系統程式

第1章. 作業系統概論

1.1. 系統架構

1.1.1.系統呼叫

系統呼叫提供一個由作業系統服務的介面。這類呼叫一般以 C 或 C++寫成的常式，有時候低階的工作會由組合語言來寫。系統呼叫再被包裝成 API，由應用程式設計者來呼叫。

系統呼叫的類型有以下五種

一 行程控制

- 正常結束，終止執行
- 載入，執行
- 建立行程，終止行程

二 檔案管理

- 建立，刪除檔案
- 開啟，關閉
- 讀出，寫入，重定位子

三 裝置管理

- 要求裝置，釋回裝置
- 讀出，寫入，重定位子
- 獲取裝置屬性，設定裝置屬性

四 資訊維護

- 取得時間或日期，設定時間或日期
- 取得系統資料，設定系統資料
- 取得行程、檔案或裝置的屬性

五 通信

- 建立，刪除通信連接
- 傳送接收訊息
- 傳輸狀況訊息

記憶法：裝檔行通知

1.1.2.行程間通訊

行程有兩種常用的通信方法，訊息傳遞與共用記憶體模式。訊息傳遞方式只有在少數資料需要交換時比較有用，而且對電腦之間的通信而言，也比共用記憶體方式容易製作。共用記憶體方式可取得最大的通信速度與便利性，但保護跟同步上會有很多問題需要克服。

♦ **訊息傳遞模式**

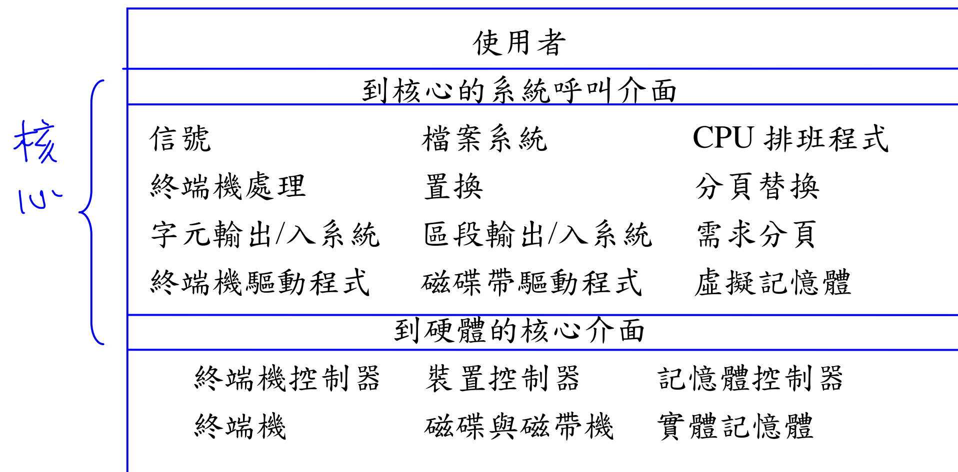
- 資訊藉由作業系統中所提供的行程間聯繫來做交換。

♦ **共用記憶體模式**

- 資訊藉由 share memory create 和 shared memory attach 系統呼叫來產生以及取得其他行程所擁有的記憶體區域的存取權。因此要預防共用所會產生的問題，例如用 critical section 來解決，而確保不會對同一個區域進行寫入的動作

傳遞參數給 system call 的方法
- 靠暫存器傳遞參數.
- 將參數放在記憶體的表格中，然後將表格的記憶體位址放入一個暫存器來傳遞.
- 程式將所有參數推入 Push (store) 堆疊(stack)中，然後作業系統再將堆疊資料抽出(pop off).

1.1.3.作業系統的分層方法



作業系統的核心是指 system call 界面以下，到硬體以上的部份。而分層方式是達成模組化的一種方式。

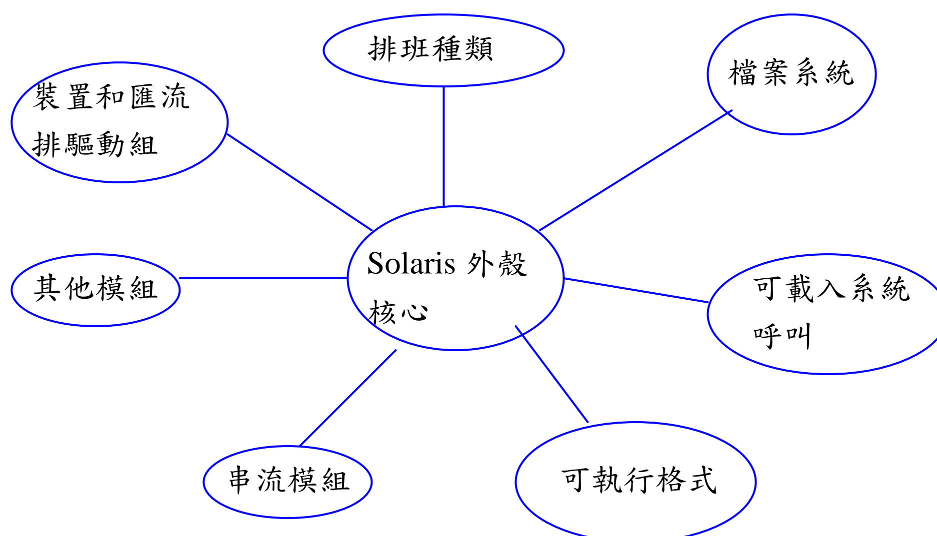
1.1.4.作業系統的微核心

微核心經由 message passing 提供通訊。客戶程式和服務者從未直接通訊，因為他們藉由微核心交換訊息達到非直接通訊。

♦ **優點**
- 作業系統容易擴展
- 所有的新服務都加入使用者區域，所以不要求核心修改
- 提供安全性與實用性，大部分服務在使用者行程執行，而不是在核心

1.1.5.作業系統的模組

Solaris 作業系統結構外殼核心組織，四周以七個類型的可載入模組所組成。



1.1.6.DMA(Direct Memory Access)

一　DMA（Direct Memory Access）

- 是一種輸出入方式，允許週邊裝置藉著擷取 CPU 執行週期的方式（稱為 cycle stealing）來直接與記憶體做存取的動作。

二　指令週期（instruction cycle）

- 是指要執行一個機器指令時所需的動作順序。

三　Cycle stealing

- 在指令週期內，由 CPU 利用一部分中斷週期去詢問是否有中斷服務常式要處執行，控制權並沒有真正交給另一個行程。

1.2.　作業系統基本觀念

1.3.　作業系統的演進

一　Monitor

♦ **目的**

- 爲了減輕操作員的負擔

♦ **工作**

- 自動工作排序
- 裝置驅動程式
- 中段處理
- 解譯 control card 與 control command

二　On-line system 與 Off-line system

♦ **On-line System**

- 意義：CPU 與 I/O 裝置直接連接
- 缺點：CPU 會一直 idle，等 I/O 的時間

♦ **Off-line System**

- 意義：CPU 與 I/O 裝置加上磁帶
- 缺點：僅適用於 direct access

三　Buffering

♦ **意義**

在主記憶體中劃出一個區域，做為存放輸出入資料的緩衝區

◆ **限制**

Buffering 是應用在對同一個 job 的輸出入處理與 CPU 計算動作重疊的方法

◆ **理想**

期望能同時處理 CPU 與 I/O 動作來提高系統效率

◆ **實際情況**

CPU 計算遠大於 I/O 速度，所以實際情況可能不如預期

四 SPOOLING(Simultaneous Peripheral Operations On Line)

◆ **意義**

用 disk 來取代 off line system 中的 tape

◆ **作法**

將磁碟劃分成數個區域，每個區域代表不同的輸出入裝置，，系統中的程序如果要做 I/O 動作，則將對實際 I/O device 的動作轉換成對磁碟上相對應的 buffer 來做動作，這樣的作法能夠達到真正的 multiprogramming 的目的，因為實際的 I/O 速度是遠不及 CPU 的；透過 SPOOLING 的觀念，則可使 CPU 的速度不受週邊設備之速度的限制，以使得系統的效率能夠因為 multiprogramming 觀念的引進而真正提高，這種利用磁碟來模擬輸出入裝置的觀念，就是 virtual device 的觀念

◆ **使用 SPOOLING 的理由**

- 可以 direct access
- 存取的速度較快

◆ **優點**

- 調和 I/O 與 CPU 的速度
- 提供真正 multiprogramming 的功能
- Device independent

1.4. 多程式系統

◆ **多程式系統的意義**

記憶體中存在多個程式，而這些程式以輪流的方式來利用 CPU 執行其程式段，這樣的系統稱之為多程式系統。

◆ **發展多程式系統的原因**

爲了讓程序一直在利用資源，則在記憶體中載入多個程式，將資源分配給這些程式使用，以免記憶體中只有一個程序在使用某一種資源，造成資源的浪費。

1.5. 分時系統

1.6. 即時系統

1.7. 分散式系統

1.7.1.分散式系統的種類

- Tight coupled system
- Loosely coupled system

1.7.2.分散式系統的功能

- 資源共享
- 加快處理速度
- 通訊
- 可靠

範例1.
何謂 heterogeneous distributed system？
何謂 process migration？它主要的目的是什麼？

- ◆ **Process migration 的目的是用來做 load balancing 工作**

基本上有兩個層次的 load balancing, 一種是 system level 自動作的，它是透過 process migration 的方式將 load 較重的 node 上的部分 processes 移到 load 較輕的 nodes 上去執行，當然這種方式只有對 multi-job 的情況有用。
若是本來每個 node 上就只有一個 process 而已，那就無從移起了，這種情形下就需要作 application level 的 load balancing 了，也就是必須改變 program(process 的工作)的安排。在寫 program 時就特別透過 data partition 或其他方法將 computing load 依 nodes 的 computing power 適當分配

1.8. 作業系統資源保護問題

1.8.1.輸出入動作的保護

一　系統呼叫

系統呼叫的類型

- ♦ **行程控制**
  - 正常結束，終止執行
  - 載入，執行
  - 建立行程，終止行程
- ♦ **檔案管理**
  - 建立，刪除檔案
  - 開啟，關閉
  - 讀出，寫入，重定位子
- ♦ **裝置管理**
  - 要求裝置，釋回裝置
  - 讀出，寫入，重定位子
  - 獲取裝置屬性，設定裝置屬性
- ♦ **資訊維護**
  - 取得時間或日期，設定時間或日期
  - 取得系統資料，設定系統資料
  - 取得行程、檔案或裝置的屬性
- ♦ **通信**
  - 建立，刪除通信連接
  - 傳送接收訊息
  - 傳輸狀況訊息

記憶法：裝檔行通知

傳遞參數給 system call 的方法

- 靠暫存器傳遞參數.
- 將參數放在記憶體的表格中，然後將表格的記憶體位址放入一個暫存器來傳遞.
- 程式將所有參數推入 Push (store) 堆疊(stack)中，然後作業系統再將堆疊資料抽出(pop off).

二 雙重模式

將系統分為 user mode 與 monitor mode 兩種模式，而指令亦分為 user instruction 與 privileged(特權) instruction，其中特權指令只有可以在 monitor mode 下面執行。

1.8.2.記憶體的保護

一 一般區域與監督器(monitor)之保護問題

利用 fence register

二 保護金鑰法(Protection Key)

記憶體中每個區塊都有鎖，只有與鎖相同 Key 值的 process 能夠存取

三 Bound register

Lower bound register 與 Upper bound register

四 Base register 與 limit register

Base 與 limit register

1.8.3.CPU 的保護

避免 CPU 被不當的利用，例如 infinite loop。解決方法是利用一個 timer，一段時間就檢查一次，看看 CPU 是否在不正常的狀態。

1.9. 中斷

1.9.1. 中斷與 trap

一 中斷

現在的作業系統是中斷驅動式(interrupt driven)。如果沒有行程要執行，沒有 I/O 裝置要服務和沒有使用者需要回應，則作業系統將安靜的進入等待事件的發生。

二 Trap

Trap 又稱為 exception，是一種軟體的中斷，它是因為錯誤(ex.除以 0 或不正確的記憶體存取)或是由使用者程式提出需要作業系統服務的特定要求所產生

1.9.2. 中斷的種類

一 外部中斷

- I/O interrupt：由輸出入裝置所發出的中斷
- Machne check interrupt

二 內部中斷

運算式除以 0

三 軟體中斷

由指令所引發的中斷，其目的是為了要保護系統資源。例如 system call，又稱為 Supervisor call(SVC)

1.9.3. 中斷發生的處理步驟

- OS 取得 CPU 的控制權，並將中斷發生前，執行程序的狀態儲存起來
- 取得中斷處理程式的位址，並執行中斷處理程式
- 回復中斷發生前的狀態，並恢復程序之執行

1.9.4. 同步中斷與非同步中斷

一 同步中斷

- 當同步中斷發生時 CPU 必須立即處理
- 又稱作 enable 或 unmask interrupt
- 與硬體有關的中斷

二 非同步中斷

- 當非同步中斷發生時，CPU 可以忽略或延遲非同步中斷的處理
- 又稱作 disable 或 mask interrupt
- 與軟體有關的中斷

1.10. 陽春機器與延伸機器的意義

1.11. 考古題

範例2.

請說明兩種命令解譯器實做實做的方式

Ans:

- Command interpreter 包含了 command 命令段
- 未包含，要用時才把相對應的程式段 load 進去記憶體裡面執行(ex. shell)

第2章. 程序排程

2.1. 程序基本觀念介紹

2.2. 排程概念介紹

範例3. 定義短期、中期和長期排程

Ans:

◆ **短期排程**

短期排程器又稱為程序排程器(process shceduler)，其作用是由已載入主記憶體中的程序中，挑出一個程序來執行

◆ **中期排程**

主要用來調節系統的負載，將行程從記憶體中有效的移開(並且從對 CPU 的競爭中移開)，而降低原多城市規劃的程度，使系統的效率始終能維持正常情況。稍後，再把該行程放回記憶體中繼續執行，這種方法稱之為 swaping。

◆ **長期排程**

長期排程又稱作 job schedule，其作用是將程序由 secondary storage 中，載入主記憶體中，即進入 ready 狀態。

2.3. 本文切換(Context Switching)

2.3.1.Context Switching 的意義

OS 儲存目前正在執行的程序的狀態，並將下一個要執行的程序的狀態載入系統並開始執行

2.3.2.PCB(Process Cotrol Block)

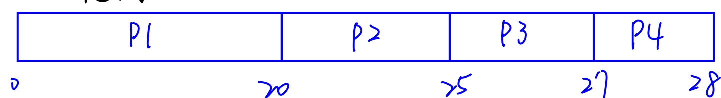
紀錄以下資訊

- 行程標籤
- 行程狀態訊息
- 行程控制訊息

2.4. 排程方法

2.4.1.FCFS

一 範例



◆ **平均迴轉時間**

(20+25+27+28)/4

♦ 平均等候時間

(0+20+25+27)/4

二 缺點

可能會造成護航效應

2.4.2.Shortest Job First

事實上程序的下一個 CPU burst time 是無法預知的，但對固定的 Job 而言，SJF 擁有最短的平均等候時間

2.4.3.Preemptive Shortest Job First

當新的程序的 CPU burst time 較目前執行的程序所剩餘的 CPU burst time 還短的時候，CPU 將配置給新的程序來使用

2.4.4.Priority schedule

2.4.5.Round-Robin scheduling-RR

每個程序都配給一個 time quantum，當 time quantum 用完的時候(run->ready)或執行 I/O 工作，系統會將 CPU 的使用權交給在 ready queue 的下一個程序來使用

2.4.6.Multi-level queue

將不同特性的程序，依照其特性，分成不同的類別。每個類別有各自的排程法，當程序被歸類為某個類別之後，就不可以在更換類別。

2.4.7.Multi-level feedback queue

原則上與上面相同，但允許程序在不同 q 裡面移動，所以比較有彈性。

2.5. Thread

對於執行緒的支援，可以由使用者層次提供(user thread)，或是由核心提供(kernel thread)

2.5.1.User level thread

利用一個 run time library package 提供 thread 執行時所需要的 routines，當 thread 需要支援時，就會呼叫 library 中的某些 routine 來提供服務

♦ **優點**

- OS 的 kernel 不用修改來 support user-level thread 的需求。(因為所有支援 user-level thread 的程式段是一個有別於 kernel 的程式段)
- Cost 較低，相當於一次 procedure call 的 cost
- Performance 較佳
- 彈性較佳，因為可視 user 的需求來調整 library 中 routines 的內容，因此具有較高的彈性

2.5.2.kernel level thread

OS 的 kernel 中包含了 thread 相關的程式段，kernel 會直接 support thread 所需要的所有服務

♦ **優點**

- Thread 間的 synchronization 與 scheduling 較容易達成，因為 kernel 中含有 threads 的所有資訊

♦ **缺點**

- Overhead 較 user level thread 高：因為每次的 operation 均會導致一個 kernel trap，由 kernel

來負責查核動作

- 彈性較差：即使某個 thread 不需要某項服務，kernel 也需要包含該項服務

第3章. 記憶體管理

3.1. 記憶體的層次結構

3.2. 記憶體保護法

3.3. 裸機

3.4. 單一使用者記憶體配置方式

3.5. 重疊置換(Overlapped Swapping)

3.6. 記憶體碎裂(fragmentation)

3.7. 固定分割法記憶体管理

3.8. 變動分割法記憶体管理

3.8.1.Merge

將兩個或兩個以上相鄰的 hole 合併成為一個大的 hole

3.8.2.Compaction

將記憶體中可用的 hole 聚集成為一個比較大的 hole，這樣的動作就是 compaction

- **優點**
  - 可解決外部碎裂
  - 可增加多程式度
- **缺點**
  - Compaction 動作耗時甚鉅
  - 僅適用於 dynamic relocation system
  - 可能執行完 compaction 動作後仍然找不到足夠大的 hole 可以用

3.9. 多重基底暫存器

3.9.1.採用多重基底暫存器的理由

將程式分成數個部分，如此較容易找適當 hole

3.9.2.作法

- 分成 read-only 與 writable 記憶體兩部份
- 分成程式碼與資料兩部份

3.9.3.特性

可重入程式(reentrant program) 就是根據這種方法製作

3.10. 分頁記憶體管理

分頁記憶體管理法允許程式在記憶體中佔用的記憶體位子可以不連續

3.10.1. 分頁的意義

- 將實際記憶體區分成為固定大小的 frame
- 將程式區分成固定大小的 page
- Frame 與 page 大小相同
- 程式的任何一個 page 可以載入實際記憶體中任何一個 frame

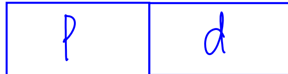
3.10.2. 位址轉換問題

一　位址轉換目的

- 將邏輯位址轉換成實際位址

二　邏輯位址的內容由兩部分組成

- 頁碼
- 頁位移



三　分頁表(Page Table)

分頁表的內容為 Page 對應到的 frame。因此要得到實際記憶體中的位址，有以下兩個步驟

- 將 page 換成 frame
- Frame 加上位移，就是在實際記憶體中的位址。

3.11. 分頁表製作問題

3.11.1. 直接對應方式

一　硬體製作

♦ **作法**

- 利用暫存器來儲存分頁表

♦ **特性**

- 程式切換時，必須將資料重新載入暫存器中
- 適用較小的分頁表
- 成本太貴

二　儲存在主記憶體中

♦ **作法**

- 利用主記憶體來儲存分頁表

♦ **特性**

- 程式切換時，僅需改變 PTBR(Page Table Base Register)的內容
- 適用於較大的分頁表
- 成本較低
- 但是必須做兩次記憶體存取

3.11.2. 關聯對應方式

一　轉譯旁觀緩衝區

轉譯旁觀緩衝區(Translation look-aside buffer, TLB)是一種特殊的小型硬體快取記憶體，由相關聯的高速記憶體組成，每個暫存器包含了兩個部份：Key 和 Value。

二　作法

- 分頁表存進去 TLB 裡面，key 代表 page，value 代表 frame。
- 找到 frame 的話直接輸出(花一次記憶體存取時間+原來的 TLB cache 存取時間)
- 如果找不到，再去記憶體裡面的分頁表抓(會花兩次記憶體存取時間+原來的 TLB cache 存取時間)，這也就是結合直接對應與關聯對應的方法。

3.12. 共用分頁問題

3.13. 分段記憶體管理

3.13.1. 使用者的觀點

記憶體由大小不固定的區段所組成

3.13.2. 分段的特性

- 分段的大小不固定
- 分段無順序性
- 一個記憶體單元，其實就是一個分段，例如副程式、變數宣告區或是堆疊等等。

3.13.3. 分段記憶體管理法邏輯位址的內容

段碼　位移

s	d

logical address

3.14. 分段表的製作

3.15. 分頁與分段比較

3.16. 分頁分段法

3.17. 分段分頁法