Operating Systems
作業系統

PROCESS
行程

行程

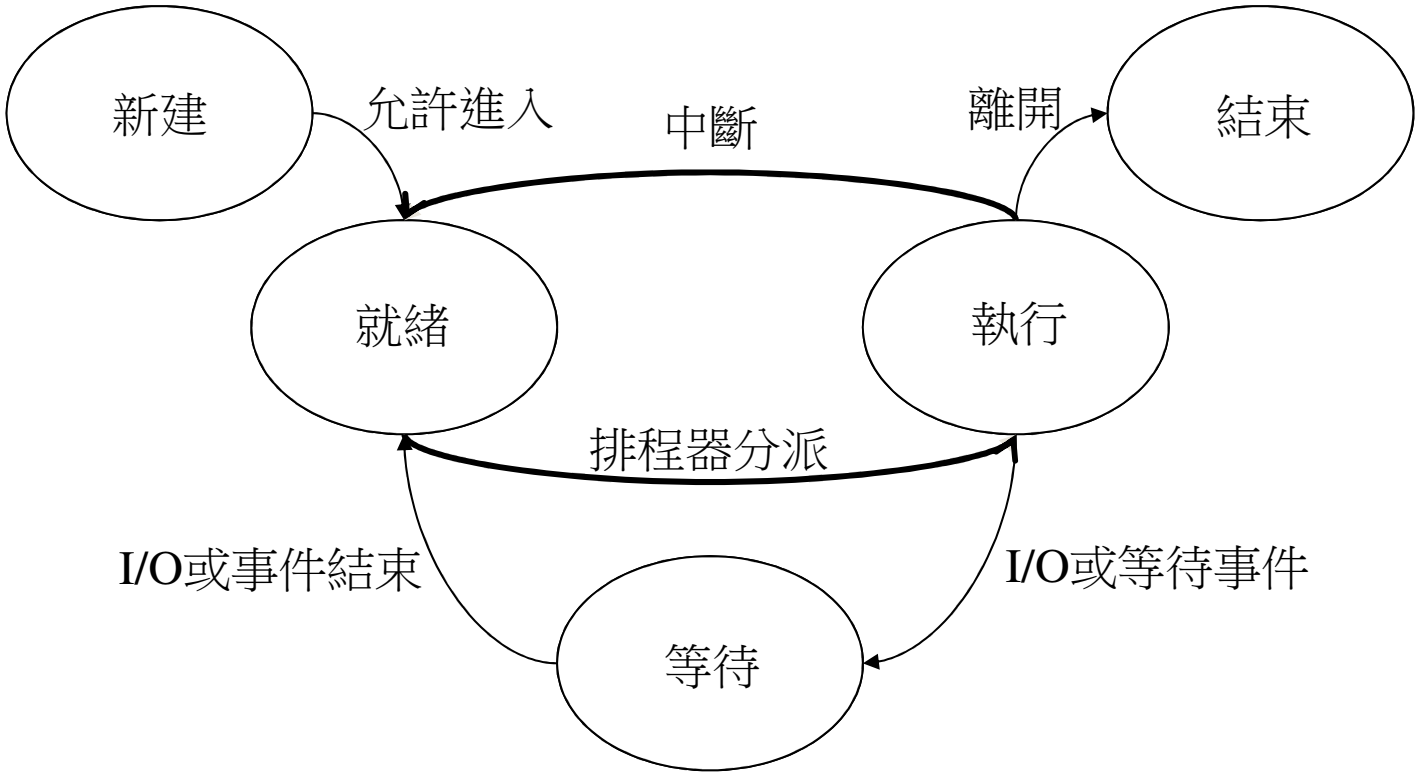
- 行程概念
  - 行程簡介
  - 行程的狀態
  - 行程控制區塊
- 行程排程
- 行程的建立與結束
- 執行緒
- 行程合作
- 行程間溝通
- 摘要

行程概念 (Concept)

- 行程與程式主要的不同點：
  - **程式**是被放在外部的儲存裝置如磁碟上，而**行程**則被放在記憶體中。
  - 程式在儲存裝置中是**靜態的**，而行程在記憶體中是**動態的**，它會隨著一些事件的發生而產生相對的改變。
- 行程，簡單來說，就是一個執行中的程式。
- 一個行程包括了
  - 相對應的程式碼
  - CPU 中各暫存器的值
  - 行程堆疊
  - 資料區段

行程的狀態 (State)

- 一個行程在執行過程中，會改變很多狀態。
- 一個行程的狀態通常有下列幾種：
  - 新建
  - 執行
  - 等待
  - 就緒
  - 終結

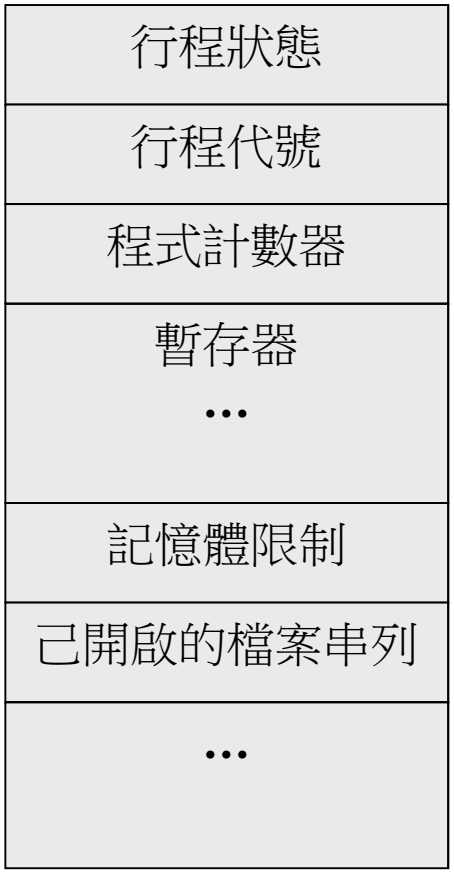


行程狀態圖

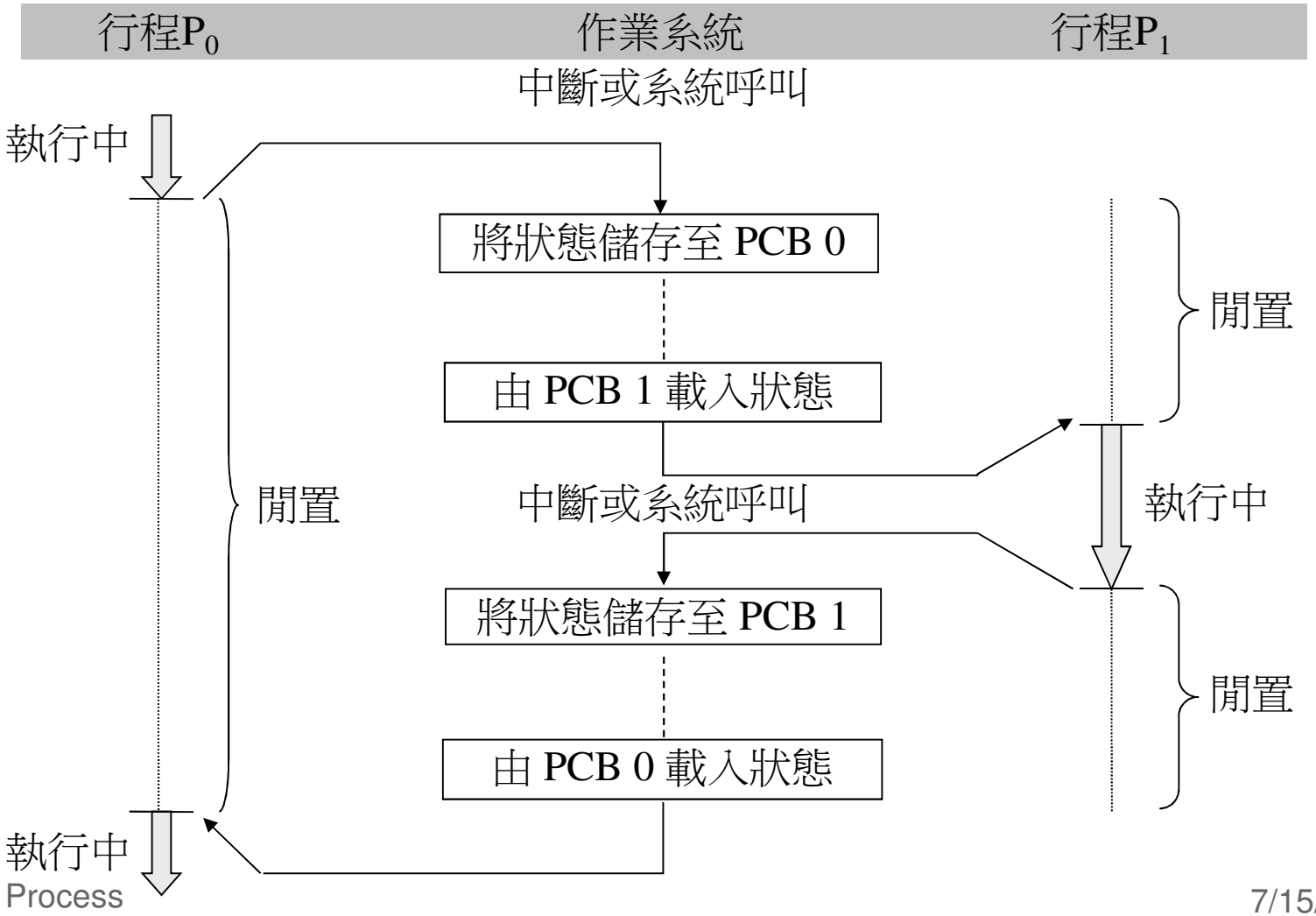
行程控制區塊

- **行程控制區塊**（Process Control Block, PCB），儲存行程在執行時相關的資訊。
- PCB 中通常包括了
  - **行程狀態**
  - CPU **暫存器**
  - **排程資訊**
  - I/O 狀態
- 當行程進行切換時，需要將目前行程的相關資訊記錄在該行程的 PCB 中，並將另一個行程的 PCB 載入至系統中，這個動作稱為**內文切換**。

行程控制區塊



行程的切換


行程P_0
作業系統
行程P_1
中斷或系統呼叫
執行中
將狀態儲存至 PCB 0
閒置
由 PCB 1 載入狀態
閒置
中斷或系統呼叫
執行中
將狀態儲存至 PCB 1
閒置
由 PCB 0 載入狀態
執行中

行程

- 行程概念
- **行程排程**
  - **排程佇列**
  - **排程器**
  - **內文切換**
- 行程的建立與結束
- 執行緒
- 行程合作
- 行程間溝通
- 摘要

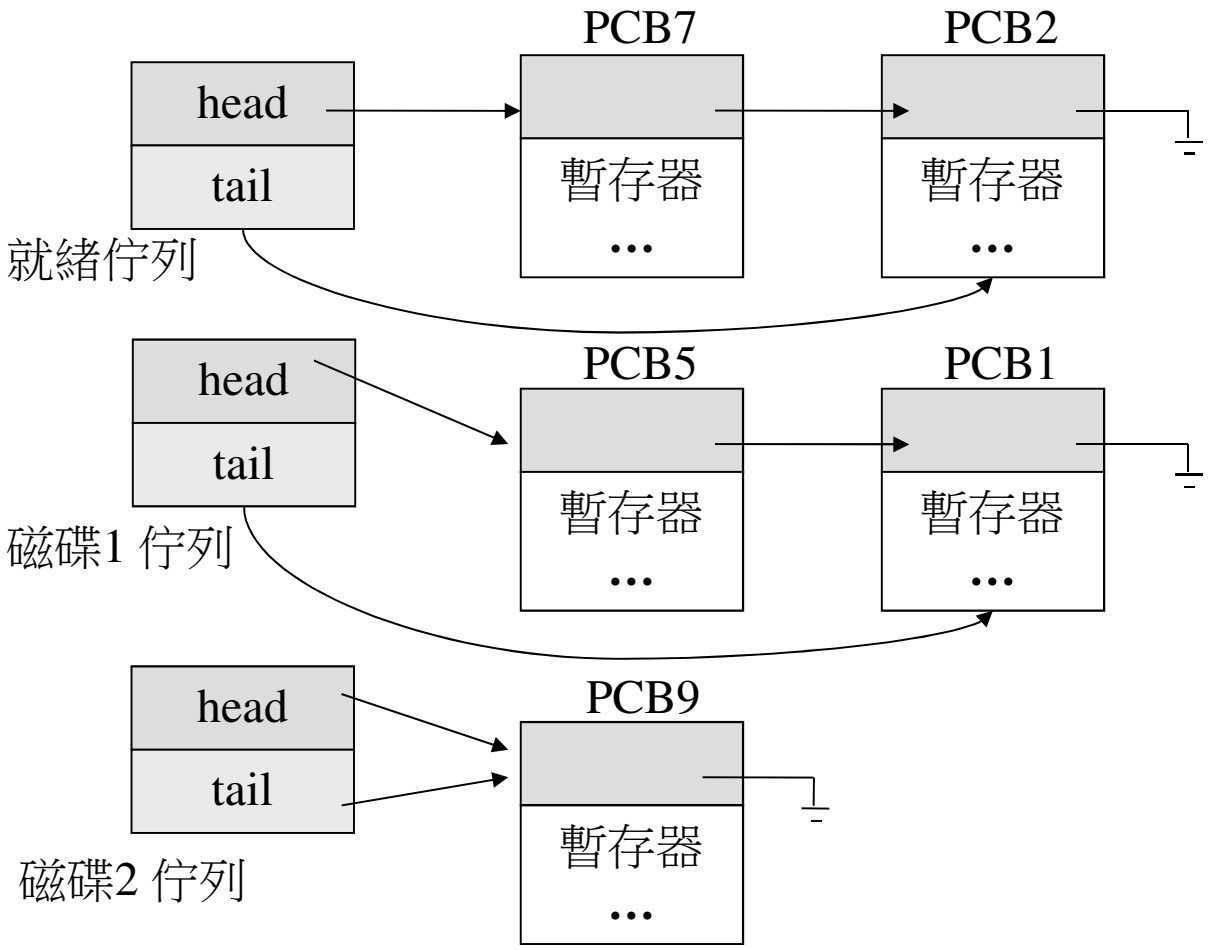
行程排程 (Scheduling)

- 為了增加 CPU 的**使用效率**而提出多個行程的觀念。
- 一個單 CPU 的系統來說，隨時只能有一個行程在執行。
- 其他行程則必須等待 CPU 空閒下來，然後再經由排程器選出，才能取得 CPU 的使用權。
- **如何排程**是影響**作業系統效能**最重要的因素。

排程佇列

- 一個行程在執行期間會在各種不同的佇列中進出。
- 一個系統中通常有
  - 工作佇列
  - 就緒佇列
  - 等待佇列
  - 裝置佇列

就緒佇列與裝置佇列


PCB7
PCB2
head
tail
暫存器
...
暫存器
...
就緒佇列
PCB5
PCB1
head
tail
暫存器
...
暫存器
...
磁碟1 佇列
PCB9
head
tail
暫存器
...
磁碟2 佇列

排程器 (Schedulers)

- 作業系統中主要的排程器有：
  - 長程排程器 (分鐘 minute)
    - 控制系統多工的程度 (degree of multiprogramming)
  - 短程排程器 (毫秒 millisecond)
- 長程排程器和短程排程器最大的不同點
  - **執行的頻率**
- **中程排程器**最主要的用途在於**降低系統多工**的程度，以增加系統可用記憶體的大小。

內文切換 (Context Switch)

- 當 CPU 的使用權由一個行程轉到另一個行程時需進行**內文切換**。
- 內文切換動作所花的時間對系統而言是額外的負擔 。
- 執行緒降低內文切換所花的時間。

行程

- 行程概念
- 行程排程
- **行程的建立與結束**
  - **行程的建立**
  - **行程的結束**
- 執行緒
- 行程合作
- 行程間溝通
- 摘要

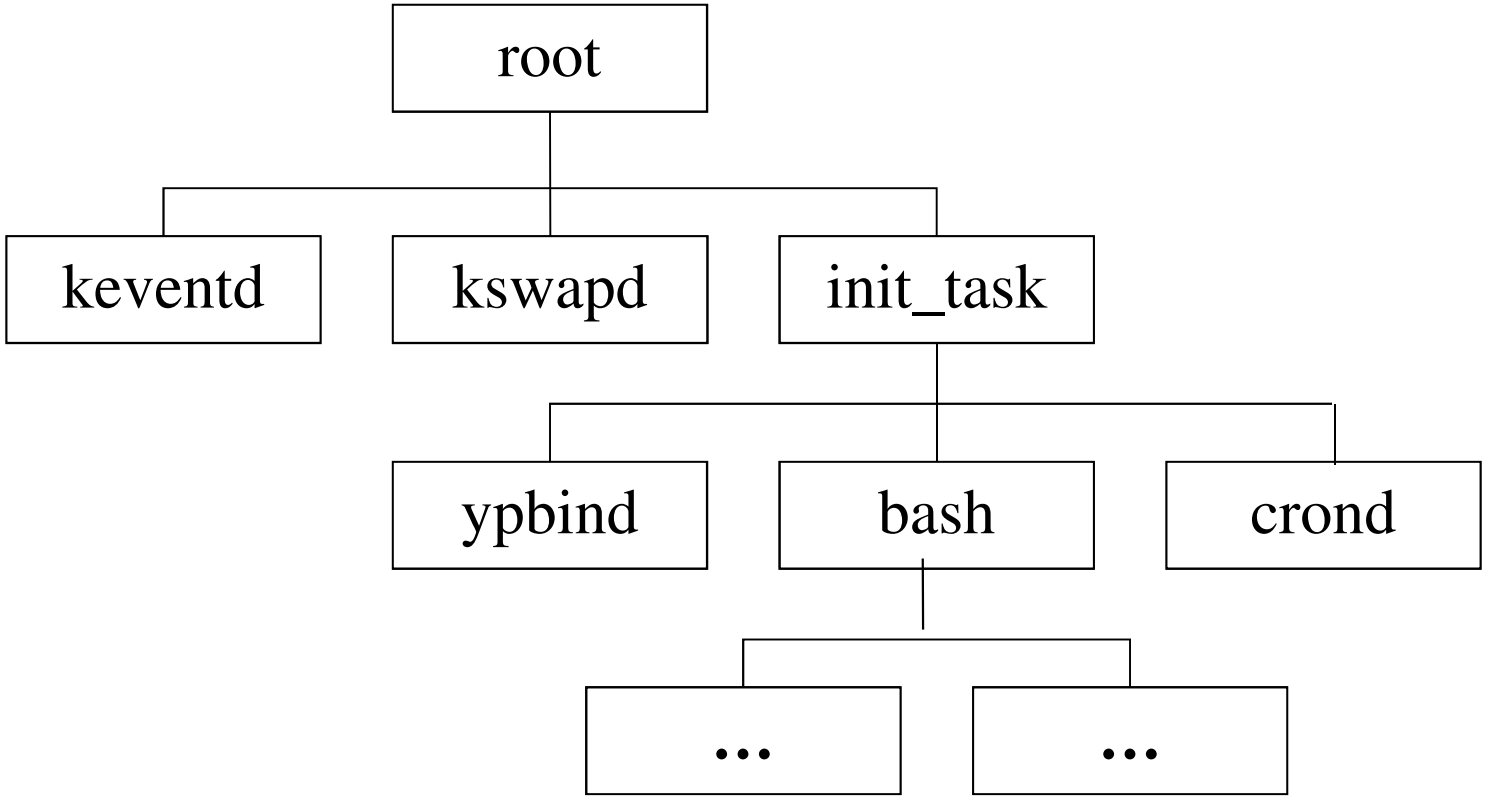
行程的建立與結束

- 不同行程在系統中可以同時執行，而且必須能動態地被**建立**與**刪除**，如此才能有效地運用或共享系統資源來完成系統的目標。
- 如何有效地建立和刪除行程也是影響作業系統效能的重要因素。

行程的建立 (Process Creation)

- 一個行程能在執行的期間透過**系統呼叫**建立很多新的行程。
- 建立新行程的行程稱為**父行程**，而新建立的行程稱為**子行程**。
- UNIX 系統中，使用行程代號（PID）來分辨不同的行程。
- 系統呼叫
  - fork()
  - execve()
  - wait()

行程樹



行程的結束 (Process Termination)

- 一個行程結束時，需要將執行期間內用到的**資源**如**實體記憶體**、**虛擬記憶體**、開啟的**檔案**和使用的I/O **裝置**等，都會交還給作業系統。
- 系統呼叫
  - exit()
  - about()

行程

- 行程概念
- 行程排程
- 行程的建立與結束
- **執行緒**
  - **執行緒概念**
  - **執行緒的優點**
  - **使用者和核心執行緒**
  - **多執行緒的模型**
- 行程合作
- 行程間溝通
- 摘要

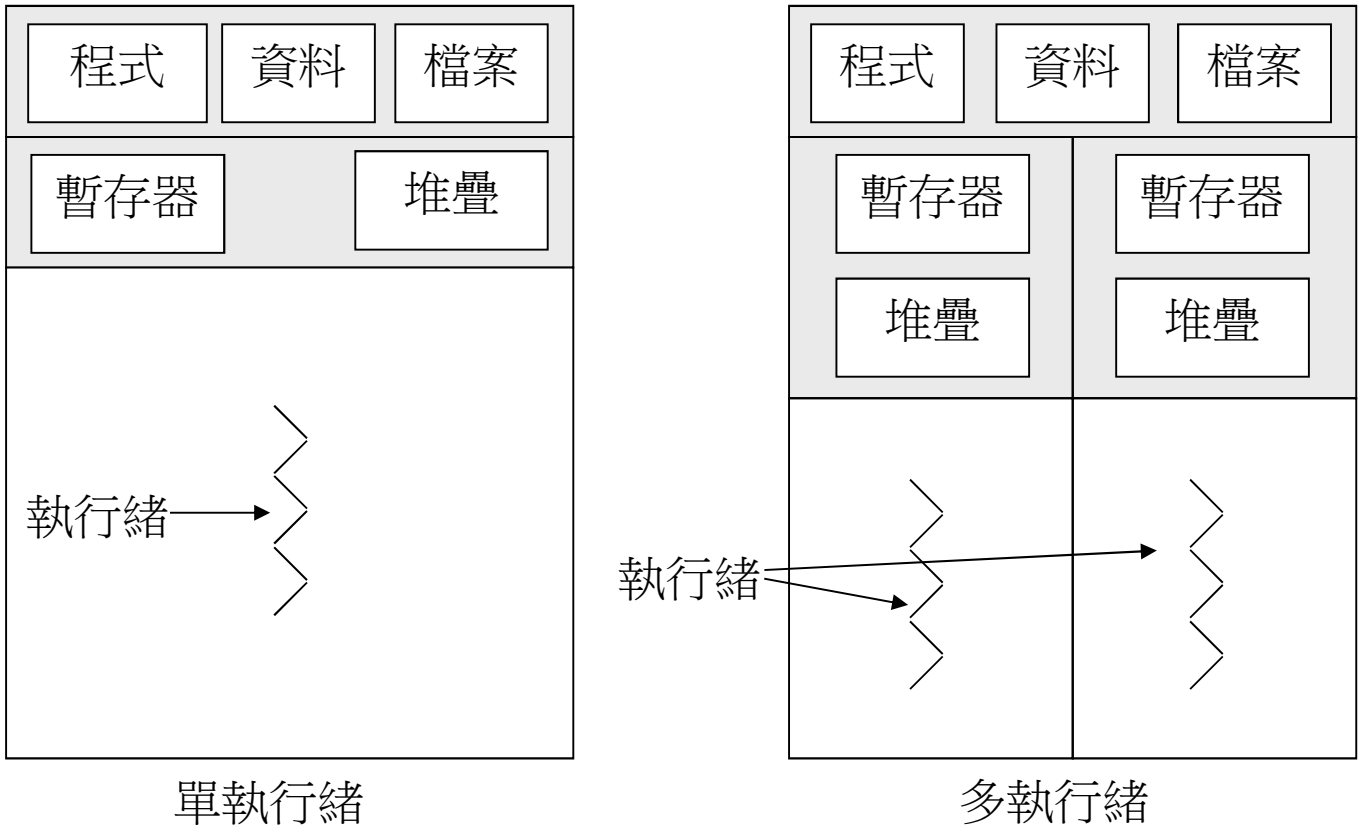
執行緒 (Thread)

- 系統呼叫 fork() 的**缺點**
  - 需要做大量記憶體的**複製**
  - 進行內文切換時需付出相當的代價
  - 兩個**行程間無法直接進行溝通**
- 若行程間可以共用一部分的記憶體空間，那麼額外的負擔就能減少，這也就是建立執行緒的基本理由。

執行緒觀念

- 執行緒
  - 輕量級行程
  - **使用 CPU 資源的基本單元**
  - 包含了一個**程式計數器**、一組**暫存器**和一個**堆疊**空間
  - 與其他的執行緒**共用同一個位址空間**
- 傳統的行程
  - 重量級行程
  - 可看成是只有一個執行緒在執行的行程

傳統行程與執行緒行程



執行緒的優點

- 使用執行緒來取代傳統行程有幾項優點：
  - 資源共享容易
  - 節省記憶體空間
  - 快速的內文切換
  - 平行處理

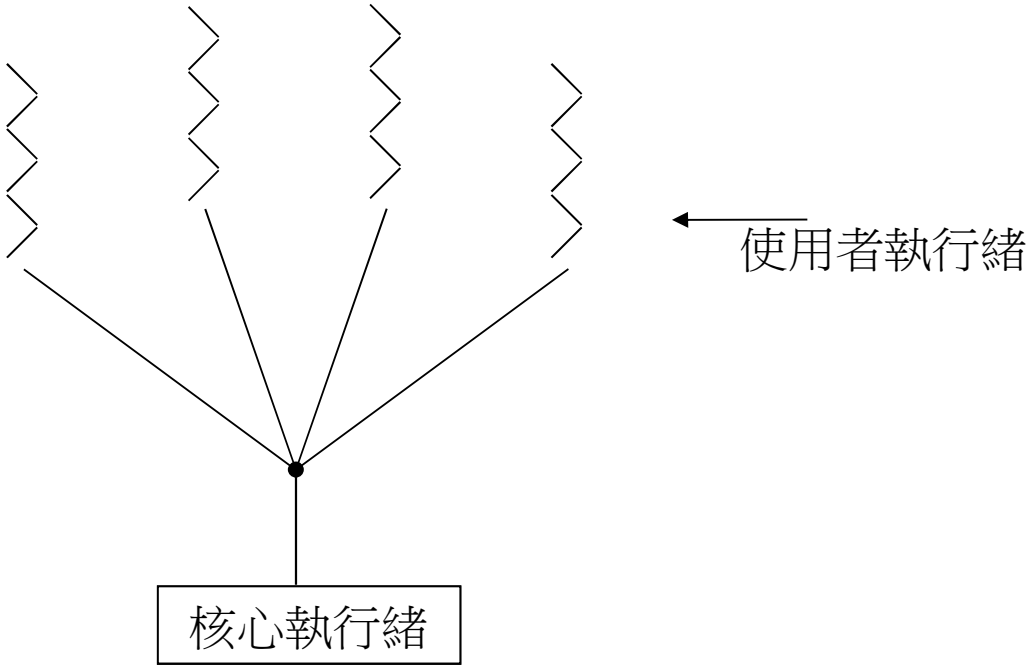
使用者和核心執行緒

- 在作業系統中，有兩種方式來**支援**執行緒
  - 使用者執行緒
    - 利用**執行緒函式庫**來提供的
    - 建立與管理執行緒時比較**有效率**
    - 若行程中的執行緒暫停，則同行程中其他所有執行緒也都會暫停執行
  - 核心執行緒
    - 由**作業系統**直接支援
    - 建立與管理執行緒時比使用者執行緒來得**慢**
    - 若行程中的執行緒暫停，核心可以安排其他在同行程中的執行緒繼續執行

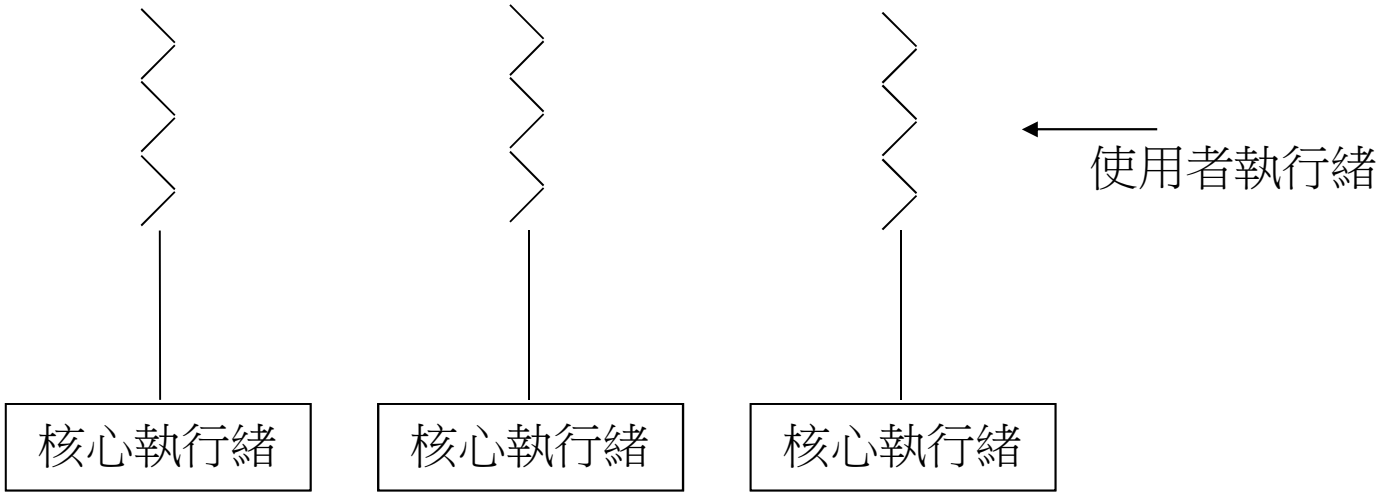
多執行緒的模型

- 實作執行緒時通常有三種模型
  - **多對一模型** - 將**許多個使用者執行緒**對應到**同一個核心執行緒**。
  - **一對一模型** - 將**一個使用者執行緒**對應到**一個核心執行緒**。
  - **多對多模型** - 將**使用者執行緒**對應到**相同**或是**較少數目**的**核心執行緒**。

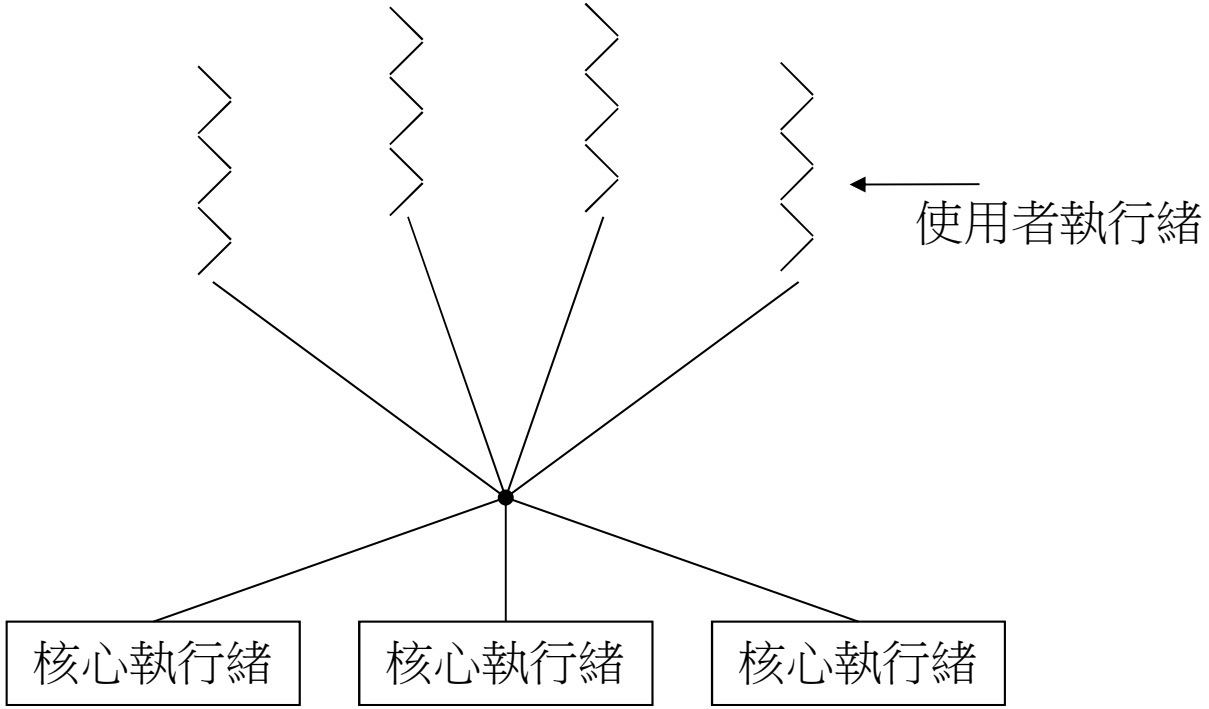
多對一模型



一對一模型



多對多模型



行程

- 行程概念
- 行程排程
- 行程的建立與結束
- 執行緒
- **行程合作**
  - **緩衝區**
  - **生產者**
  - **消費者**
- 行程間溝通
- 摘要

行程合作 (Processes Cooperation)

- 當多個行程同時在一個系統中執行時，可能會形成：
  - **多個獨立的行程** - 指行程之間沒有任何共享的資料。
  - **一些合作的行程** - 指行程之間有共享的資料。
- 典型行程合作的問題：
  - **生產者** - 產生資訊給消費者
  - **消費者** - 消耗生產者所產生的資訊

緩衝區 (Buffer)

- 當**生產者**和**消費者**同時執行時，需要有一個**緩衝區**給生產者存放產品，以提供給消費者使用。
- 緩衝區可分為：
  - 無限緩衝區
  - 有限緩衝區
- 緩衝區由作業系統提供的 **IPC (InterProcess-Communication)**機制來產生，或是直接經由**程式設計師**在程式中使用**共享記憶體**的機制來進行

生產者與消費者的例子

- 使用**共享記憶體**機制
- 生產者及消費者的行程共享了下列的變數：
  - #define BUF_SIZE
  - int iIn = 0, iOut = 0
  - user_def_type buffer[BUF_SIZE]
- 這個例子中同時最多只能使用到 n-1 個緩衝區的空間

生產者

```
do {
        …
        產生一個型別為 user_def_type 的資料並存放於 nextp
        …

        /* 若緩衝區已滿則等待 */
        while ((iIn + 1) % BUF_SIZE == iOut);

        buffer[iIn] = nextp;
        iIn = (iIn + 1) % BUF_SIZE;
}
while (FALSE);
```

消費者

```
do {
        while (iIn = iOut);
                /* 若緩衝區為空則等待 */
        nextc = buffer[iOut];
        iOut = (iOut + 1) % BUF_SIZE;
                …
        將存於 nextc 的資料消耗掉
                …
}
while (FALSE);
```

(iIn+1) % *n* = (0 + 1) % 7 = 1 != iOut = 0

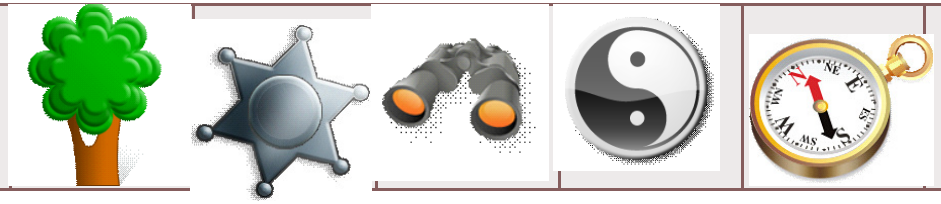
iIn = (iIn +1) % *n* = (0 + 1) % 7 = 1

n = 7
iIn = 0
iOut = 0

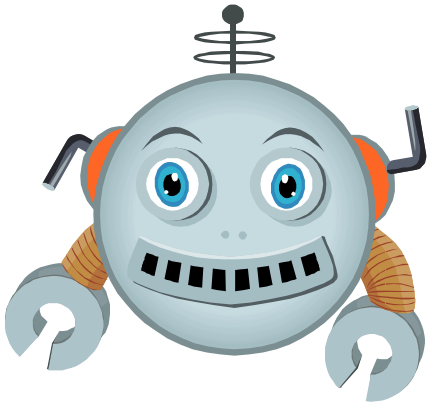
(iIn+1) % *n* = (6 + 1) % 7 = 0 == iOut =

n = 7
iIn = 6
iOut = 0

buffer



buffer[iIn = 0]



iOut = (iOut + 1) % *n* = (0 + 1) % 7 = 1

iIn = 6 != iOut = 0

行程

- 行程概念
- 行程排程
- 行程的建立與結束
- 執行緒
- 行程合作
- **行程間溝通**
  - **基本架構**
  - **直接溝通**
  - **間接溝通**
  - **同步**
  - **緩衝**
  - **例外狀況**
- 摘要

行程間溝通

- 共享記憶體利用一塊行程間**共享的緩衝區**來進行合作的溝通，而溝通的方式完全由程式設計師自行設計。
- 行程間溝通（IPC）由**作業系統**提供。
- IPC 包含了一些機制，讓行程與行程間能夠進行**溝通**與**同步**。
- 共享記憶體或是 IPC 的機制是可以同時使用的。

基本架構

- 行程間溝通（IPC）為作業系統所提供用來達到行程間資料交換與共享的機制。
- IPC 通常會提供兩個函式：
  - send() - 傳送訊息
  - recv() - 接收訊息
- 行程間在溝通時會建立一條**通訊鏈結**。透過鏈結send()和 receive() 就可以傳送和接收訊息。
- 實作傳送與接收功能時有幾種方法：
  - **直接或間接**的溝通
  - **對稱或非對稱**溝通
  - **傳送複製**或只**傳送參考**

直接溝通

- 每個行程要與其他行程進行溝通時，必須要明確地指出訊息傳送的**目的地**或接收訊息的**來源**。
- 可分為**對稱**與**非對稱**，而 send() 和 receive() 分別被定義成
  - 對稱 –
    - send(A, message)　/*傳送訊息給 A 行程*/
    - receive(B, message) /*從 B 行程接收訊息*/
  - 非對稱 –
    - send(A, message)　/*傳送訊息給 A 行程 */
    - receive(id, message)/*從任何一個行程接收一個訊息。
      其中 id 這個變數代表正在與接收端進行溝通的行程 */

直接溝通（續）

- 使用 IPC 機制程式較容易撰寫。

```
do {
        ....
        /* 生產者產生新的資料 p_product */
        …
        send(消費者, p_product);
}
while (FALSE);
```

生產者行程

```
do {
        receive(生產者, c_product);
        …
        /* 消費者消耗c_product */
        …
}
while (FALSE);
```

消費者行程

間接溝通

- 訊息的傳送與接收是透過**信箱**來完成。
- send() 與 receive() 可定義為
  - send(A, message)　/*傳送一個訊息到 A 信箱*/
  - receive(A, message)/*從 A 信箱接收一個訊息*/
- 多個行程同時**共用相同的信箱**，若多個行程同時接收訊息，那個行程會收到訊息？
  - 同時只允許兩個行程共享信箱
  - 同時只允許一個行程呼叫 receive()
  - 交給作業系統，讓作業系統決定由誰得到訊息

間接溝通（續）

- 信箱的擁有者可以為
  - 行程
  - 作業系統
- 若信箱的擁有者為作業系統，則作業系統必須提供功能讓行程去：
  - 建立一個新的信箱
  - 透過信箱去傳送和接收訊息
  - 銷毀一個信箱

同步

- 行程間可透過 send() 與 receive() 來進行同步。
- 實作 send() 與 receive() 時，可為
  - **阻隔式發送**: 送出訊息的行程會被阻隔(blocking)直到接收端或信箱收到為止
  - **非阻隔式發送**: 送出訊息的行程直接繼續往下執行
  - **阻隔式接收**: 接收端的行程被阻隔(blocking)直到它收到訊息才會繼續往下做
  - **非阻隔式接收**: 不管是不是有收到訊息，接收端的行程都會繼續執行
- **當 send() 與 receive() 皆為阻隔式時，發送端與接收端之間就會是同步的**。

緩衝

- 一個通訊鏈結中，可能設置緩衝區來暫時儲存正在鏈結中傳遞的訊息，一般是用訊息佇列來實作。
- 實作訊息佇列時可使用下列幾種方法：
  - **無緩衝**: 發送端要等接收端接收到資料才能繼續傳下一個訊息
  - **有限緩衝**
  - **無限緩衝**
- 使用有緩衝的訊息佇列時，若行程間需要同步則需另外處理。

例外狀況

- 例外處理，用來處理各種例外狀況的一種**錯誤回復**機制。
- 當下列狀況發生時，系統可能需要進行錯誤的回復：
  - 行程結束
  - 訊息遺失
  - 訊息的錯誤

摘要 (1)

- 行程簡介
  - 行程就是一個執行中的程式。
- 行程的狀態
  - 新建
  - 執行
  - 等待
  - 就緒
  - 終結
- 作業系統中的佇列
  - 就緒佇列
  - I/O 佇列
  - 等待佇列

摘要 (2)

- 作業系統中主要的排程器
  - 長程排程器
  - 短程排程器
- 內文切換
  - 將上一個行程的相關資訊先保存起來，並且把即將要使用 CPU 的行程的相關資訊載入系統中。
- 執行緒的概念
  - 減少行程間切換所造成的額外負擔。

摘要 (3)

- 同一個行程中的執行緒共用
  - 程式區段、資料區段
  - 一些從作業系統取得的資源
- 行程同時存在系統中執行時，可能會是
  - 獨立的行程
  - 合作的行程
- 作業系統提供一些方法讓行程之間能夠溝通
  - 共享記憶體
  - 訊息傳遞系統