Operating Systems
作業系統

SCHEDULER
排程

排程

- 排程概念
  - 行程行為
  - CPU 排程器
  - 排程時機
  - 分派器
  - 排程準則
- 排程方法
- 特殊用途排程
- 排程評估
- 摘要

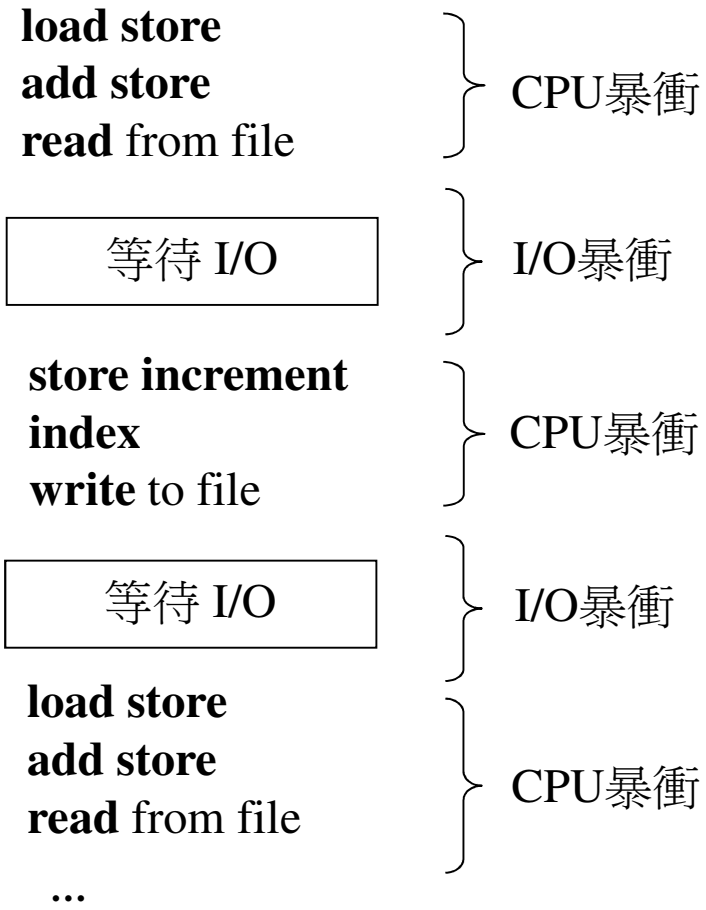
排程概念

- 一個行程在**執行期間**會有很多時間在**等待**（如：等待 I/O 完成）。
- 在單一行程或批次的系統中，當行程在等待時 CPU 是閒置的。
- 在多行程的系統中，若是某一行程變成等待的狀態，該行程會被作業系統從**就緒佇列**中**移除**，然後由排程器在就緒佇列中選出一個最適當的行程，並將 CPU 的使用權交給這個行程。
- 排程就是**將系統資源作更有效的利用**。

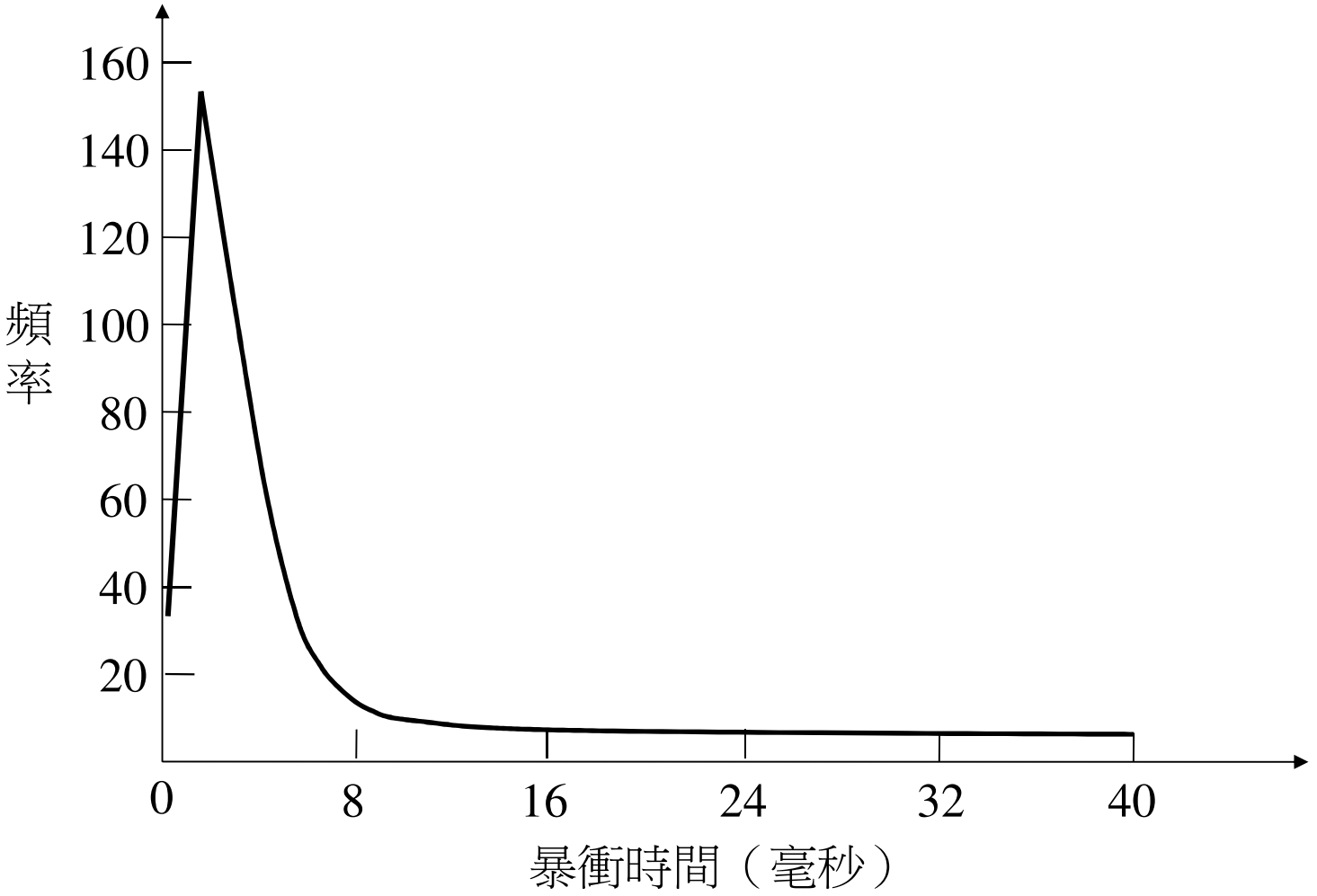
行程行為

- 行程的執行會在兩個狀態間不停的切換
  - CPU 暴衝 (CPU burst)
  - I/O 暴衝 (I/O burst)
- 行程一**開始**都是 **CPU 暴衝**，接著是 **I/O 暴衝**，然後再回到 CPU 暴衝和 I/O 暴衝。在正常的狀況下，行程就在這兩個狀態間一直循環，最後以 CPU 暴衝作為結束。

CPU 暴衝與 I/O 暴衝


load store
add store
read from file
CPU暴衝
等待 I/O
I/O暴衝
store increment
index
write to file
CPU暴衝
等待 I/O
I/O暴衝
load store
add store
read from file
CPU暴衝
...

CPU 暴衝時間統計示意圖



CPU 排程器

- **短程排程器**（或是 CPU 排程器），由**就緒佇列**中選出下一個可以執行的行程。
- 實作一個**就緒佇列**可根據不同的需求使用
  - 先進先出（FIFO)佇列 (First-In First-Out)
  - 優先權佇列 (Priority)
  - 樹 (Tree)
  - 鏈結 (Linked List)
    - 例如：在分時作業系統中，使用鏈結就很適合。

排程時機 (Schedule Occasion)

- 有 4 種行程狀態的變化，需要進行排程。
  - 由**執行**狀態切換到**等待**狀態
  - 由**執行**狀態切換到**結束**狀態
  - 由**執行**狀態切換到**就緒**狀態
  - 由**等待**狀態切換到**就緒**狀態
- 前兩種狀態為行程**主動放棄執行權**，而後 2 種狀態為**被動**的。
- 一個系統中若**只有行程主動放棄執行權**時才會進行重新排程的話，則這個系統的排程方法為**不可搶先的**(non-preemptive)，**反之**，稱為**可搶先**的(preemptive)

分派器 (Dispatcher)

- 當排程器選出下一個行程後，就把工作交給分派器。
- 一個分派器會做下面幾件事：
  - 內文切換
  - 切換回使用者模式（排程是在**管理者**或**核心模式**進行）
  - 重新回到使用者的程式，並且從適當位址重新開始執行
- 由分派器停止一個行程到開始執行另一個行程的這段時間稱為**分派延遲**。

排程準則 (Scheduling Criteria)

- 在選擇一個排程器時有下列幾項準則：
  - **CPU 使用率** (CPU utilization)
  - **產量** (Throughput)
  - **回覆**時間 (Turnaround time)
  - **等待**時間 (Waiting time)
  - **反應**時間 (Response time)
- 根據不同的**系統需求**，可以使用不同的準則來選擇不同的排程器。
- 舉例來說，為了保證使用者能得到最好的服務，我們會盡量**減小最大反應時間**。

The sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the cpu, and doing I/O

The time from the submission of a request until the first response in produced

排程

- 排程概念
- 排程方法
  - 先到先做排程
  - 最短工作優先排程
  - 優先權排程
  - 循環分時排程
  - 多層佇列排程
  - 多層反饋佇列排程
- 特殊用途排程
- 排程評估
- 摘要

排程方法 (Scheduling Algorithms)

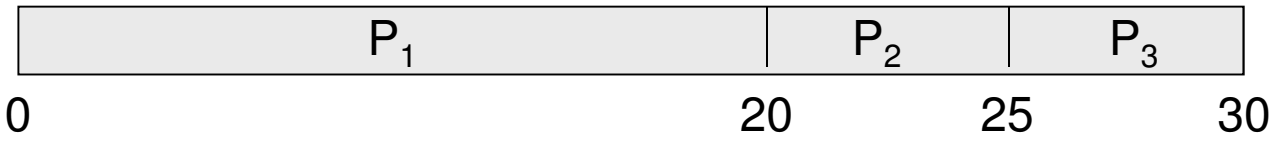
- 一個排程方法決定**就緒佇列**中那一個行程可以使用 CPU 資源。
- 常見的排程方法有：
  - 先到先做排程 (First-Come, First-Served Scheduling)
  - 最短工作優先排程 (Shortest-Job-First Scheduling)
  - 優先權排程 (Priority Scheduling)
  - 循環分時排程 (Round-Robin Scheduling)
  - 多層佇列排程 (Multilevel Queue Scheduling)
  - 多層反饋佇列排程 (Multilevel Feedback Queue Scheduling)

先到先做排程 (First-Come, First Served, FCFS)

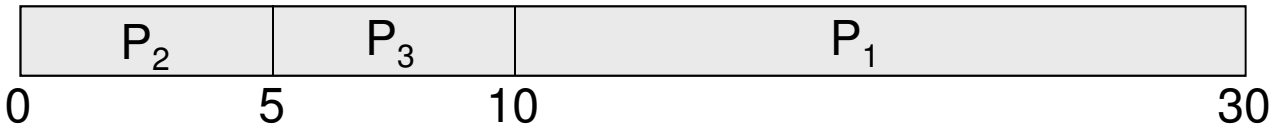
- 先到先做（FCFS）為最簡單的**不可搶先排程法**。
- 根據行程**要求**使用 **CPU** 的順序，來取得 CPU 的使用權。
- 所產生的**平均等待時間**經常都很長。
- 使用先到先做排程法時，**若系統中存在一個 CPU 暴衝時間很長的行程時**，則會產生**護航現象**(Convoy effect)
  - All the other processes wait for the one big process to get off the CPU
    - Lower CPU and device utilization

先到先做排程 (續)

行程	CPU 暴衝時間（毫秒）
P_1	20
P_2	5
P_3	5



平均等待時間：(0 + 20 + 25）/ 3 = 15 毫秒



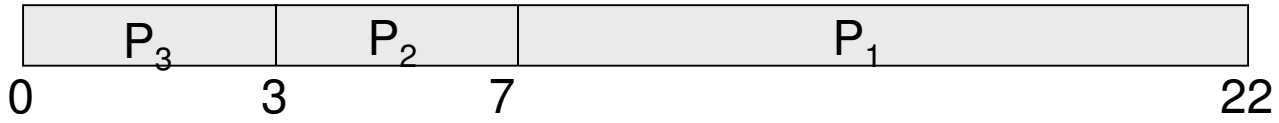
平均等待時間：(10 + 0 + 5）/ 3 = 5 毫秒

最短工作優先排程

- 下一次 CPU 暴衝最短的行程可優先取得 CPU 的使用權。
- 對於**平均等待時間**而言最短工作優先排程（SJF）為**最佳的不可搶先排程法**。
- 若兩個行程下一次的 CPU 暴衝時間**等長**的話，則可以**使用 FCFS 排程方式**來排程。

最短工作優先排程 (續)

行程	CPU 暴衝時間（毫秒）
P_1	15
P_2	4
P_3	3



平均等待時間：（7 + 3 + 0）/ 3 = 3.3 毫秒

若使用 FCFS 排程法，行程到達的順序為 P_1、P_2、P_3
則平均等待時間：（0 + 15 + 19）/ 3 = 11.3 毫秒

FCFS 的平均等待時間大約為 SJF 的 3.4 倍。

最短工作優先排程 (續)

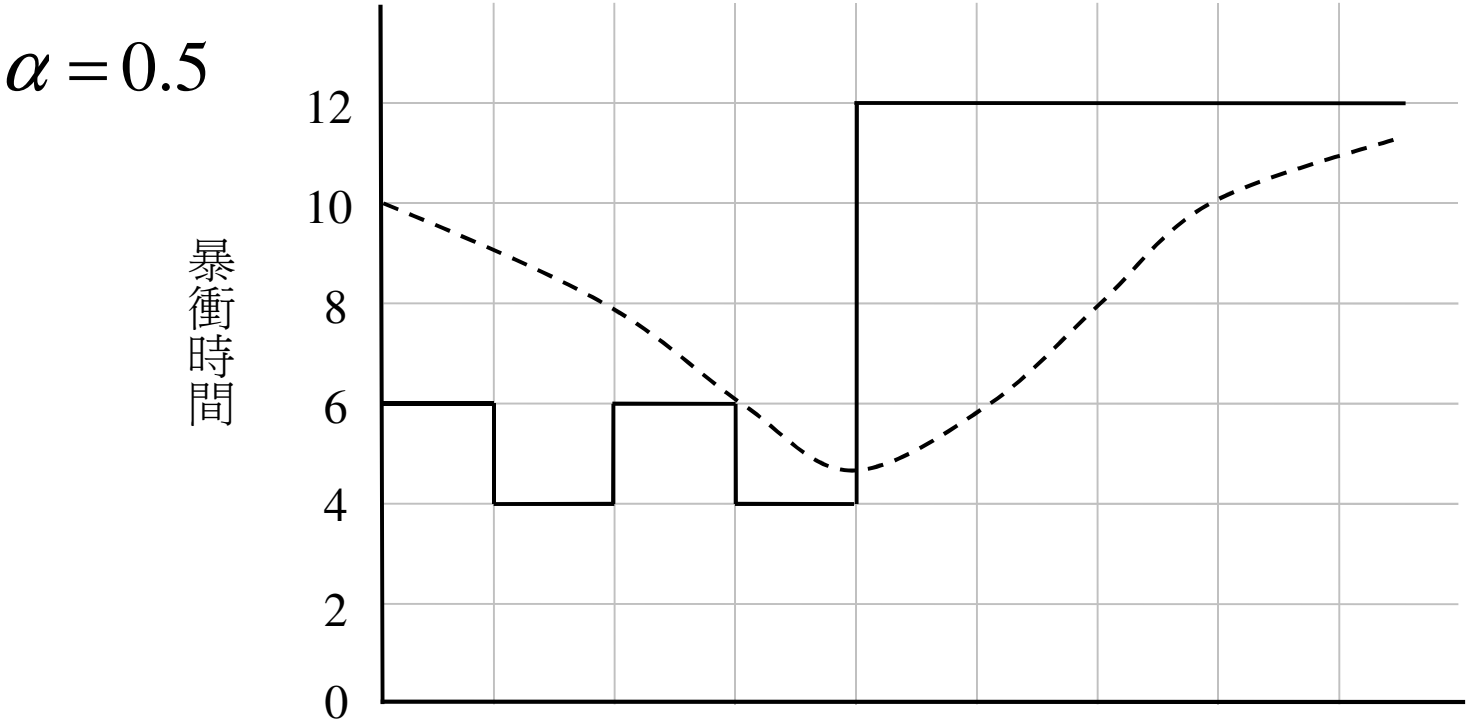
- SJF 在**實作上有困難**，因為很難知道行程下一個 CPU 暴衝時間的確實長度。
- 使用預估的方法來求得近似的值。行程下一個 CPU 暴衝時間的預估值可以設為前幾次 CPU 暴衝時間的幾何平均值。
  - t_n 為第 n 次 CPU暴衝時間的長度
  - T_{n+1} 表示再下一次 CPU 暴衝時間的預估值

$$T_{n+1} = \alpha t_n + (1-\alpha)T_n$$

$\alpha(0 \le \alpha \le 1)$為一常數

下一次 CPU 暴衝時間的預測

$\alpha = 0.5$

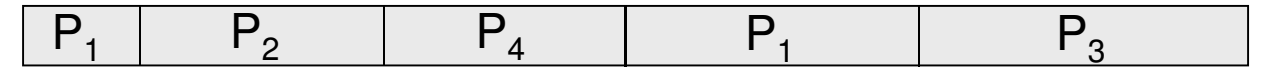


CPU暴衝（t_i）——		6	4	6	4	13	13	13	…
預測（T_i）----	10	8	6	6	5	9	11	12	…

最短工作優先排程 (續)

- SJF **也可以**是**可搶先**的。
- 可搶先的 SJF 排程又稱為**最短剩餘時間優先**的排程法。

行程	CPU 暴衝時間（毫秒）	到達時間
P_1	6	0
P_2	3	1
P_3	7	2
P_4	4	3



0 1 4 8 13 22

平均等待時間：（7 + 0 + 11 + 1）/ 4 = 4.75 毫秒

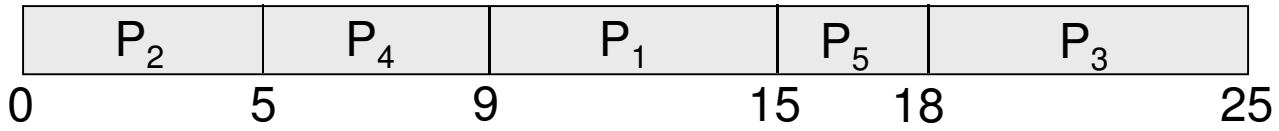
優先權排程

- 排程器依行程的優先權高低來分配 CPU 的使用順序，優先權愈高的行程可優先使用 CPU。
- SJF 也可以視為是一種**優先權排程法**。
- 優先權的定義可以分成兩種類型：
  - **內部** - 使用行程內可以測量的項目，如行程使用的記憶體大小。
  - **外部** - 使用如使用者繳費的多寡等外部資訊。
- 優先權排程可以是**不可搶先**的或**可搶先**的。
- 有可能發生**飢餓**(starvation)的現象
  - 可使用**老化**(aging)來解決

優先權排程 (續)

使用不可搶先的優先權排程
優先權數值愈小代表優先權愈高

行程	CPU 暴衝時間（毫秒）	優先權
P_1	6	2
P_2	5	0
P_3	7	3
P_4	4	1
P_5	3	2



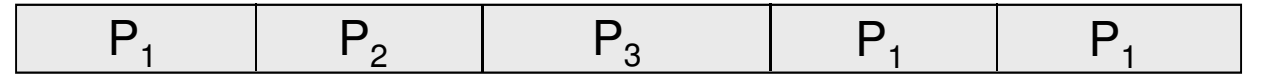
平均等待時間：（9 + 0 + 18 + 5 + 15）/ 5 = 9.4毫秒

循環分時排程 (Round-Robin Scheduling)

- 將時間等切成一小片一小片的**時間切片**，每一個時間切片則為每個行程每次得到 CPU 使用權後可執行的時間。
- 特別為分時系統所設計，為**可搶先**的。

行程	CPU 暴衝時間（毫秒）
P_1	15
P_2	4
P_3	4

(時間切片為 5 毫秒)



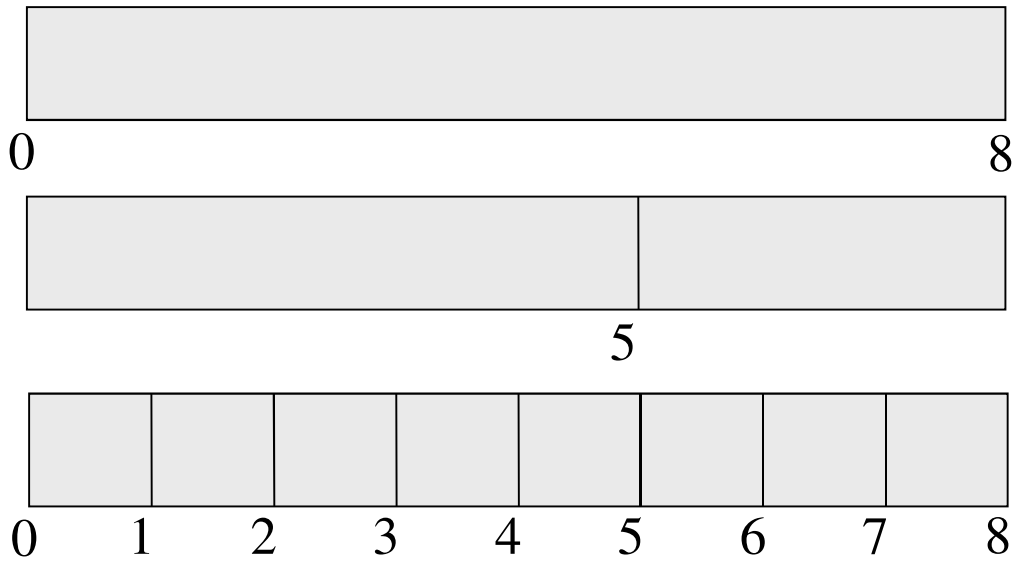
0 5 9 13 18 23

平均等待時間：（8 + 5 + 9）/ 3 = 7.33 毫秒

循環分時排程 (續)

- RR 排程法的**效能**與**時間切片** (time quantum or time slice)的長短關係非常密切
  - 太長 - 如同FCFS 排程法
  - 太短 - 產生**處理器分享**的現象
- 使用 RR 排程法時，必須注意**內文切換**對效能的影響。
- 使用 RR 排程法最重要的一點就是**定義時間切片** 的長短。一般的**經驗法則**是 80% 行程的 CPU 暴衝時間應該要比一個時間切片 要來得短。

時間切片與內文切換



時間切片長度	內文切換次數
10	0
5	1
1	7

多層佇列排程

- 將**行程分類**，相同類型的行程分在同一佇列，而每一佇列都有自己的排程方法。
- 最常見的分類將行程分成
  - **前景**（互動）行程 - RR 排程法
  - **背景**（批次）行程 - FCFS 排程法
- 佇列與佇列之間還有**優先權**的關係，且**佇列之間**還需要另一個**整體排程**方法
  - 可搶先的固定優先權排程法
- 可能會產生**飢餓**的現象。

多層佇列排程法

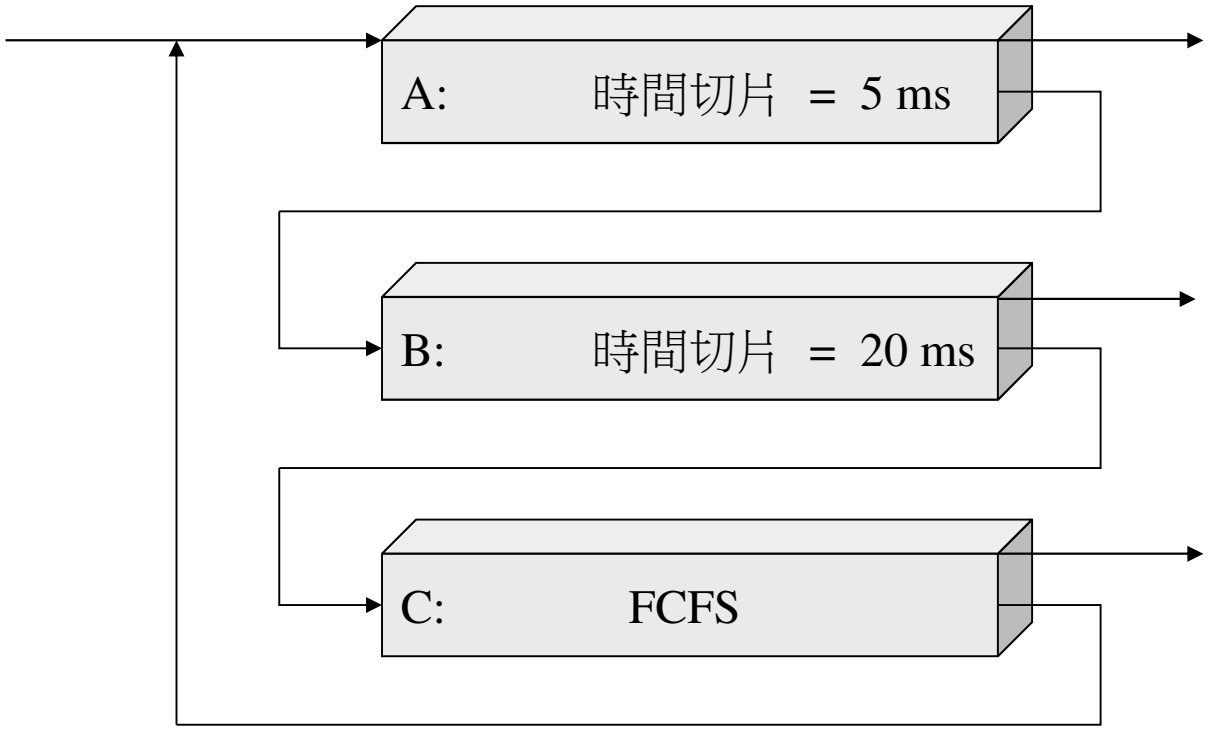
高優先權

低優先權

多層反饋佇列排程

- 多層反饋佇列排程法允許行程在各個佇列間**移動**。
- CPU 暴衝時間愈長的行程就移到優先權比較低佇列中。
- 能**避免飢餓**的現象發生。

多層反饋佇列排程 (續)



排程

- 排程概念
- 排程方法
- 特殊用途排程
  - 多處理器排程
  - 即時排程
- 排程評估
- 摘要

特殊用途排程

- 特殊的系統中，需要使用特別的排程方法才能完成系統的特殊需求，如
  - 多處理器的環境
  - 即時系統

多處理器排程 (Multiple-Processor Scheduling)

- 若系統中的**處理器**皆為**相同架構**時，此系統稱為**同質**，反之，稱為**異質**。
- 在同質系統中，通常將所有**就緒**的行程都放置於**同一個佇列**中，等待著被分派給空閒的處理器執行。
- 共用就緒佇列的架構下形成了兩種排程方式
  - **對稱式多元處理** - 各處理器自行進行排程
  - **非對稱式多元處理** –由某一顆處理器來幫其他處理器進行排程
- 異質系統，較少被使用，目前傾向使用分散式系統取代

即時排程 (Real-Time Scheduling)

- 提高所有工作的**可排程性**及**可預測性**，也就是保證最多工作能在所要求的時間限制下完成。
- 依照系統對時間限制所要求的程度可分為兩類：
  - 硬即時系統
  - 軟即時系統
- **硬即時系統**須**保證關鍵工作在一定的時間內能夠完成**，否則對系統會有**負面**的影響。
  - Ex: 沒有硬碟或虛擬記憶體支援的系統
- **軟即時系統**對時間的要求就沒有那麼嚴苛，萬一工作沒有在一定的時間內完成，它還是有部份價值
  - 軟即時系統的排程必須採用優先權排程法
  - Ex:當按下行動電話按鈕要應接來電時，則必須在按下按鈕時即建立連結。然而，此限制時間並非絕對，亦可有些許的延遲時間

參考資料來源： http://zone.ni.com/devzone/cda/tut/p/id/6459

即時排程 (續)

- 為了讓即時行程能馬上執行，分派延遲必須很短。
- 為了縮短分派延遲，系統呼叫必須是可以搶先的：
  - 在系統呼叫中加入可搶先點。
  - 讓整個核心都是可搶先的
    - 整個核心中的資料結構都需要使用同步的機制來保護。
    - 會產生**優先權倒轉**(Priority inversion)的狀況。

如果高優先權的行程需要存取的核心資料已被其他低優先權的行程所使用，則高優先權的行程必須等待低優先權的行程完成才能繼續執行。

排程

- 排程概念
- 排程方法
- 特殊用途排程
- 排程評估
  - 定性模式
  - 排隊模式
  - 模擬
  - 實作
- 摘要

排程評估 (Scheduling Evaluation)

- 不同的排程方法，適用於不同的系統。
- 有幾種不同的**評估方法**來為系統選擇合適的排程方法：
  - 定性模式
  - 排隊模型
  - 模擬
  - 實作

定性模式 (Deterministic Modeling)

- 定性模式是一種**分析式評估**，使用預先選定的行程組合來評估各種不同的排程方法。
- 例如預先選定一組行程組合，然後分別使用 FCFS、SJF 和 RR 排程法來評估那一種排程法所產生的平均**等待時間**最短。
- 定性模式的評估**快速簡單**，但
  - 需要事先知道很多**系統的資訊**當作輸入。
  - 只適用於**行為比較固定**的系統上。

排隊模式 (Queuing Models)

- 電腦系統可看成是以網路相連的一群伺服器的組合：
  - CPU 是執行**就緒佇列**中行程的伺服器
  - I/O 系統是執行**裝置佇列**中行程的伺服器
- 佇列中行程的**數目隨著時間變化**，不適合以定性模式來評估。
- 若知道新行程到達的速率與舊行程被處理的速率，就可能求出
  - CPU 使用率
  - 佇列平均長度
  - 平均等待時間

排隊模式 (續)

- Little's formula

$$n = \lambda \times W$$

n:為 平均的佇列長度

W: 為行程在佇列中等待的時間

λ: 為新行程平均到達的速率

- Little's formula 所作的排程評估相當有用，因為它的結果對**各種排程方法**都是成立的。
- 系統排程的模型很難以演算法和行程到達的分佈函數完全模擬出來。
- 排隊模型得到的結果經常只是**模擬**的**近似值**。

模擬 (Simulations)

- 透過軟體建立一個**電腦系統模型**，以軟體的資料結構來模擬系統中的主要元件。
- 模擬程式在執行時，會產生很多**資料**與**數據**，這些資訊通常都被收集起來以備日後分析使用。
- 模擬時需要有很多資料來當作模擬程式的輸入，通常使用
  - **亂數產生器**產生行程和 CPU 的**暴衝時間**。
  - **追蹤磁帶**
- 模擬可以得到比較準確的評估結果，但它需要**花費很大的成本**，通常一次模擬都需花上數小時的時間。

實作

- 要完整地評估一個排程方法最終還是需要實作出排程器。
- 實作的**主要困難點**在於所**花費的代價太高**：
  - **撰寫排程器**與**修改作業系統**需花費大量的時間
  - 使用者對作業系統經常改變所產生反應的代價
- 另一個困難點在於**評估的環境**是**變動**的。

摘要 (1)

- 排程的目的是藉由行程在 CPU 之間的切換，增加整體的系統效能。
- 不同的排程方法對最適當的行程會有不同的定義。
- 排程方法可以分成可搶先的或是不可搶先的。
- FCFS 排程是最簡單排程法，但有可能造成 CPU 暴衝短的行程必須等待 CPU 暴衝很長的行程。

摘要 (2)

- 對平均等待時間而言，SJF 是最理想的排程法，不過要實作 SJF 排程法是很困難的。
- 優先權排程法是根據行程優先權的高低來分配 CPU 使用權的順序，高優先權的行程能優先使用 CPU。
- 優先權與 SJF 排程法都可能會造成飢餓現象，使用老化技術可以避免飢餓現象。
- RR 排程法，是特別為分時系統所設計的， 使用 RR 排程法最重要的一點就是定義時間切片的長短。

摘要 (3)

- 多層佇列排程法將行程分類，相同類別的分在同一佇列，每一佇列都有自己的排程法，行程不可以在各佇列間移動。
- 多層反饋佇列排程法類似多層佇列排程法，但允許行程在各個佇列間移動。
- 評估排程方法時通常有幾種方式：
  - 定性模式
  - 排隊模型
  - 模擬
  - 實作