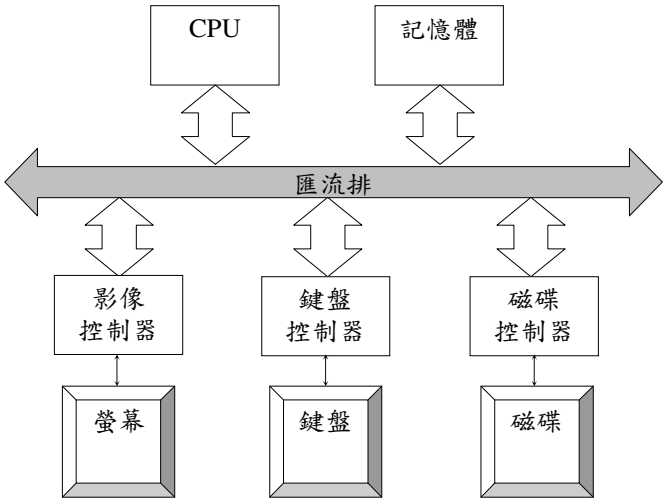
硬體結構

資科系
林偉川

硬體結構

- 越進步的硬體提供作業系統越強大的運算能力，強大的作業系統需更進步的硬體架構
- 了解作業系統也必須清楚底層硬體運作的方式，作業系統有很多實作的技巧都與硬體有密切的關係
- 一般用途的個人電腦，大多由CPU、記憶體以及輸出入裝置組合而成，之間以匯流排互相連接
- CPU
- 儲存裝置
- I/O 結構
- 儲存階層
- 硬體保護

電腦與週邊



3

CPU

- CPU就是中央處理器
  - 是電腦系統的心臟
  - 專門處理各種運算
  - 負責與週邊設備溝通
- CPU執行是靠程式計數器指向記憶體的程式區段，然後取出所指的指令來執行，每執行完一個指令後，程式計數器會自動指向下一個指令， CPU一次只能處理一個指令。

4

CPU

- 有效利用CPU是提高系統效能的關鍵之一
  - 軟體—CPU 排程 (縮短CPU閒置時間)
  - 硬體—階層式架構 (拉近硬體間速度的差異來提高效能)

5

暫存器

- CPU 與暫存器
  - CPU 要對任何資料作運算前，必須先將資料載入到暫存器中。
  - 暫存器功用與記憶體類似，是最接近 CPU 的記憶體。
  - CPU 存取暫存器的速度相當快，所以把資料存到暫存器之後再作運算，可以加快資料處理的速度。

6

暫存器種類

- 特殊用途暫存器與一般用途暫存器
  - 特殊用途暫存器：用來控制硬體所提供的特殊功能或是有特殊用途的暫存器。
    - 如 Intel 486 相容的架構下有 4 個特殊用途的暫存器 — CR0、CR1、CR2、CR3，主要用來管理浮點運算器(FPU)、分頁(Paging)及快取(Caching)的功能。
  - 一般用途暫存器：一般用途的暫存器可以用來存放資料或是記憶體的位址。
    - 如 Intel 486 相容架構下的 EAX、EBX、ECX、EDX。
    - Intel P4以後的暫存器?

7

范紐曼機

- 范紐曼機是美國普林斯頓大學的范紐曼博士所提出來的電腦系統架構。
- 他將電腦大致分為 5 個單元
  - 算術邏輯單元(+-*/及比較)、控制單元(CPU)
  - 記憶單元(儲存目前執行工作的程式碼及資料)
  - 輸入單元
  - 輸出單元(將CPU處理後的資料輸出)

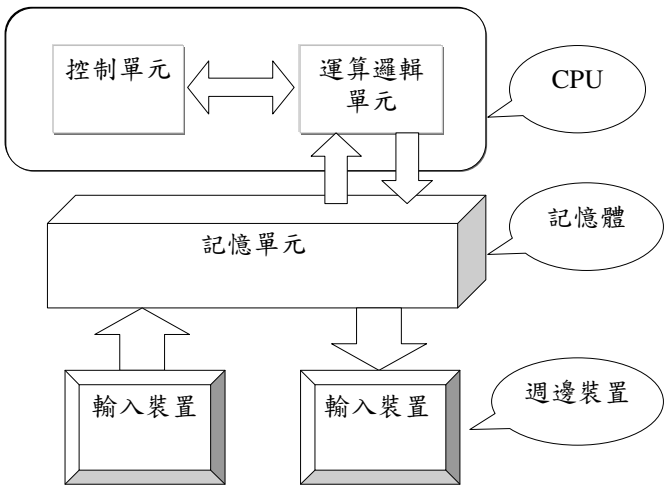
8

范紐曼機

- 當CPU開始執行時，控制單元會先將記憶單元裡的指令取出放在暫存器中，由控制單元判斷此指令是否應從輸入單元讀資料至記憶單元，或是從記憶單元內讀出資料交給算術邏輯單元運算，再存回記憶單元中，最後輸出到週邊裝置

9

范紐曼機



10

儲存裝置

- 主記憶體通常會有下列的缺點
  - 主記憶體通常不大，無法將所有要執行的程式同時載入。
  - 主記憶體大多是屬於揮發性記憶體。
- 因此電腦系統提供輔助記憶體來彌補主記憶體的不足
- 輔助記憶體是用來長期儲存大量的資料(光碟、軟碟、磁帶)
- 不同的裝置其存取速度、容量、體積大小、價格及物理性質也大不相同

11

記憶體

- 記憶體大致上分為 ROM 與 RAM 兩種。
- 資料燒錄在ROM後就無法修改，好處是資料不會因電源關閉後消失
- RAM通常用來當作主記憶體，因為RAM裡面的資料都可以快速地隨需要而修改
- ROM 與 RAM 會因製造技術及用途而有不同

12

記憶體

- Masked ROM是於生產時已由廠商將資料燒死在裡面，所以沒辦法修改裡面的資料，如此可降低成本，但其被讀取速度比RAM還慢
- PROM與Masked ROM不同的地方是其出廠後可用IC燒錄器寫入資料一次，之後就無法再寫入，其價格比Masked ROM還貴
- EPROM在寫入資料後，可以用紫外線照射十到二十分鐘將資料消除，然後再度寫入資料，這種ROM在程式除錯時很有用

13

記憶體

- FLASH是可以將資料重複寫入的記憶體，它在消除或寫入資料不用IC燒錄器幫助，而是電流，在寫入資料時是以一整個區塊(256-4K位元組之間的大小)為單位寫入，沒辦法一次只改掉1個位元組，且其寫入的速度不是很快，所以不適合寫入頻繁的工作，適用於更新頻率不是很高的用途上，例如：BIOS、PDA上的應用軟體之儲存，可透過網路或串列埠來更新軟體

14

記憶體

- EEPROM與EPROM不同的地方在於EPROM要用紫外線消除資料，而EEPROM只要在它特定的接腳上提供電壓(一般為3.3或5V)即可消除資料。其存取方式與FLASH非常相似，不同的是EEPROM一次只能讀取或寫入1個位元，所以其讀寫速度相當慢。通常EEPROM只用來儲存少量的資料，例如網路位址、使用者名稱、資料數量

15

記憶體

- DRAM的每一個位元是由1個電容及6個電晶體所組成的，因電容的容量不大，內部電荷在經過一段時間後會慢慢流失，所以每經過一段時間就必須要充電來維持電位，以確保資料的正確性
- SRAM所儲存的資料不會隨時間而消失，所以不用花額外的時間來充電，其讀寫速度也比DRAM快，但是價格較貴
- SDRAM、RAMBUS、DDR SDRAM、DDR2 SDRAM？

16

記憶體

- 下面介紹幾種不同類型的記憶體。

記憶體名稱	讀取速度	寫入速度	可寫入次數	揮發性記憶體
masked ROM	快	不能	0	否
PROM	快	不能	1	否
EPROM	快	不能	許多	否
Flash	快	慢	10,000	否
EEPROM	慢	慢	1,000,000	否
RAM	非常快	非常快	無限次	是
附註：此處的寫入速度是指在電腦上寫入的時間，以燒錄機寫入皆標為不能				

17

快取

- CPU存取內建暫存器只要花一個時脈的時間
- 快取是提昇系統效能的重要機制
  - 存取記憶體必須透過匯流排傳送資料到暫存器內，所以需要花費許多時間，所以CPU就必須停下來等資料，對系統而言是相當大的浪費。
  - 快取是加在 CPU 與主記憶體間的快速記憶體，當 CPU 在存取主記憶體的資料時，會複製一份相同的資料到快取之中，等 CPU 下次讀取相同一段記憶體位址的資料時，就能夠直接從快取中讀出。

18

快取

- 快取是提昇系統效能的重要機制
  - 如果所有資料可從快取裡找到，則效率提昇不少
  - 如果CPU無法從快取裡找到資料，存取一次所花的時間會是資料從記憶體讀出時間加上系統存取快取時間，系統的效能也因存取時間的增加而下降(Hit rate)
  - 快取的速度與成本皆高於主記憶體，通常CPU的快取只有幾百K位元組，因此快取管理對系統效能的影響相當大
  - 小心選擇快取的大小及好的管理策略，可讓80%-99%的資料於快取中找到

19

快取

- 快取是提昇系統效能的重要機制
  - Intel的快取分為兩層，第一層為最小的L1 快取，通常又分成兩部份，一部份是指令，一部份是資料，優點為速度最快，延遲最小，應該設定成跟CPU跑同時脈。在這一層的命中率大約在90%上下(視大小)，也就是剩下的10%將要在下一層中去找.

20

快取

- 第二層為L2快取，可以把它放置位置大致分成幾種：第一種為在主機板上的--Socket7架構、第二種為建立在CPU兩側的高速SRAM--K7.P6都有、第三種為由於製程的先進，在CPU上內建高速且,低延遲的快取，命中率視大小而定約在80%上下

- Intel Duo core, Intel Core 2 Duo V.S. Intel Core 2 Quad之L1 & L2 cache？
- XP 32bit OS只能認3GB以下記憶體
- (90Tku、96ncu)

21

晶片組(chipset)

- 北橋(north bridge)：作為CPU與記憶體與顯示卡(AGP或PCI Express控制器)與南橋溝通的橋樑(855，915，945)
- 南橋(south bridge)：整合週邊的介面。例如：IDE,USB,1394,網路,序列式傳輸扈,滑鼠,鍵盤,PCI (ICH?)
- 整合型晶片：把南北橋整合為一個封裝

22

磁碟(硬碟)

- 磁碟是電腦系統中常見的輔助記憶體，由下列機構所組成
  - 磁盤由直徑1.8、2.5、3.5、5.25吋左右的CD，兩面皆塗有均勻的磁性物質，藉由改變這些磁性物質排列的方式來儲存資料
    - 磁軌(track)
    - 磁區(sector)
    - 磁柱(cylinder)
  - 讀寫頭➔讀取資料
  - 磁碟臂➔讀寫頭繫在此
  - 摯動器➔轉動磁碟臂

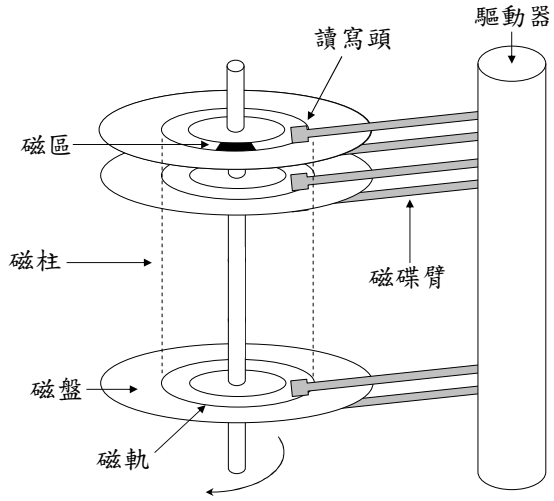
23

磁碟

- 讀寫頭利用磁盤高速旋轉所產生的氣墊來避免直接與磁盤表面撞擊，但是還是有可能撞到磁盤產生壞軌，而且愈來愈嚴重
- 影響磁碟讀取速度的因素
  - 傳送速率➔每秒有多少資料從磁碟傳到電腦
  - 定位時間，或稱為隨機存取時間(硬碟轉速)
    - 搜尋時間➔磁碟臂移到正確磁柱上的時間
    - 旋轉延遲➔正確磁區轉到讀寫頭位置的時間

24

磁碟構造圖



25

磁帶

- 磁帶是早期就有的輔助記憶體
  - 以循序的方式來存取資料，較花時間 。
  - 比隨機存取的磁碟慢上好幾千倍，不適合拿來當作輔助記憶體使用
  - 可以儲存大量資料。
  - 主要的用途是備份大型系統的資料。
- DAT(**Digital Audio Tape**) ？

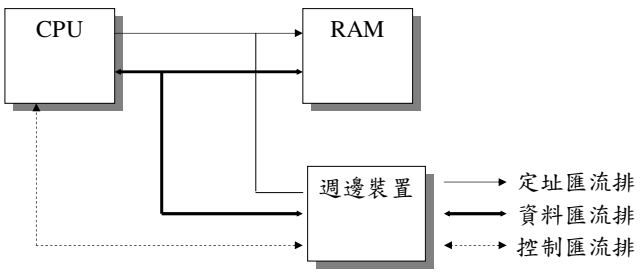
I/O結構

- CPU 和週邊設備有兩種溝通方式
  - I/O 對映(Intel 組合語言中的IN與OUT指令)
    - CPU 透過特殊的指令控制週邊設備。
  - 記憶體對映 I/O
    - CPU 與週邊設備間建立起相同的記憶體位址空間，彼此就以這段位址傳遞指令與資料。

27

匯流排

- 匯流排可以分成 3 個主要的部分
  - 定址匯流排 (address bus)➔指定資料傳輸位址
  - 資料匯流排 (data bus)➔雙向傳輸資料的管道
  - 控制匯流排 (control bus)➔傳送控制訊號通知周邊裝置該收送資料，還是其他的準備動作



28

匯流排

- 當CPU與記憶體的速度愈來愈快，匯流排就成為整個資料流量的瓶頸，各式各樣的新匯流排規格也應運而生，其傳輸的速度及方式也個有不同。
- 幾種常見的匯流排規格
  - ISA ➔ 老舊的IBM PC/AT的匯流排速度工作頻率為8.33MHZ，每次可傳送2位元組的資料，所以其速度最快為16.67MB/SEC
  - PCI➔工作頻率為66MHZ ，每次可傳送8位元組的資料，所以其速度最快為528MB/SEC

29

匯流排

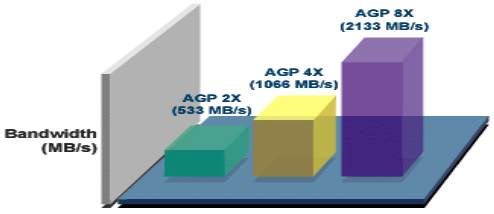
- 幾種常見的匯流排規格
  - IDE➔用來連接磁碟或是CD-ROM的介面，這種介面的控制晶片都是做在週邊設備上，所以介面卡較簡單，價格也較低
  - USB➔連接比較慢的外部I/O裝置，是屬於中控式的匯流排，有一個主要裝置會以輪詢(Polling：CPU不斷去檢查裝置是否需要服務的方式)的方式查看，所有裝置共用一個驅動程式，讓系統再加入一個新裝置時，不用安裝驅動程式，也不用重新開機就能使用該裝置，USB提供三種速度：低速 (1.5 Mbps)、全速 (12 Mbps、USB 1.1) 和高速 (480 Mbps、USB 2.0)

30

匯流排

- 幾種常見的匯流排規格
  - SCSI➔高效率的匯流排，專門接高速的磁碟等裝置，其資料傳送速率可以達160MB/SEC
    SCSI-II、Ultra Wide SCSI
  - AGP 8X？
  - IEEE 1394？
  - PCI-Express？

Mode	Approximate Clock Rate (MHz)	Transfer Rate (MB/s)
AGP 1X	66 MHz	266 MB/s
AGP 2X	133 MHz	533 MB/s
AGP 4X	266 MHz	1,066 MB/s
AGP 8X	533 MHz	2,133 MB/s



31

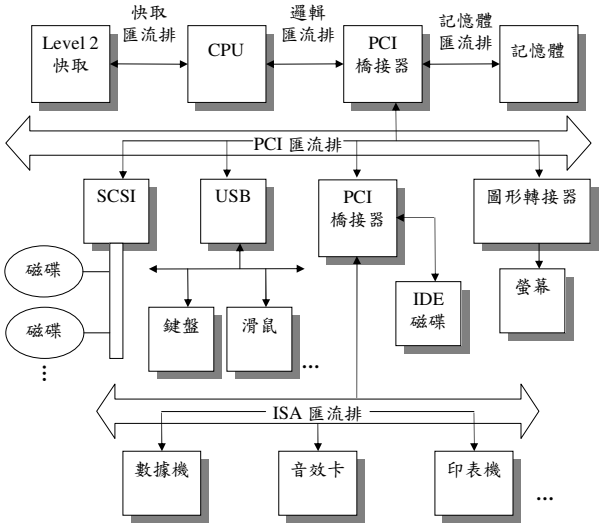
匯流排

- 幾種常見的匯流排規格
  - PCI Express➔PCI Express (亦稱為 PCIe)，是由 Intel 公司所開發，是取代 PCI 以提供更高頻寬的最新 I/O 介面。PCI Express 最明顯的改善就是其點對點的拓樸，能夠讓共享交換器根據優先順序，分配共享資源 給接附的 PCI Express 裝置。
  - PCI Express x4 / x8 / x12 是用於伺服器。

PCI Express Type	Bandwidth Single Direction	Bandwidth Dual Directions
PCI Express x1	250 MBps	500 MBps
PCI Express x2	500 MBps	1000 MBps
PCI Express x4	1000 MBps	2000 MBps
PCI Express x8	2000 MBps	4000 MBps
PCI Express x12	3000 MBps	6000 MBps
PCI Express x16	4000 MBps	8000 MBps

32

Intel Pentium 系統結構圖



33

控制器

- 一個週邊設備包括了兩個部分
  - 控制器
  - 裝置本身
- 控制器會有一些暫存器可以用來控制裝置，暫存器是用來回報裝置狀態、負責控制裝置的動作、或是存放資料
  - 驅動程式必須要有能力去存取這些暫存器
  - I/O 對映與記憶體對映 I/O 兩種方式就是將資料或指令寫入這些暫存器內來驅動裝置

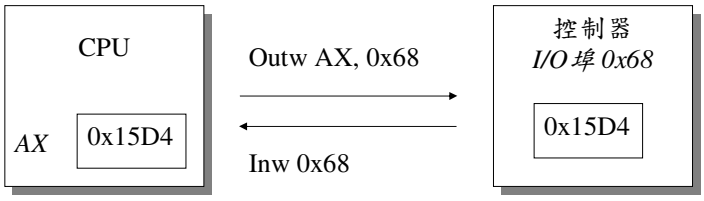
34

控制器是 CPU 與週邊設備溝通的橋樑

35

I/O 對映

- 每一個控制器上的暫存器都被給定一個特殊的 I/O 埠。
- Intel 的 IN 跟 OUT 指令可以用來分別讀出或寫入暫存器的值。



36

記憶體對映 I/O

- 記憶體對映 I/O 是將週邊設備的暫存器映對到記憶體位址空間。
- CPU 在存取這些暫存器時，就像是在存取記憶體裡面的值一樣。

37

記憶體對映 I/O

- CPU記憶體位址0X0-0XEFFF是分配給主記憶體使用，而0XF000之後的位址次給周邊裝置使用，改變0XF000內之值，就是改變控制器裡的暫存器，使用此種方式的裝置有串列埠(COM)和並列埠(LPT)
- 透過記憶體對映 I/O 的串列埠送出幾位元組長的字串時，CPU會將1位元組的資料寫到資料暫存器內，然後設定控制暫存器內某個特定位元，以通知周邊裝置去讀取資料

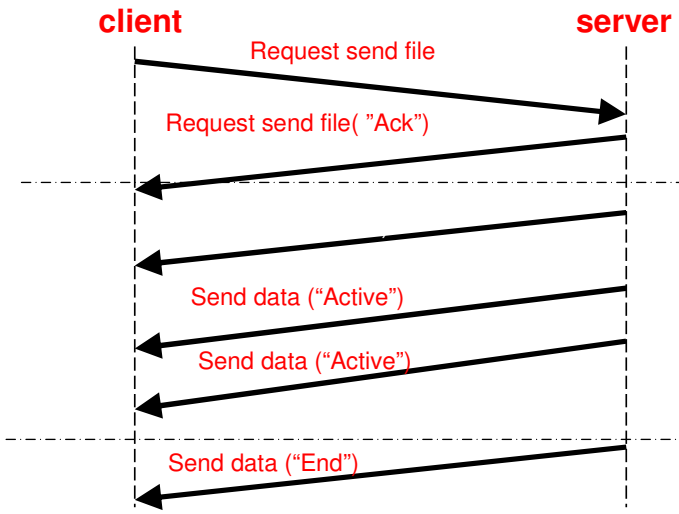
38

記憶體對映 I/O

- 週邊裝置讀取資料後，會將控制暫存器的特定位元清除，CPU知道後就可以開始傳送下一個位元組的資料到資料暫存器
- 假使CPU是以輪詢的方式去檢查控制暫存器，來得知資料是否準備好了稱為程式化I/O
- 假使CPU不是以輪詢的方式去檢查控制暫存器，而是當週邊裝置取得資料後，發出中斷通知CPU可以繼續傳送資料，這種方法稱為中斷驅動I/O

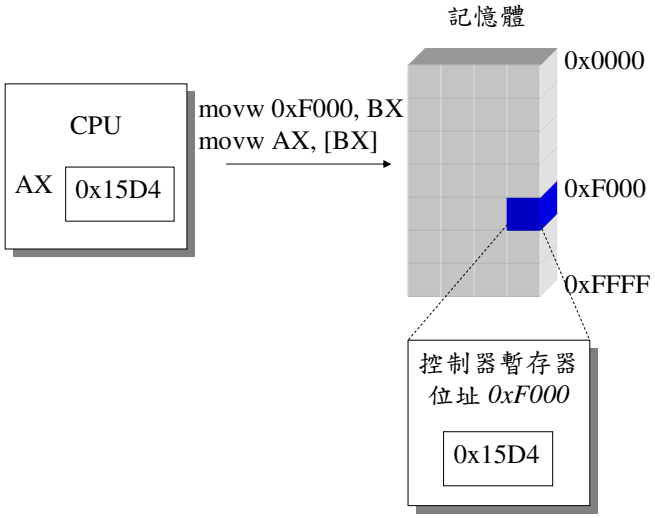
39

Client/Server Service Handshake



40

記憶體對映 I/O



41

I/O 中斷

- 一個 I/O 動作包含了下面的步驟
  - CPU 將資料載入到裝置控制器的暫存器中。
  - 裝置控制器依照暫存器裡面的值，讓裝置動作。
  - 裝置控制器完成工作後，發出一個中斷訊號通知 CPU 工作已經完成了。
- PnP V.S. UPnP？

42

I/O 中斷

- 有兩種等待中斷訊號的方式(91Tku)
  - CPU會等到I/O處理完才繼續工作稱為同步 I/O
  - 當讀取要求送出後，控制權馬上交回系統去執行其他的工作稱為非同步 I/O，此法可以讓周邊裝置處理I/O的同時，CPU還可以執行系統的其他工作
    - 發出讀取要求的行程必須等到I/O處理完才繼續完成工作(wait指令或是無窮迴圈來等待中斷完成)
    - 無窮迴圈也可能是不斷去詢問所有裝置是否完成工作，如果完成時，裝置會改變暫存器某個位元的值，供CPU檢查

43

I/O 中斷

- 只要資源不互相衝突的裝置，就應該能夠同時處理，這樣比較能得到比較好的效能
- 若使用同步 I/O，作業系統一次只能處理一個I/O要求，因此非同步 I/O使用讓CPU有機會去處裡其他的工作，會是對整體效能比較有效率，但是同步 I/O還是讓I/O更穩定且有效率
- 為了讓非同步 I/O可行，作業系統必須要記錄所有週邊裝置的狀態，此資料結構稱為裝置狀態表。

44

I/O 中斷

- 狀態表每個欄位記錄著裝置的類型、位址及狀態(忙碌或閒置)等(透過SNMP、MRTG)
- 對一個忙碌的裝置提出要求，作業系統會以等待佇列的形式將I/O要求記錄下來，等到裝置有空時才將佇列裡的工作一一完成
- 如果佇列空間滿了，有的裝置會通知作業系統停止再送要求，而有的裝置則很簡單地把新要求忽略掉

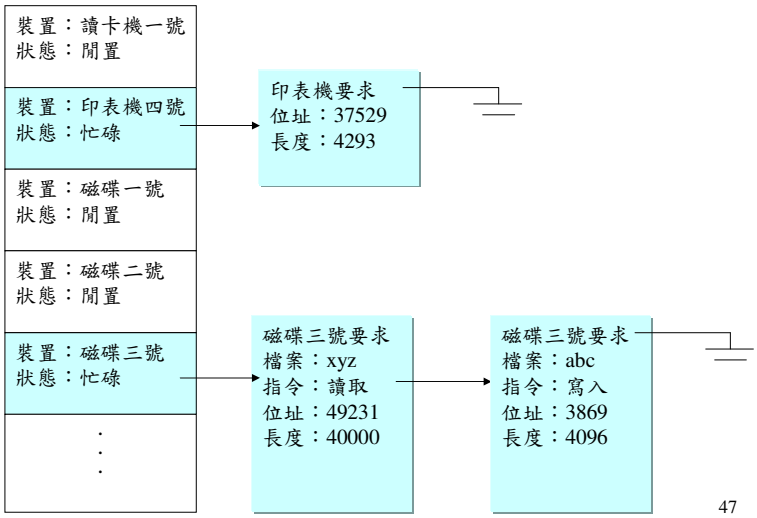
45

I/O 中斷

- 通常中斷發生，代表某裝置需要服務，作業系統會去檢查該中斷是哪個裝置發出，然後去檢查裝置狀態表的狀態欄位，得知裝置目前狀態後，依照中斷要求去修改欄位
- 大多數裝置會在完成I/O要求時發生中斷，如果此裝置還有其他I/O要求，作業系統就會繼續處理
- I/O要求處理完畢後，控制權會回到最初提出要求的行程
- 94ncu

46

裝置狀態表



47

直接記憶體存取 (DMA)

- 控制器都有提供DMA的功能，可以不經由 CPU 就能存取到記憶體
- 沒有 DMA 的控制器若想要存取記憶體中的資料，就必須依賴 CPU 的幫忙，每次存取大小只有1-4位元組。
- 終端機每833微秒就接收或傳送1byte，就對CPU發出中斷要求，CPU 需2微秒對中斷做出反應，一個中斷處理每個字元需要3微秒，則剩餘的828微秒，CPU 可處理其他事情(833-828)
- 對一些高速裝置，例如磁碟或通訊網路，傳送資料的速度與記憶體相近，所以會產生高頻率的中斷(4微秒中斷一次)

48

直接記憶體存取 (DMA)

- DMA 為解決高速裝置傳送的問題，DMA 控制器的兩個特性(94Tku、93Tku、93ncu)
  - 可以不經由 CPU 就能存取到記憶體。
  - 一次可以處理一整個區塊的資料。
- 當磁碟要傳送1MB到記憶體時，相關資料會先存在磁碟控制器裡的緩衝區，然後DMA會設定寫入記憶體的位址及寫入大小

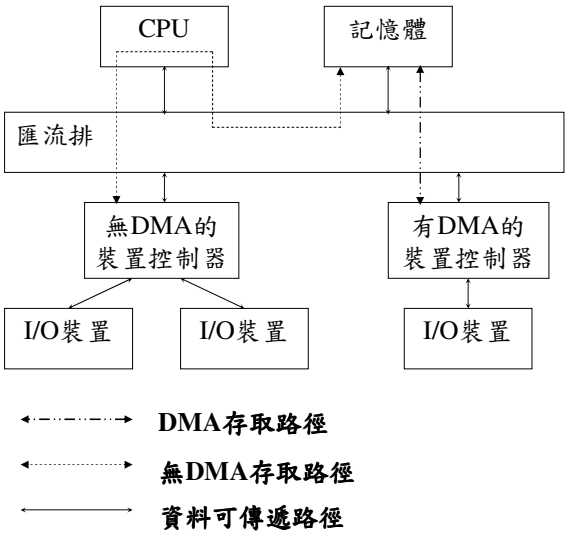
49

直接記憶體存取 (DMA)

- 等資料傳完或緩衝區滿了，DMA會向CPU發出要求使用匯流排的訊號，因為同一時間只能有一個裝置使用同一條匯流排，等到CPU送回同意的訊號時，DMA就依記憶體的位址，直接將資料整塊地寫入記憶體中
- 若緩衝區大小為4096位元組，DMA需4微秒來處理每個位元組，DMA至多只會每4096*4=19348微秒才發出一個中斷要求

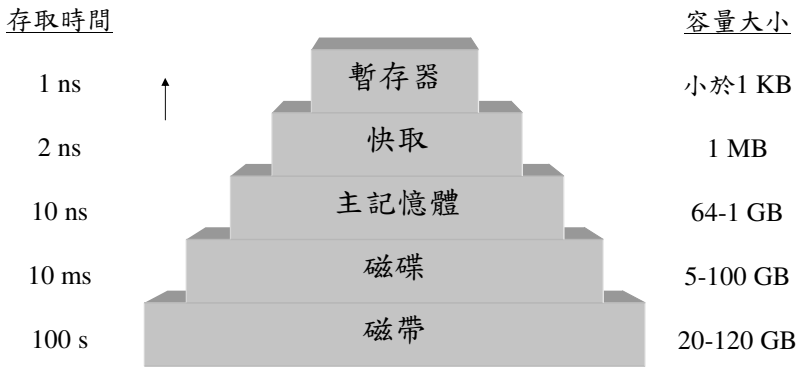
50

DMA存取路徑



51

儲存階層



52

儲存階層

- 在儲存階層中，在愈高層的裝置其價格愈高，但是速度也愈快。
- 從儲存階層往下看，儲存裝置的速度變得愈來愈慢，價格也往下降。
- 以價格來考量，上層裝置比例愈少愈好
- 以效能來考量
  - 希望所需要用的資料在上層就可以找到
  - 因此出現了快取機制及許多快取演算法
  - 上層裝置可當做下層的快取機制

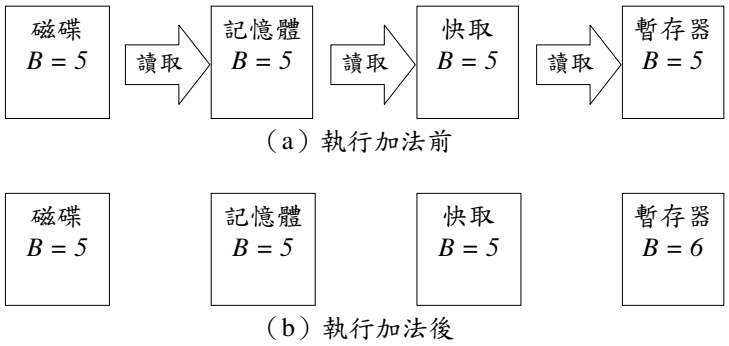
53

連貫性

- 以效能為考量，修改上層裝置的資料，不會每次都寫回下層的裝置。
- 檔案A裏的整數B被加1，因檔案A放在磁碟裏，所以在執行加法指令前，CPU會發出一個I/O要求，將整數B所在的磁碟區塊複製一份到主記憶體，硬體會將記憶體中的整數B複製到快取中，最後複製到暫存器後，由CPU加1才完成(6)，此時快取、記憶體及磁碟裡的整數B都是舊值(5)，必須等到寫回後整數B的值才會被更新(要等待一段時間)，所以若在此時斷電，資料可能未更新就流失。

54

連貫性



（a）執行加法前

（b）執行加法後

55

一致性

- 一致性是指系統中的工作在存取記憶體的時候，存取的動作不會互相干擾，而影響到資料的正確性。

56

一致性

- 一致性的問題在下面的情況會變得較為複雜
  - 多工環境下，CPU在許多工作間切換，可能造成很多工作同時修改同一筆資料，可能資料出錯的情形稱為競爭情況
  - 多 CPU 的環境，各自有自己的快取與暫存器
    - 不同CPU之快取資料的一致性
  - 分散式系統的環境 ，把一個相同檔案放在分散於各處的電腦中，各電腦只以網路做溝通，會發生相同的資料更新的問題
    資料庫之refresh動作及Linux之sync指令？

57

硬體保護

- 單使用者系統的時代
  - 程式設計師全權控制整個系統
- 多工作業系統的時代
  - 作業系統全權控制，提供許多的系統函式取代程式設計師的工作

58

作業系統結構

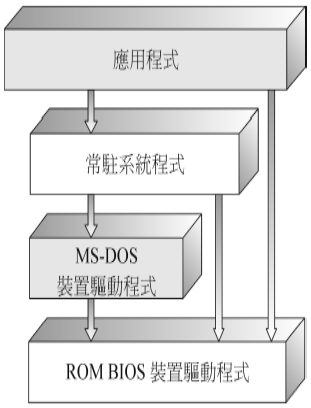


圖 2.7 MS-DOS 層次結構

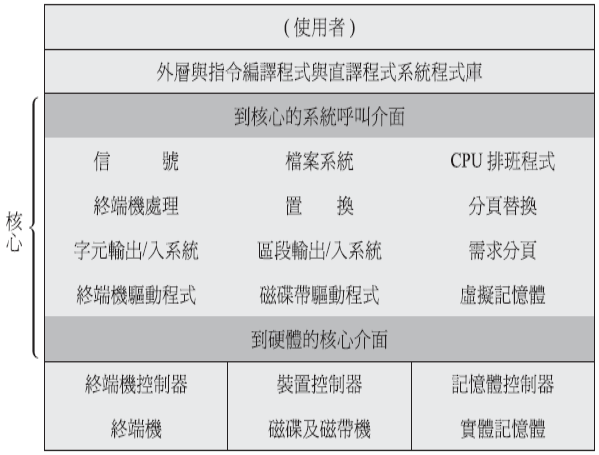


圖 2.8 UNIX 系統結構

59

分層方法

- 分層方式最主要的好處是結構簡單和除錯。每一個層次只能使用較低層的功能與服務。這種方式更容易做系統的除錯與驗證。第一層改正時，不必考慮系統的其餘部份，它只甩了基礎的硬體去完成它的功能。一旦第一層改正後，在第二層的工作可以假設它的功能正確。在特定層次除錯時，如果發現錯誤，可以知道錯誤必定在那一層，因為在它底下的層都已改正。當系統分層時，系統的設計與製作都可以簡化。

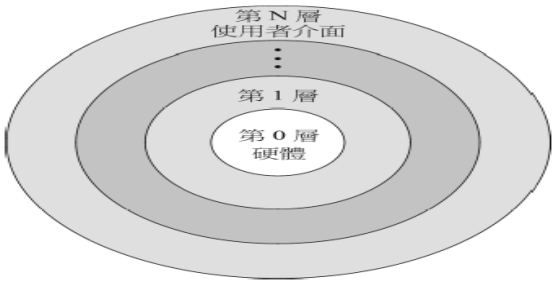


圖 2.9 分層作業系統

60

核心

- 在 1980年代中期Carnegie Mellon大學的研究人員發展出一套叫做Mach的作業系統，Mach使用微核心 (microkernel)的技術將核心模組化。這種方法藉由移去核心所有非必要的元件將作業系統結構化，並且改以系統和使用者層次的程式來製作。
- 模組

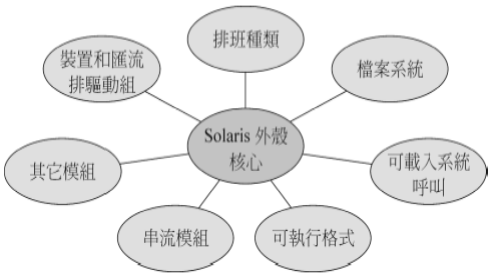


圖 2.10 Solaris 可載入模組

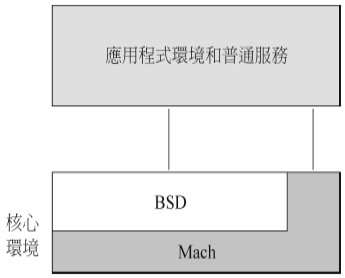


圖 2.11 Mac OS X 結構

虛擬機器

- 虛擬機器基本觀念是將單一電腦硬體 (CPU、記憶體、磁碟機、網路介面卡等等)想像成幾個不同的執行環境，因此產生了每一個獨立執行環境在自己個人電腦執行的幻覺。

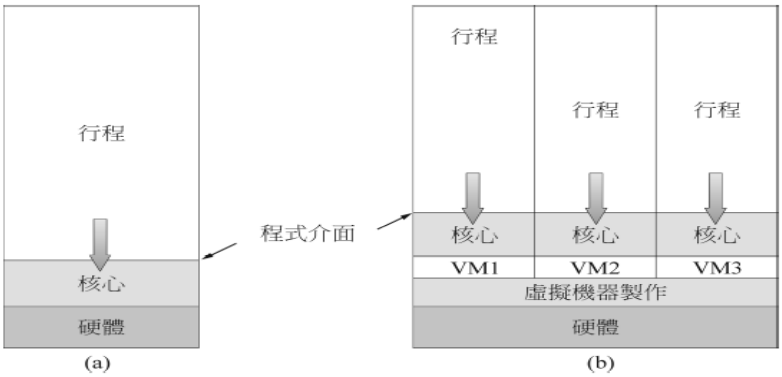


圖 2.12 系統模式 (a) 非虛擬機器；(b) 虛擬機器

62

虛擬機器

- 製作起來相當困難。困難是在於提供與實際機器完全一樣的複製。要記住的是實際機器有兩種模式：使用者模式和核心模式。
  - 虛擬機器軟體可以在核心模式中執行。虛擬機器自己只可以在使用者模式中執行。就像實際機器有兩個模式，所以虛擬機器也必須是如此。

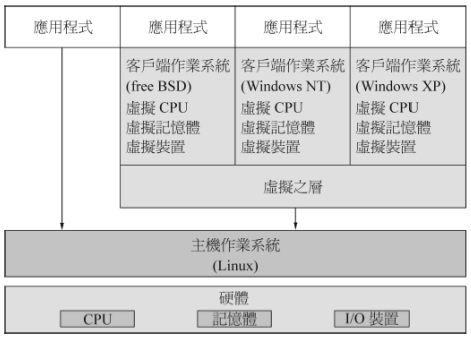


圖 2.13 VMware 架構

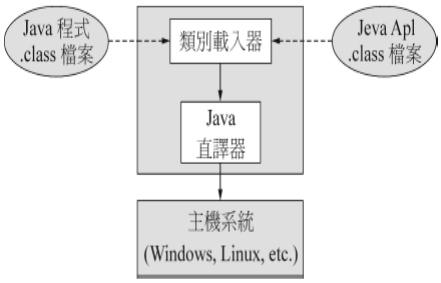


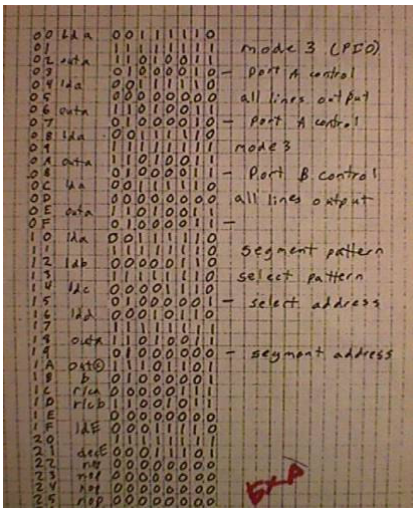
圖 2.14 Java 虛擬機器

作業系統建立之考慮

- 使用那一種CPU？設立那些選擇項 (擴充指令集、浮點小數運算等等) ？如果是多CPU，每個CPU都必須介紹。
- 有多少記憶體可以使用？有些系統會藉由一個接著一個參考記憶體位置直到一個“違法位址”的錯誤產生來自己決定這個值。這個處理程序定義最後合法的位址以及可用記憶體的數量。(判斷現有硬體的記憶體大小)
- 什麼裝置可以使用？系統必須知道如何對每一個裝置定址 (它的裝置號碼)、裝置中斷編號、它的類型與模式以及任何特別的特性。
- 想要什麼類型的系統或使用什麼參數值？這些選擇或值可能包括多少個這種大小的緩衝區可以使用、想要的CPU排班演算法則形式、可以支援行程的最大數量等等。

系統載入

- 藉由載入核心來開啟一部電腦的步驟就是載入 (booting)系統。
- 在大部份電腦系統之中，有一小段程式碼，叫做靴帶式程式(bootstrap program)或叫做靴帶式載入器 (bootstrap loader)位於核心、載入記憶體，並開始執行核心程式。(96nttu)
- 有些電腦系統，把這個步驟分成兩個階段，先有一個非常簡單的靴帶式載入器，再從磁碟載入一個更複雜的載入程式，然後再由後者載入核心。



bootstrap program

硬體保護

- 為了提高系統的使用率，作業系統將系統上的資源同時分享給許多程式使用
  - 因多元程式可以同時讓許多程式載入記憶體，要對記憶體作保護，以免每個程式可以修改其他程式的程式碼區段或是資料區段
  - 許多程式的錯誤只能靠硬體來偵測，由作業系統來繼續處理這些錯誤，當程式執行發生錯誤(不存在的指令、讀取不屬於這個使用者的記憶體位址、除以0)，硬體觸發例外中斷，由例外中斷的處理函式來執行，一般發生錯誤，記憶體的內容會寫到一個檔案(Linux's core vs 事件檢視器)

雙模式運作

- 兩種執行模式供作業系統使用
  - 使用者模式 (一般應用程式執行的模式user mode)
  - 系統模式 (作業系統執行的模式system mode)
  - 硬體在特殊用途暫存器中提供一個模式位元，來表示現在執行的模式為何
- 作業系統是靠中斷(Interrupt)來切換成系統模式，當中斷發生時，硬體由使用者模式切換成系統模式，處理完中斷將控制權還給使用者時，再切換為使用者模式

67

雙模式運作

- 需要雙模式運作的兩個原因
  - 保護共享資源，避免許多程式同時寫入一個裝置
  - 避免硬體指令造成傷害，系統中有些硬體指令可能造成系統傷害稱為特權指令，這指令只能在系統模式下才能被執行，除非透過系統呼叫，否則使用者程式是無法執行特權指令，若硬要執行則會被硬體偵測到，觸發一個例外中斷處理
  - 系統呼叫是一種軟體中斷，用來處理特權指令，當然也是在兩種模式中切換，此中斷又稱為陷阱中斷(Trap) (94ncu、96tpu、92nttu)

68

Buffer Overflow 機制

- 先舉一個例子說明什麼是 Buffer Overflow :

```
void function(char *str){
        char buffer[16]; strcpy(buffer,str);  }
void main() {
  char large_string[256]; int i;
  for( i = 0; i < 255; i++) large_string[i] = 'A';
  function(large_string);
}
```

69

Buffer Overflow 機制

- 這段程式中就存在 Buffer Overflow 的問題，傳遞給function的字串長度要比buffer大很多,而function沒有經過任何長度校驗直接用strcpy將長字串拷入buffer。如果執行這個程式的話，系統會報告一個Segmentation Violation 錯誤。
- 為什麼會這樣? 未執行strcpy時堆疊(stack)中的情況:

```
   16        4        4              4
...[buffer] [ebp] [ret位址] [large_string位址]
|              |
esp            ebp
```

70

Buffer Overflow 機制

- 當執行 strcpy時, 程式將256 Bytes拷入buffer中，但是buffer只能容納16 Bytes,那麼這時會發生什麼情況呢? 因為C語言並不進行邊界檢查, 所以結果是buffer後面的250位元組的內容也被覆蓋掉，這其中自然也包括ebp, ret位址，large_string位址。因此 ret位址變成了0x41414141h ,所以當過程結束返回時, 它將返回到0x41414141h位址處繼續執行,但由於這個位址並不在程式實際使用的記憶體空間範圍,所以系統會回報Segmentation Violation.

71

Buffer Overflow 機制

- 透過Buffer Overflow來改變在堆疊中存放的過程返回位址,從而改變整個程式的流程,使它轉向任何我們想要它去的地方.這就為駭客們提供了可乘之機, 最常見的方法是: 在長字串中嵌入一段程式碼, 並將過程的返回位址覆蓋為這段程式碼的位址, 這樣當過程返回時,程式就轉而開始執行這段自編的程式碼。一般這段程式碼都是執行個Shell程式(如\bin\sh), 當要入侵一個帶有Buffer Overflow缺陷且具有suid-root屬性的程式時, 會獲得一個具有root權限的shell,在這個shell中可以幹任何事。這段程式碼一般被稱為Shell Code.

72

I/O 保護

- 避免使用者的程式很有可能發出不合法的 I/O 指令。
- I/O 指令被定義成特權指令(Java不允許不當存取)(91nttu)
  - 使用者不能直接使用 I/O 指令，必須透過系統呼叫來完成
- 作業系統必須代替使用者處理每個 I/O 指令
  - 為了效率考量，系統無法檢查所有可能被入侵的地方，各個I/O中斷處理還是可能被不當入侵，而處在系統模式，以獲得較高的系統控制權，所以需要考量安全問題

73

CPU 保護

- 如果使用者執行無窮迴圈而拒絕讓出 CPU 的使用權，其他程式就無法被執行。
  - 用計時器來保護 CPU，於特定時間發出中斷，作業系統就可用計時器的中斷來取得控制權，並決定下一個使用CPU的程式
- 倒數計時器(one-shot timer)
  - 計時器倒數到設定的時間到後發出中斷，下次中斷需要由程式重新設定才會再開始進行。

74

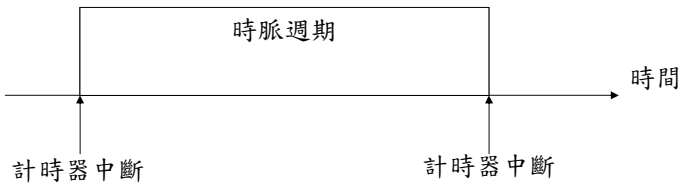
CPU 保護

- 週期計時器
  - 會自動重新設定同一段時間後再發生中斷。
- 可變週期計時器
  - 為週期計時器的值乘上時脈週期(Linux)
- 計時器可以用來實作分時系統。
- 將時間均分成時間切片(Time slice)，此時間為一個程式每次所能執行的時間，當程式執行的時間超過時間切片，作業系統將CPU控制權交給另一個使用者，這個動作稱為內文切換(Context switch)(93ncu)

75

時脈週期

- 週期計時器的週期又稱為時脈週期



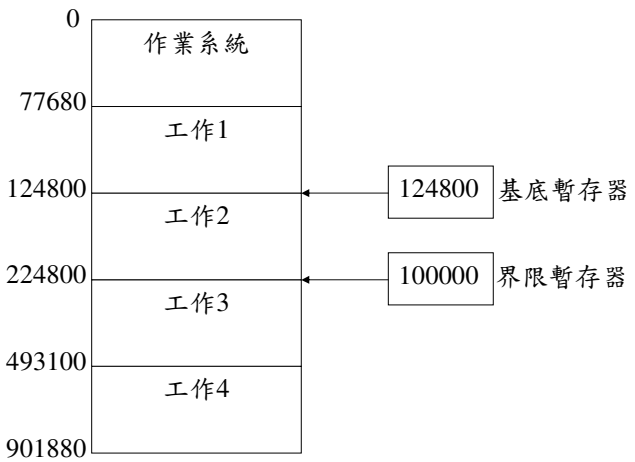
76

記憶體保護

- MS-DOS 是個很簡單的作業系統
  - 使用者程式可以修改作業系統在記憶體中的程式碼或是資料區段。
- 可以用額外的硬體來解決，兩個暫存器的值，代表目前執行的使用者程式能存取的記憶體範圍(96nttu、92nttu)
  - 基底暫存器 (EBX)➔開始位址
  - 界限暫存器 (EBP) ➔大小
- 只有特權指令才能夠修改基底暫存器跟界限暫存器

77

基底暫存器與界限暫存器(124800-224799)



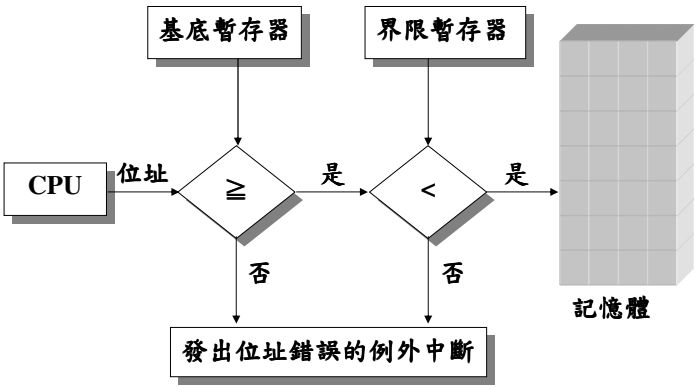
78

基底暫存器與界限暫存器

- CPU每次存取記憶體都會去檢查是否超過使用者可以存取的範圍，若超出範圍，則硬體會引發例外中斷來通知作業系統，如此避免使用者程式蓄意非法修改作業系統或其他使用者的程式或資料
- 只有特權指令才可以修改基底暫存器與界限暫存器的值，所以只有在系統模式下執行

79

基底暫存器與界限暫存器的硬體記憶體位址偵測



80

摘要

- 作業系統與硬體
  - 硬體的架構決定了作業系統各方面實作的方法。
  - 使用非同步 I/O，讓 CPU 與週邊裝置的工作能夠同時進行，以增進效率。
  - CPU 忙碌於執行工作時，DMA 分攤 CPU 傳遞資料到記憶體的工作。
- CPU
  - 執行指令和存取記憶體是 CPU 對外控制的兩個管道。
  - 記憶體對映 I/O 可以讓 CPU 直接控制週邊裝置，否則就必須靠特殊的指令。
  - 主動的方式— CPU 以輪詢的方式查看週邊裝置是否需要服務。
  - 被動的方式—週邊裝置以中斷通知 CPU 目前有工作需要進行。

81

摘要

- 儲存階層
  - 在儲存階層的頂端，其執行速度與價格較高
  - 隨著階層往下，儲存裝置的速度與價格也遞減。
  - 除了速度與價格需要考量之外，也要考慮每種儲存裝置的物理特性
    - 揮發性的記憶體在斷電後資料就會消失，但往往其執行速度也比較快。
    - 非揮發性記憶體就比較適合用來儲存資料。

82

摘要

- 硬體保護
  - 雙模式運作—避免使用者的程式干擾系統的運作。
  - I/O 保護—特殊指令成為特權指令。
  - CPU 保護—使用計時器防止 CPU 被單一使用者佔據。
  - 記憶體保護—利用特殊的基底暫存器與界限暫存器加以保護記憶體。

83