CH05-Process Scheduling

CPU scheduling decisions may take place when a process:

1. 從執行狀態轉變為等待狀態
2. 執行狀態轉變為就緒狀態
3. 等待狀態轉變為就緒狀態
4. 終止

Scheduling under 1 and 4 is nonpreemptive

Scheduling under 2 and 3 is preemptive

Dispatcher分配器

由它把CPU的控制權授予經由短期排班程序選出來的處理程序

1. 轉換處理資料
2. 轉換到使用者模式
3. 跳到使用者程式中的某一適當地點，從新開始執行程式

Dispatch latency：dispatcher去停止一個程序並開始另一程序所花的時間。

Scheduling Criteria演算法的量測準則

* CPU使用率
* 生產量
* 回轉時間(Turnaround Time)：(Completion Time) – (Start Time)
* 等待時間：Waiting in the Ready Queue
* 回應時間：提出要求到第一個回應的時間

Scheduling Algorithms

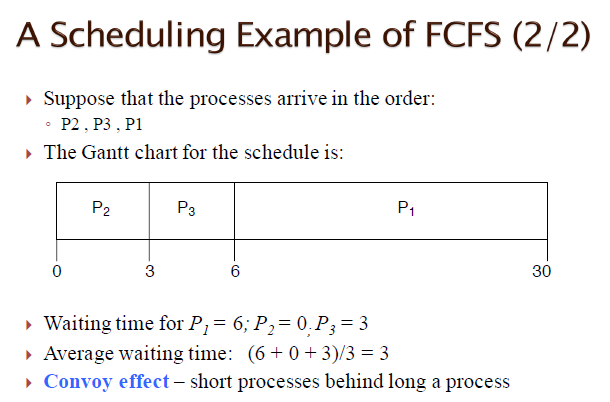
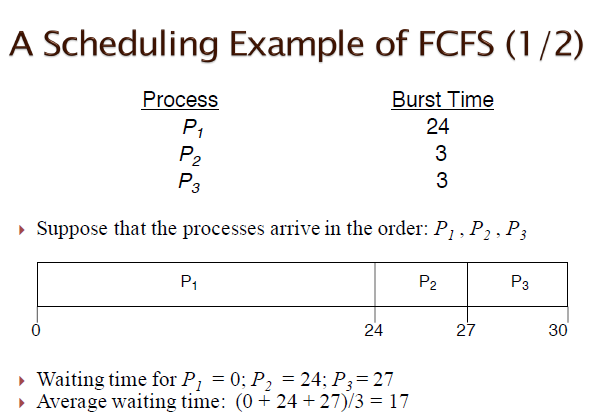
1. First-Come, First-Served Scheduling (FIFO)

按照進入ready queue的順序而決定使用CPU的先後

Properties：

* 不可插隊的(Non-Preemptive)
* CPU可能被占用很久

Ex：

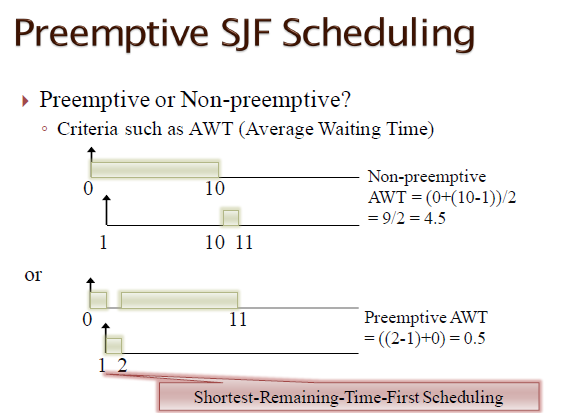


Convoy effect(護衛效應)：很多長短混合的處理程序，都在等一個long time process的處理程序

1. Shortest-Job-First Scheduling (SJF)

* Non-Preemptive SJF在所有程序到達時間皆為0時最佳
* 短期CPU排程還不能實施SJF，因為不能確認下一個CPU動作時間
* 分為Preemptive(SRJF-shortest remaining time job first)和Non-Preemptive

Ex：

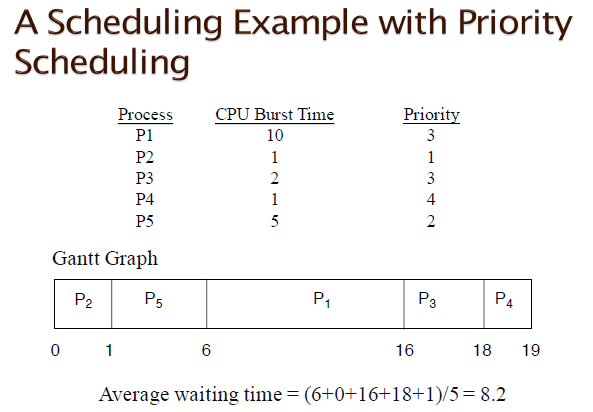


1. Priority Scheduling

* 會將CPU指定給具有最高優先權的程序
* Priority Assignment
  + 內部定義：使用可測量值來計算處理程序的優先權。Ex：開啟檔案數量、平均I/O動作、平均CPU動作
  + 外部定義：屬於作業系統的外部因素決定。

Ex：處理程序重要性

Ex：

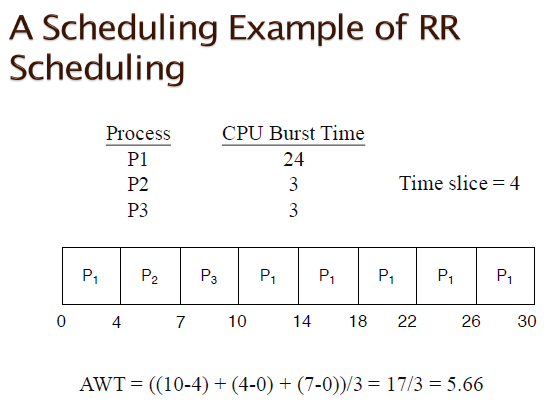


* Issues of priority scheduling
  + 問題：Sarvation(飢餓現象)-優先權低的程序可能永遠不會被執行
  + 解決方法：Aging(老化)-將等候很長的時間的處理程序的優先權逐漸提高
  + A special case：CPU分割越大的，優先權越低

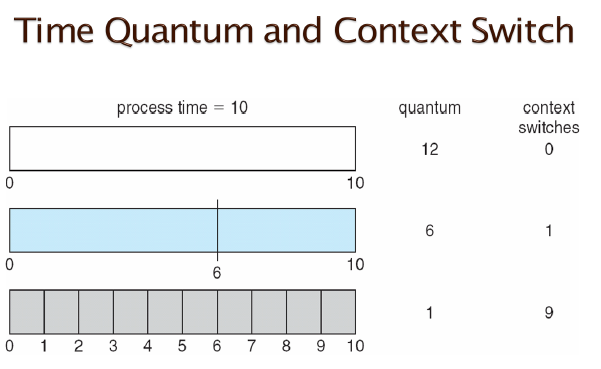
1. Round-Robin Scheduling (RR)

* 每個程序分得一個單位的CPU時間(Time Quantum配額)
* 當程序執行超過一個配額時，則中斷，將程序放入預備佇列

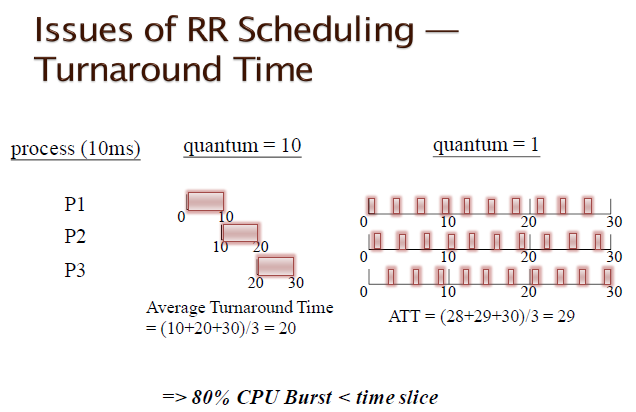
Ex：



* Context switch

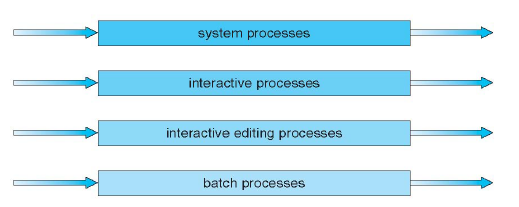


* Isssue og RR scheduling
  + 時間配額太大->FIFO
  + 時間配額太小->context switching 負擔太大



1. Multilevel Queue Scheduling

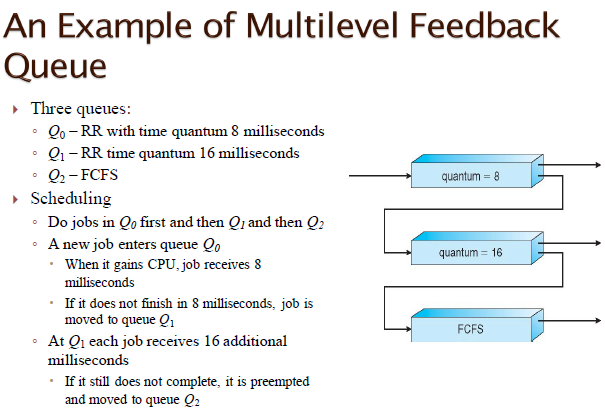
* 將預備佇列分成幾個不同佇列->根據處理程序的某些特性，固定地將處理程序分配到不同佇列



1. Multilevel Feedback Queue Scheduling

* 程序可以在不同佇列間移動->實作aging
* 定義因素
  + 佇列個數
  + 每一個佇列的排班法則
  + 決定什麼時候把程序提升到佇列的方法
  + 決定降低高優先權佇列行程到下層佇列時機的方法
  + 當程序需要服務時，決定該行程進入那一個佇列時機的方法

Ex：



Thread Scheduling

* 為了在CPU上執行，使用者層次執行緒(User threads)必須被排班到相關的核心執行緒(kernel thread)
* Local Scheduling
  + Process-Contention Scope(PCS)行程競爭範圍
  + 執行緒程式庫決定要排班到哪個核心執行緒
* Global Scheduling
  + System-Contention Scope (SCS)系統競爭範圍
  + 為了決定那一個核心執行緒排班到CPU

Multiple-Processor Scheduling

* 多處理器的排班
* Homogeneous System (同質)

處理器的功能性是相同的

* Heterogeneous System(異質)

程序必須在適當的處理器上編譯獲得指令

Homogeneous Processors

* Asymmetric multiprocessing非對稱多元處理

只有一個處理器存取系統資料，減少對資料共享的需要

* Symmetric multiprocessing(SMP)對稱多元處理器

每個處理器自行排班，所有行程可能在一個普通的就緒佇列中，或每一個處理器有他自己的私人就緒佇列

Multiple-Procssor Scheduling-Processor Affinity親和性

* Hard affinity：允許一個行程指定他能夠執行之處理器的子集合
* Soft affinity：
  + 讓一個程序一直在單一處理器上，但有可能在處理器間轉移
  + 非均勻記憶體存取特性的架構->CPU對於主記憶體某部分的存取比其他部分快

Multiple-Procssor Scheduling-Load Balancing負載平衡

* 在SMP系統，讓所有的處理器保持工作量平衡
* Push migration推轉移：特定任務週期性檢查每個處理器的載入狀態，若發現不平衡，用搬移程序將過度附載的處理器平均地分散在入到較不忙的處理器
* Pull migration拉轉移：閒置處理器由忙碌的處理器拉一個等候的任務

Multicore Processors

* 多個處理器核心在同一個實體晶片上
* 問題：Memory Stall記憶體停滯 – 處理器存取記憶體時將花費可觀的時間在等待資料便可以使用

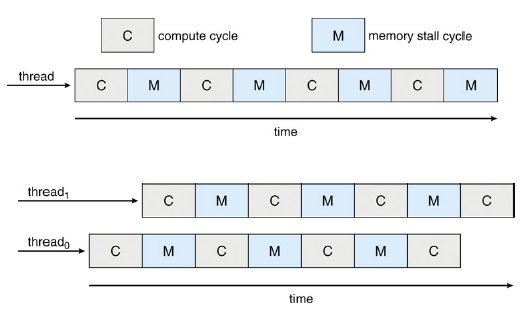
方法：

* + Course-Grained Multithreading粗糙多執行緒

執行緒執行直到長時間停滯發生

* Fine-Grained Multithreading精緻多執行緒

對執行緒間切換有更好的架構



Real-Time Scheduling

* 每個工作都必須在期限內完成
* Soft real-time systems - 確保即時工作能在期限內完成，故對於何時被排班沒有保證，指確保該行程比非迫切形成更優先被考慮
* Hard real-time systems – 即時工作需在期限內完成，在期限過後的服務等同於完全沒有服務
* 影響效能的兩種潛伏期(latency)
  + Interrupt latency 中斷潛伏期 - 中斷到達CPU到開始執行中斷服務常式(ISR)的時間間隔
  + Dispatch latency 分配潛伏期 – 當下程序結束到切換到下個程序的時間間隔

Scheduling Algorithm Evaluation

* A General Procedure

定義選取演算法的標準，一般都是：CPU使用率、反應時間、總產量

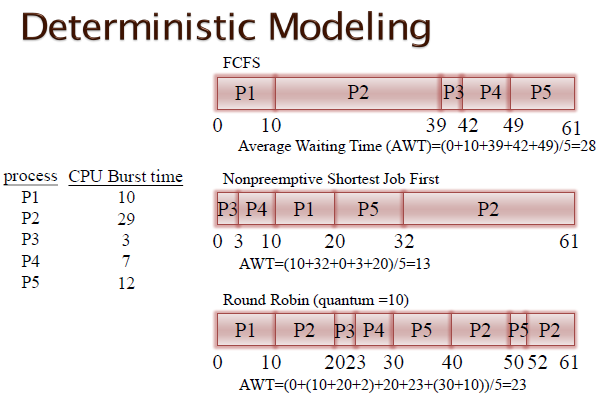
* 演算方法
  + Deterministic modeling
  + Queuing models
  + Simulation
  + Inplementation

Deterministic modeling

* 一種分析式的評估

取一個特殊預定的工作量，並且規定針對該工作量做每種演算的效能

* 特性
  + 簡單、快
  + 以實際的產量讓演算法能互相比較
  + 需要實際的輸入數量，而其結果也只適合這些情況



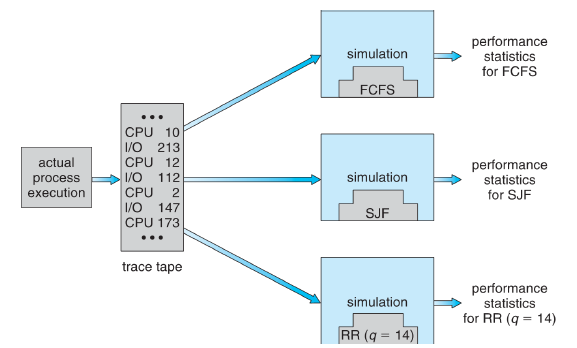
Queuing models

* 動機：執行程序(工作量)都不相同，沒有固定的一組程序來做定量模式
* 需訂出
  + 工作率：
    - 到達速率：給定程序到達系統的時間分布
    - CPU和I/O分割的分布情形
  + 服務率

->由工作率和服務率就可以計算出電腦的使用率、平均佇列長度等等

Simulation

* 動機：為了得到更正確的排班演算法之評估
* 程序：
  + 設計一個電腦系統的模型
  + 驅動模擬的資料
    - 依照機率分布隨機產生數值->不太準確，只表示有多少事件發生，但不表示他們之間的發生次序
    - Trace tapes：模擬實際系統，紀錄實際事件的次序



Implementation

* 動機：得到比模擬更準確的評估
* 程序
  + 撰寫程式碼
  + 放到作業系統中
  + 評估實際工作情形
* 困難
  + 將演算法撰寫成程式碼及修改作業系統
  + 使用者對於一個經常改變的作業系統會產生的反應
  + 演算法的環境會改變