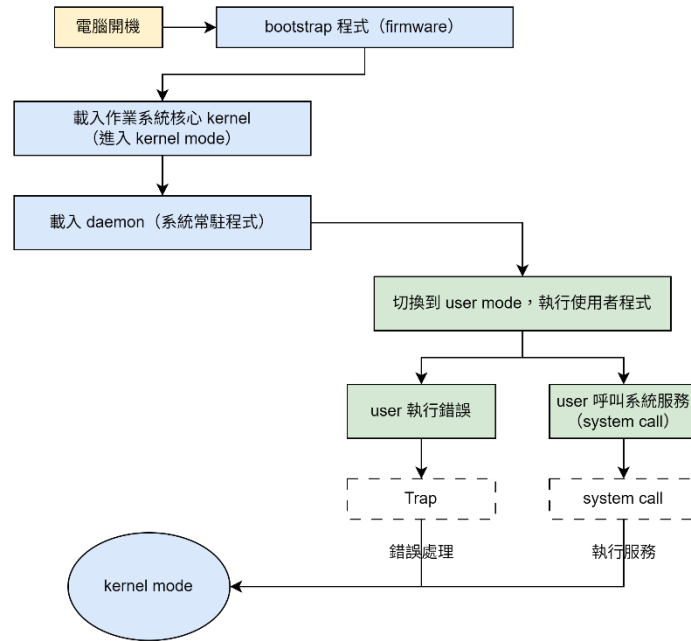


Figure 13：電腦開機到進入使用者模式



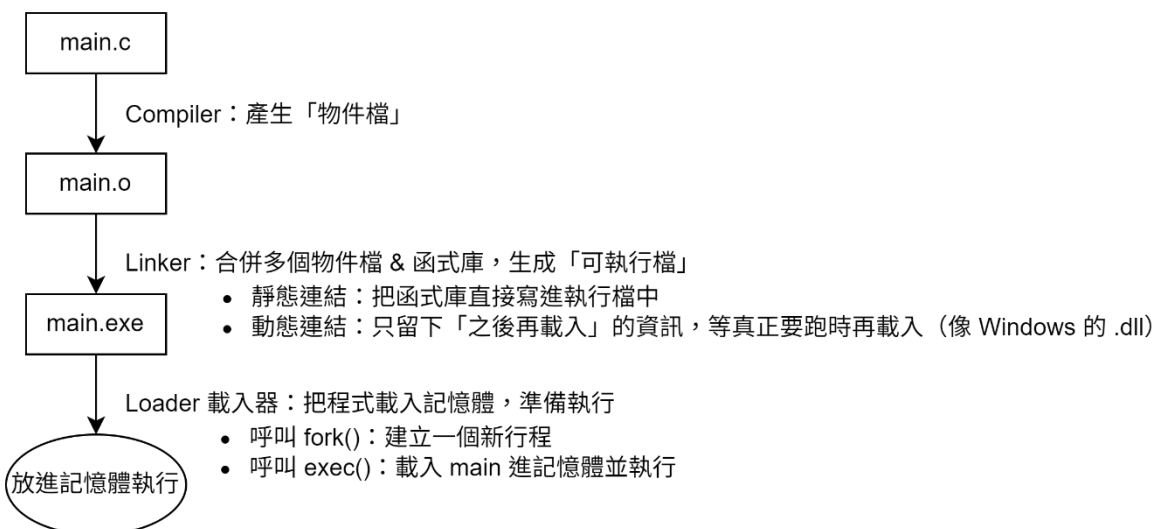
## 14. 計時器(Timer)

定時中斷，強迫程式交出 CPU 控制權，目的是避免程式無限迴圈或長時間佔用 CPU

## 15. Linker & Loader

當你寫好一支程式，例如 main.c，你不能直接執行它。它必須經過以下流程：

Figure 14：從原始碼到可執行檔的載入與執行流程



## 16. I/O Systems Overview

作業系統為什麼要管理 I/O 裝置？I/O 裝置是計算機系統中不可或缺的一環，像是鍵盤、滑鼠（人機互動），硬碟、SSD（資料儲存），印表機、顯示器（輸出）。每個 I/O 裝置的速度與操作方式都不同，這對作業系統是個挑戰。而當前 IO 技術發展逐漸往以下兩種趨勢：（彼此衝突）

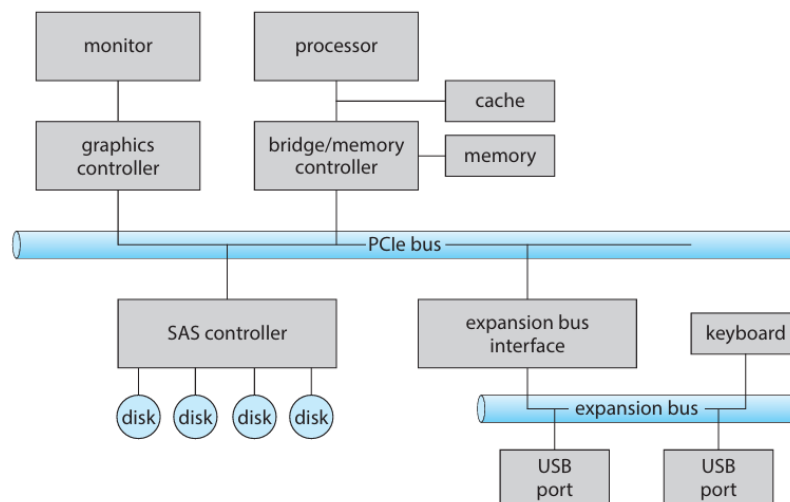
- 標準化（Standardization）：軟體與硬體的 I/O 介面越來越標準化，讓新一代裝置可以無縫整合進現有系統。
- 多樣化（Diversification）：I/O 裝置的種類越來越多（如 VR 控制器、掃描槍、機器手臂等），這讓整合變得更困難。

裝置透過 Port/BUS 傳送「電壓訊號」與系統互動，定義嚴格協定與傳輸時序。

Table 7：硬體介面與連線方式

名稱	說明
Port（埠口）	裝置連接點，如 USB 埠、Serial Port（串列埠）
Bus（匯流排）	多裝置共用的傳輸線（如 PCIe、USB、SATA）
Daisy Chain	鍊狀連接裝置（常見於 SCSI）
PHY (Physical Layer)	OSI 七層中的實體層，常用於資料中心術語

Figure 15：A typical PC bus structure



### 16.1 記憶體對應 I/O(Memory-Mapped I/O)

那在我們之後會談到，我在記憶操作值這一塊，OS 並不是直接操作，而是會區分為「邏輯地址」與「實體地址」。Memory-Mapped I/O 是一種 I/O 設計方法，將設備控制器的暫存器映射（map）到 CPU 可存取的實體記憶體空間中。這同時也表示：

- I/O 裝置的暫存器（例如：status、command、data）會佔用記憶體中的某些位址。
- CPU 可使用一般的資料存取指令（例如 mov、ld、st）直接對這些裝置暫存器讀寫，而不需要特殊的 I/O 指令（如下圖所示）。

Figure 16 : Device I/O port locations on PCs (partial)

I/O address range (hexadecimal)	device
000-00F	DMA controller
020-021	interrupt controller
040-043	timer
200-20F	game controller
2F8-2FF	serial port (secondary)
320-32F	hard-disk controller
378-37F	parallel port
3D0-3DF	graphics controller
3F0-3F7	diskette-drive controller
3F8-3FF	serial port (primary)