排程

資科系
林偉川

排程概念

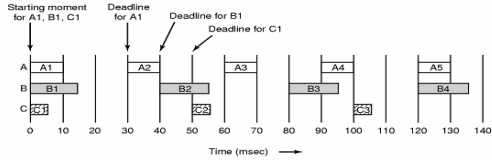
- 一個行程在執行期間會有很多時間在等待(如：等待 I/O 完成)。
- 在單一行程或批次的系統中，當行程在等待時CPU是閒置的。
- 在多行程的系統中，若是某一行程變成等待狀態，該行程會被作業系統從就緒佇列中移除，然後由排程器在就緒佇列中選出一個最適當的行程，並將CPU 的使用權交給這個行程。
- 排程就是將系統資源作更有效的利用。
- OS就是以排程為中心來設計的

2

基本觀念

- 在單一處理器系統裏，不可以同時有多個程式在執行，如果有多個行程，其它的都必須在旁邊等待CPU有空，才能接者重新排班。
- 多元程式規劃系統的主要目的，就是要隨時保有一個行程在執行，藉以提高CPU使用率。

Multimedia Process Scheduling

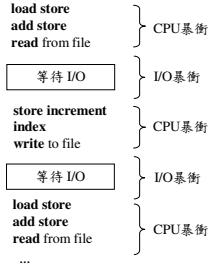


行程行為

- 行程的執行會在兩個狀態間不停的切換
  - CPU暴衝(burst)：行程在CPU執行的期間
  - I/O暴衝(burst)：行程在等待I/O的期間
- 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。
- 但有一些I/O動作，其實是屬於CPU暴衝。例如當一個檔案想要寫入磁碟中，需透過CPU作一些記憶體的搬動，但是發生I/O暴衝時，行程會被阻隔，並將CPU的使用權交出去。

4

CPU暴衝與 I/O 暴衝



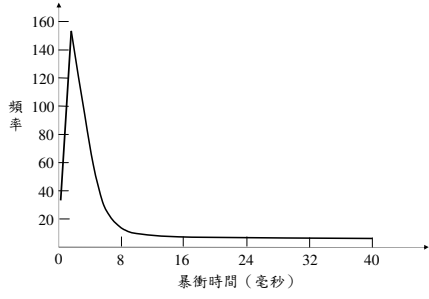
5

CPU 暴衝與 I/O 暴衝

- CPU為主(CPU bounded)的行程通常是由一些比較長的CPU暴衝所組成
- I/O為主的行程則是由很多小的CPU 暴衝所組成
- 判斷一個行程是CPU為主或I/O為主的評估標準，為根據CPU暴衝而不是I/O暴衝，但CPU暴衝的時間都非常短
- 由於I/O為主的行程需要速度較慢的I/O裝置，因此是整個系統效能的瓶頸

6

CPU 暴衝時間統計示意圖



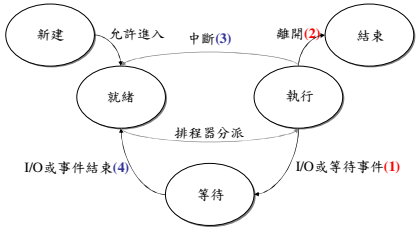
7

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
  - 優先權佇列
  - 樹(二元樹、AVL樹、紅黑樹)
  - 鏈結
    - 例如：在分時作業系統中，使用鏈結就很適合。因為要做內文切換時，排程器很容易透過鏈結，找到下一個準備要執行的行程

8

行程狀態圖



9

排程時機

- 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(發生I/O要求) 1
  - 由執行狀態切換到結束狀態 2
  - 由執行狀態切換到就緒狀態(發生中斷) 3
  - 由等待狀態切換到就緒狀態(完成I/O或是等待計時器時間到) 4
- 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。

10

排程時機

- 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(non-preemptive)的(1,2)，反之，稱為可搶先(preemptive)的(3,4)！
- 在不可搶先的排程方式下，每當行程取得CPU資源後，除非行程自行將CPU釋放出去(進行I/O要求)，否則其他的行程無法取得CPU的使用權。(95tpu)

11

排程時機

- 在可搶先的排程方式，設計OS核心要特別注意同步問題，當核心在處理系統呼叫時很可能要修改一些重要的資料結構，如果這個時候被搶先了，且搶先的行程又修改了相同的核心資料結構，這樣很可能造成系統當機。最簡單處理方式，就是當系統進入系統呼叫後，就將系統中斷關閉，就是當處理系統呼叫時，不會被任何行程搶先，結束系統呼叫時，再將中斷開啟，以確保核心資料結構的一致性，但是即時系統則不適用此種方法。

12

分派器

- 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

13

分派器

- 當每次行程切換動作產生時，分派程式都會執行一次，故其效率很重要。
- 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。

14

排程準則

- 不同排程方法有不同的特性，故必須針對不同的系統需求，來選擇合適的排程方法
- 在選擇一個排程器時有下列幾項準則：
  - CPU使用率：愈高愈好，愈忙愈好
  - 產量(throughput)：單位時間內系統完成的行程個數
  - 回覆時間(turnaround time)：針對單一行程來進行觀察時，要知道其執行開始到結束總共花多少時間，包括等待載入記憶體的時間、在就緒佇列等待時間、在CPU執行的時間、和處理I/O所花的時間，通常回覆時間受限於輸出裝置的速度

15

排程準則

  - 等待時間(waiting time)：排程演算法不會影響行程花在I/O與CPU上的時間，只會影響一個行程在就緒佇列中等待的時間，為一個行程從開始執行到結束這段時間內，在就緒佇列中等待時間的總和
  - 反應時間(response time)：交談式系統中，反應時間才是比較合適的測量準則，也就是計算當使用者送出一個要求，需經多少時間才會產生第一個反應，指的是開始有反應的時間

16

排程準則

- 一般希望OS有最大的CPU使用率和產量，最小的回覆、等待及反應時間，大部分情形，都會平均地做最佳化，為保證使用者能得到最好的服務，OS會儘量減少最大反應時間，對交談式系統(分時作業系統)，最小化反應時間的差異比最小化平均反應時間來得重要，如果反應時間可以預測，會比平均反應時間很快，但是變化很大的系統(快慢不一)要來得更令人滿意。

17

排程方法

- 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。
- 一個真實的系統，一個行程從開始執行到結束的這段期間，可能由數百個CPU暴衝和I/O暴衝交互組成，在加上如果系統中有數十個行程交互執行，其複雜程度難以說明與分析，故假設每個行程都只由一段CPU暴衝所組成，而比較其平均等待時間及回覆時間
- (Linux cron V.S. Windows排程)

18

排程方法

- 常見的排程方法有：
  - 先到先做排程(FCFS)
  - 最短工作優先排程(SJF)
  - 優先權排程(Priority)
  - 循環分時排程(Round-Robin)
  - 多層佇列排程(Multilevel Queue)
  - 多層反饋佇列排程(Multilevel Feedback Queue)
- (92tpu、94tpu、96tku、92tku、97tku、93ncu、96nttu、95nttu、91nttu)

19

先到先做排程

- 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，先發出要求的行程可以先取得CPU 的使用權。
- 實作時需要一個FIFO佇列，當行程進入就緒佇列時，行程的PCB會被連結至佇列的末端(rear)，當CPU閒置時，排程器只需將佇列中第一個行程取出(front)，在把CPU的使用權交給它

20

計算題項目觀念

- 回覆時間(turnaround time)：針對單一行程來進行觀察時，要知道其執行開始到結束總共花多少時間，包括等待載入記憶體的時間、在就緒佇列等待時間、在CPU執行的時間、和處理I/O所花的時間，通常回覆時間受限於輸出裝置的速度
- 等待時間(waiting time)：排程演算法不會影響行程花在I/O與CPU上的時間，只會影響一個行程在就緒佇列中等待的時間，為一個行程從開始執行到結束這段時間內，在就緒佇列中等待時間的總和

21

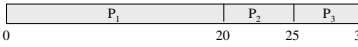
先到先做排程

- 所產生的平均等待時間經常都很長。
- P1、P2和P3 到達時間皆為0，到達次序為P1、P2和P3(到達次序為P2、P3和P1)，CPU暴衝時間如下：

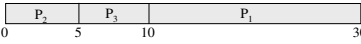
行程	CPU暴衝時間(毫秒)
P1	20
P2	5
P3	5

22

先到先做排程



到達次序為P1、P2和P3，平均等待時間：(0 + 20 + 25) / 3 = 15 毫秒
平均回覆時間：(20 + 25 + 30) / 3 = 25 毫秒



到達次序為P2 、 P3和 P1，平均等待時間：(10 + 0 + 5) / 3 = 5 毫秒
平均回覆時間：(5 + 10 + 30) / 3 = 15 毫秒

23

先到先做排程

- 顯示FCFS排程法並不穩定，FCFS的平均等待時間在各種排程法中並非最小，且當行程中存在CPU暴衝時間很長的行程，平均等待時間就變得很長。但FCFS很容易實作
- 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，其他行程必須等待此行程結束的現象，則會產生護航現象(Convey effect)。
- FCFS排程法非常不適合在分時系統中使用，因其為不可搶先的方式，會造成使用者無法在預定時間內取得CPU使用權

24

先到先做排程

- 在很多行程中只有一個CPU為主的行程，其CPU暴衝為1秒，其他都是I/O為主的行程，每個I/O為主的行程CPU暴衝都很短，但須作100次磁碟讀取，當CPU為主的行程取得CPU使用權後，會持續使用1秒，此時其他I/O為主的行程就必須等待，直到CPU為主的行程將CPU的使用權釋出，如此I/O裝置使用率非常低。當CPU為主的行程結束CPU暴衝釋出CPU使用權進入I/O佇列，此時其他I/O為主的行程也很快結束CPU暴衝也進入I/O佇列，此時CPU使用率又非常低➔護航現象

25

先到先做排程練習

- P1、P2和P3 到達時間皆為0，到達次序為P1、P2和P3，或是到達次序為P3、 P2和 P1，或是到達次序為P2 、 P1和 P3，CPU暴衝時間如下：

行程	CPU 暴衝時間（毫秒）
P1	24
P2	3
P3	3

26

最短工作優先排程

- 根據行程下一次 CPU 暴衝最短的行程可優先取得 CPU 的使用權。
- 若兩個行程下一次的 CPU 暴衝時間相等，則可以使用 FCFS 排程方式來排程。
- 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。
- 藉著將CPU暴衝時間較長的行程延後執行，其他CPU暴衝時間較短的行程就能減少等待時間，因此平均等待時間也跟著下降

27

最短工作優先排程

- P1、P2和P3 到達時間皆為0，到達次序為P1、P2和P3 ，CPU暴衝時間如下：

行程	CPU 暴衝時間（毫秒）
P1	15
P2	4
P3	3

28

最短工作優先排程

使用 SJF 排程法
平均等待時間：(7 + 3 + 0) / 3 = 3.3 毫秒
平均回覆時間：(22 + 7 + 3) / 3 = 11 毫秒

若使用 FCFS 排程法，行程到達的順序為 P1、P2、P3
則平均等待時間：(0 + 15 + 19) / 3 = 11.3 毫秒
平均回覆時間：(15 + 19 + 22) / 3 = 19 毫秒

FCFS 的平均等待時間大約為 SJF 的 3.4 倍。

29

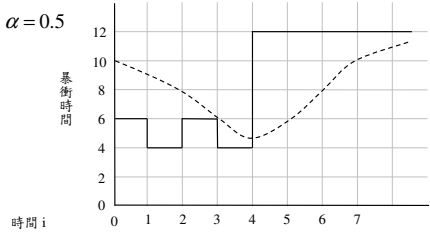
最短工作優先排程

- 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。
- 只能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
  - 若α=0，則$T_{n+1}=T_n$，若α=1，則$T_{n+1}=t_n$
  - 若α=0.5表示最近CPU暴衝時間，與過去CPU的暴衝時間所估的比重相等

$$T_{n+1} = \alpha\, t_n + (1-\alpha) T_n \quad \alpha(0 \le \alpha \le 1)\text{爲一常數}$$

30

下一次 CPU 暴衝時間的預測



31

最短工作優先排程

- P1、P2、P3和P4 到達時間皆為0，到達次序為P1、P2、P3和P4 ，CPU暴衝時間如下：

行程	CPU 暴衝時間(毫秒)
P1	6
P2	8
P3	7
P4	3

32

最短工作優先排程

- P1、P2、P3和P4 到達時間皆為0，到達次序為P1、P2、P3和P4 ，CPU暴衝時間如下：

行程	CPU 暴衝時間(毫秒)
P1	8
P2	4
P3	9
P4	5

33

最短工作優先排程

- SJF 也可以是可搶先的。有新行程進入就緒佇列，且其下一次的CPU暴衝時間比目前行程剩下的CPU暴衝時間更短，這時正在執行的行程會被搶先。
- 可搶先的 SJF 排程又稱為最短剩餘時間優先的排程法。

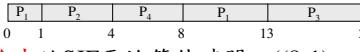
34

最短工作優先排程

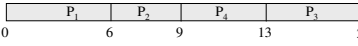
行程	CPU 暴衝時間（毫秒）	到達時間
P1	6	0
P2	3	1
P3	7	2
P4	4	3

35

最短工作優先排程



- 可搶先的SJF平均等待時間：((8-1) + (1-1) + (13-2) + (4-3)) / 4 = 4.75 毫秒，平均回覆時間：((13 + (4-1) + (20-2) + (8-3)) / 4 = 9.75 毫秒



- 不可搶先的的SJF平均等待時間：(0-0 + (6-1) + + (13-2) + (9-3)) / 4 = 5.5 毫秒，平均回覆時間：((6-0)+(9-1)+(20-2)+(13-3)) / 4 = 10.5 毫秒

36

最短工作優先排程

- P1、P2、P3和P4，CPU暴衝時間如下：

行程	CPU 暴衝時間(毫秒)	到達時間
P1	6	0
P2	8	1
P3	7	2
P4	3	3

37

最短工作優先排程

- P1、P2、P3和P4，CPU暴衝時間如下：

行程	CPU 暴衝時間(毫秒)	到達時間
P1	8	0
P2	4	1
P3	9	2
P4	5	3

38

優先權排程

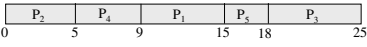
- 排程器依行程的優先權高低來分配 CPU 的使用順序，優先權愈高的行程可優先使用 CPU。當優先權相同時，可使用FCFS排程來決定執行的順序
- SJF 也可以視為是一種優先權排程法。行程的優先權由行程下一次CPU暴衝時間的長短來決定，CPU暴衝愈短的行程優先權愈高
- 優先權數值愈小代表優先權愈高(0 - 7)
- 優先權排程可以是不可搶先的或可搶先的。

39

優先權排程

5個行程到達時間皆為0，使用不可搶先的優先權排程

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，平均回覆時間：((15 + 5 + 25 + 9 + 18)) / 5 = 14.4毫秒

40

優先權排程

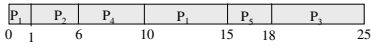
- 優先權的定義可以分成兩種類型：
  - 內部：使用行程內可以測量的項目，來計算行程的優先權，如行程使用的記憶體大小、開啟檔案個數或是平均的CPU暴衝時間➔SJF排程法
  - 外部：使用如使用者繳費的多寡或是行程的重要性等外部資訊，來分配行程的優先權。
- 當使用可搶先的優先權排程法時，若新行程的優先權比現行行程高，現行行程將會被搶先。若使用不可搶先的優先權排程法，則只把新行程放到就緒佇列的最前端

41

優先權排程

5個行程，使用可搶先的優先權排程(ts=3)

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



平均等待時間：((10-1) + 0 + (18-2) + (6-3) + (15-4)) / 5 = 7.8毫秒，平均回覆時間：(15 + (6-1) + (25-2) + (10-3) + (18-4)) / 5 = 12.8毫秒

42

優先權排程

5個行程，使用可搶先、不可搶先的優先權排程(92tku)

行程	CPU 暴衝時間（毫秒）	優先權	到達時間
P_1	10	2	0
P_2	1	0	1
P_3	2	3	2
P_4	1	4	3
P_5	5	1	4

平均等待時間：??毫秒，平均回覆時間：??毫秒

43

優先權排程

- 有可能發生飢餓(starvation)的現象：低優先權的行程有可能永遠不會被執行，在一個負載很高的系統中，若存在一個優先權很低的行程，且執行中不斷有高優先權的行程進入系統，此行程最後可能會有兩種結果：當系統負載減輕時(凌晨三點到五點之間)可能會輪到這個低優先權的行程執行；另一可能則是這個行程永遠不會被執行或是被廢棄(10年未執行的工作)。
- 可用老化(**aging**)來解決：逐漸提高停留在系統中已經經過一段長時間的行程之優先權。

44

循環分時排程(RR)

- 特別為分時系統所設計，為可搶先的排程。
- 將時間等切成一小片一小片的時間切片(time slice/quantum)，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。

45

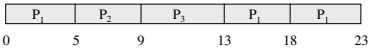
循環分時排程(RR)

- 當時間用完時，排程器就會由就緒佇列中取得下一個行程做內文切換，可以用FIFO佇列來實作就緒佇列，當新行程進入就緒佇列時，會被放在佇列最末端，而排程器會由佇列的最前端取出行程，並將計時器設置成在一個時間切片後產生中斷來重新排程，最後將CPU的使用權遞交給取出的行程。

46

循環分時排程

行程	CPU 暴衝時間（毫秒）	
P1	15	
P2	4	
P3	4	(時間切片為 5 毫秒)



平均等待時間：((13-5) + 5 + 9) / 3 = 7.33 毫秒，
平均回覆時間：((23 + 9 + 13)) / 3 = 15毫秒

47

循環分時排程

行程	CPU 暴衝時間（毫秒）	
P1	24	
P2	3	
P3	3	(時間切片為4毫秒)

平均等待時間：?? 毫秒，平均回覆時間：??毫秒

48

循環分時排程(RR)

- 使用RR排程法會出現兩種狀況：一是行程的CPU暴衝時間小於一個時間切片，另一種則是大於一個時間切片
- 行程的CPU暴衝時間小於一個時間切片則行程自行交出CPU使用權，排程器由就緒佇列中取出下一個行程
- 行程的CPU暴衝時間大於一個時間切片則計時器會在一個時間切片結束時產生中斷，告知OS重新排程，這時正在執行的行程會被搶先，並放在就緒佇列的最末端，排程器就從就緒佇列選出下一個行程

49

循環分時排程

- 使用RR排程法時，每一個循環中沒有任何行程可以執行超過一個時間切片，如果行程的CPU暴衝超過一個時間切片，則行程會被搶先，並放於就緒佇列的最末端。
- 假設就緒佇列有n個行程，且時間切片為t，則一個循環中每個行程的等待時間最多不會超過(n-1)*t
- 有3個行程且時間切片為10毫秒，則在30毫秒內每個行程可以分到10毫秒的執行時間，等待時間最多不會超過20毫秒

50

循環分時排程

- RR 排程法的效能與時間切片的長短關係非常密切(96nttu, 96tpu, 97tku)
  - 太長 - 如同FCFS 排程法
  - 太短 - 產生處理器分享的現象，若系統中有n個行程在1個處理器執行，每個行程如同在一個速度為原處理器1/n倍的處理器上執行一般

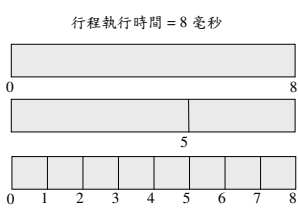
51

循環分時排程

- 使用 RR 排程法時，必須注意內文切換對效能的影響。若一個行程需要執行8毫秒，而時間切片為10毫秒，則在一個時間切片內不會有任何內文切換的額外負擔。如果時間切片為5毫秒，會有一次內文切換的額外負擔，若時間切片為1毫秒，則會有7倍的內文切換的額外負擔
- 使用 RR 排程法最重要的一點就是定義時間切片的長短。時間切片太短則內文切換太過頻繁，太長則得到如FCFS排程法的結果，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。

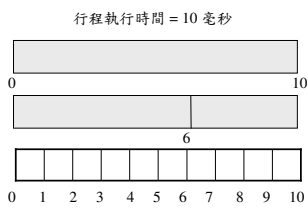
52

時間切片與內文切換



53

時間切片與內文切換



54

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

55

多層佇列排程

- 佇列與佇列之間還有優先權的關係，前景行程的優先權高於背景行程
- 4類行程優先權依序為：系統、伺服精靈、批次、使用者行程，每一佇列都有自己的排程方法
- 前景行程(系統、伺服精靈)和背景行程(批次、使用者行程)佔不同的佇列
- 前景佇列是用循環方式排班(RR)，背景佇列是用FCFS方式排班

56

多層佇列排程

- 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，可保證高優先權的佇列能被優先處理，亦即當還有系統及伺服精靈行程時，批次行程就不能執行，當批次或使用者行程尚在執行，若有系統或伺服精靈行程進入就緒佇列時，現行行程就會被搶先

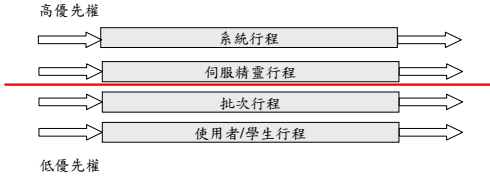
57

多層佇列排程

- RR排程法類似，優先權較高的佇列可得到較長的時間切片，優先權較低的佇列則得到較短的時間切片，前景佇列分到時間切片為8毫秒，而背景佇列的時間切片為2毫秒。前景佇列有80%的CPU時間進行RR排程，背景佇列有20%的CPU時間進行FCFS排程，前景佇列或許只要短的時間切片，但須快速的反應時間。但如此可能會產生飢餓的現象。

58

多層佇列排程法



59

多層反饋佇列排程

- 多層佇列排程法一旦行程分類屬於某一佇列，就不能改變也不會於各個佇列移動，雖減少排程額外負擔，但較沒有彈性，並可能發生飢餓現象

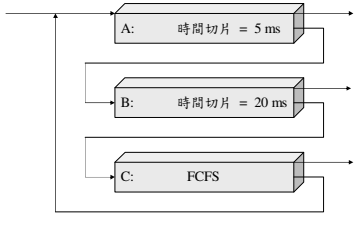
60

多層反饋佇列排程(97nttu)

- 多層反饋佇列排程法允許行程在各個佇列間移動。
- 根據CPU 暴衝時間愈長的行程就移到優先權比較低佇列中，使一些CPU暴衝較短或需要較短反應時間的行程先執行，以降低平均等待時間，行程在低優先權的佇列中等待太久，應用老化的方法，則將行程移至優先權較高的佇列，如此能避免飢餓的現象發生。

61

多層反饋佇列排程



62

多層反饋佇列排程

- 佇列A和B都使用RR排程法，當行程進入就緒佇列會被先放於佇列A，若行程未於5毫秒內結束，則將被移至佇列B。如果佇列A是空的，佇列B中的行程就可以被執行，若行程未於20毫秒內結束，則行程會被移至佇列C。在佇列C中的行程必須等佇列A與B都空了，才能被執行。
- 多層反饋佇列排程很有彈性且複雜，各佇列的時間切片應設為多少、何時應升高行程的優先權等參數設定，皆會影響其排程方法及系統效率

63

多層反饋佇列排程(ts=3、10)

行程	CPU 暴衝時間(毫秒)	到達時間
P1	28	0
P2	19	1
P3	6	2
P4	3	3
P5	5	4

64

多層反饋佇列排程(ts=10、15)

行程	CPU 暴衝時間(毫秒)	到達時間
P1	35	1
P2	21	2
P3	26	0
P4	30	4
P5	15	3

65

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

66

多處理器排程

- 在多處理器中如何進行行程的排程？
- 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。(一片主機版可插多顆CPU)
- 在同質系統中，一個I/O裝置只連接到某個處理器私有的匯流排，當行程需要使用此I/O裝置時，排程器必須將行程交由該處理器執行，因為其他處理器無法存取此I/O裝置

67

多處理器排程

- 每個處理器可擁有自己的就緒佇列，當處理器空閒時就由自己的就緒佇列中選出下一個行程，但可能當某個處理器的就緒佇列空了，使得該處理器閒置，而另一個處理器卻仍處於很忙碌的狀態，所以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。
- 異質系統的使用率較少，通常使用分散式系統來取代

68

多處理器排程

- 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(SMP) - 處理器的行為都是對等的，所以各處理器自行進行排程，當處理器空閒時就到就緒佇列中挑選一個行程來執行，但會發生同步的問題，因可能發生多個處理器同時選出同一個行程的情況，排程器需使用同步機制來確保此狀況不會發生
  - 非對稱式多元處理(NMP) – 由某一顆處理器來幫其他處理器進行排程，為主從架構，此特定的處理器為主伺服器，不僅做排程，也處理I/O和其他系統的反應，其他就只單純負責執行程式

69

處理器親和性

- 大部份對稱式多元處理系統(SMP)試著避免行程由一個處理器轉移到另一個處理器，在同一處理器上保持一個行程，這就是處理器親和性(Processor affinity)，表示行程對並行執行的處理器有親和性。
- **Processor affinity** is a modification of the native central queue scheduling algorithm. Each task in the queue has a tag indicating its preferred processor

70

處理器親和性

- 處理器親和性有幾種型式。當作業系統企圖保持一個行程在相同處理器上執行的策略，這種情況稱為軟性親和性(soft affinity)，它可能讓行程在處理器間轉移。有些系統(如Linux)提供支援硬性親和性 (hard affinity)的系統呼叫，因此允許指定行程不能轉移到其它處理器。

71

多處理器排程

- 非對稱式多元處理架構比對稱式架構要簡單得多，因同一時間只有一個處理器存取系統的資料結構，但非對稱比較沒有效率，因為I/O為主的行程，可能造成主伺服器的瓶頸
- 在對稱式系統中，負載平衡(load balancing)企圖保持工作均勻的分散在所有處理器。通常負載平衡必須在每個處理器有自己合格的行程私人佇列之系統來執行，大部份現代作業系統支援對稱式系統，但負載平衡和親和性處理器的利益是相衝突的

72

負載平衡有兩個方法

- 推轉移(push)中特定週期性檢查每個處理器負載，若發現不平衡，則將過度負載的處理器平均將行程分散到閒置或較不忙的處理器
- 拉轉移(pull)當閒置處理器由忙碌的處理器拉一個等候的行程，推與拉轉移並不是互斥而是負載平衡中平行地執行(FreeBSD OS)

73

對稱多執行緒

- SMP系統允許一些執行緒並行地在提供多個實體處理器上執行。提供多個邏輯處理器而非實體處理器是一個替代的策略。這種策略稱為對稱多執行緒 (或SMT)
- 在英特爾晶片也稱為超執行緒技術(hyper-threading)。

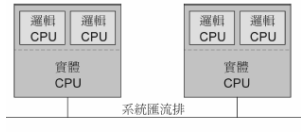


圖 5.8 基本的 SMT 架構

74

競爭範圍

- 在使用者層次和核心層次的執行緒間有一項差別，是在於如何被排班。在製作多對一和多對多模式的系統上，執行緒庫(thread library)排班，使用者層次的執行緒在可取得的較輕省的處理器上執行，這種技巧稱為行程競爭範圍(Process-contention scope, PCS)，因為CPU的競爭發生在屬於相同行程的執行緒。
- 當執行緒庫排班，使用者執行緒到可用的較輕省的處理器上時，並不是指執行緒正在某一個CPU上執行，這必須要作業系統排班核心執行緒在實體的CPU上執行。

75

Sun Solaris 排班

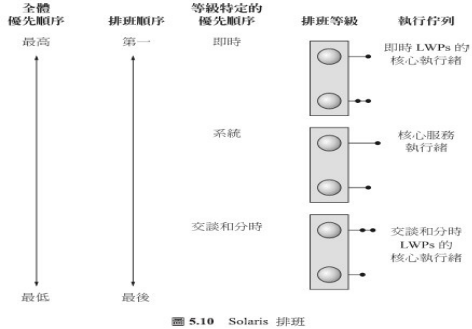


圖 5.10 Solaris 排班

Windows XP 排班

	realtime	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

圖 5.12 Windows XP 的優先權

Linux 排班

圖 5.13 優先順序與時間片段的關係

圖 5.14 依據優先權的任務索引串列

即時排程

- 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，並預測工作排程的結果，而不是越快完成單一工作越好。
- 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

79

即時排程

- 硬即時系統須保證關鍵工作在一定的時間內能夠完，否則對系統會有負面的影響。 當一個行程進入系統時，會將完成工作所需要的執行時間或I/O時間通知排程器，若排程器允許工作執行，則保證這個工作會準時完成。若排程器拒絕此工作，即表示系統無法滿足工作的需求，這就是預約資源。且其排程器需知道系統呼叫的最大執行時間，如此排程器才能保證工作在一定時間之內完成，此系統通常安裝在一些特殊的系統上，例如：沒有硬碟或虛擬記憶體支援的系統

80

即時排程

- 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。
- 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。其主要要求為重要行程的優先權需高於其他行程，此種排程一定是優先權排程法，為了讓即時行程能馬上執行，分派延遲必須很短，但因OS中內文切換必須等到系統呼叫完成或I/O等待時才能進行，若某些系統呼叫相當複雜或I/O裝置速度相當緩慢，分派延遲就會很長

81

即時排程

- 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。若執行到可搶先點且檢查到有更高優先權的行程需要執行，則該行程搶先現行行程，並進行內文切換，但此可搶先點必須在核心資料結構不會被修改的地方。

82

即時排程

– 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。如果高優先權行程需要存取核心資料已被低優先權的行程使用，則高優先權行程必須等待直到低優先權行程完成才能繼續執行，暫時違反優先權排程中高優先權行程必須先執行的原則，這會產生優先權倒轉的狀況。

83

即時排程

- 優先權倒轉可能造成高優先權無法在要求的時間內完成工作，優先權繼承協定就是為了減少優先權倒轉的影響。佔用某資源的低優先權的行程在高優先權行程要使用該資源時，先繼承高優先權行程的優先權，直到繼承的行程不再使用此資源後，優先權才還原成原來的值，以確保高優先權行程只被低優先權行程阻隔一次，不會因低優先權行程被其他中優先權行程搶先，造成高優先權行程不確定地延遲更久

84

排程評估

- 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

85

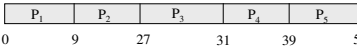
定性模式

- 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。

86

定性模擬 - FCFS

行程	CPU 暴衝時間（毫秒）
P1	9
P2	18
P3	4
P4	8
P5	13

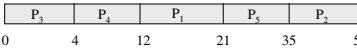


平均等待時間：(0+9+27+31+39) / 5 = 21.2 毫秒，平均回覆時間：(9+27+31+39+52) / 5

87

定性模擬 - SJF

行程	CPU 暴衝時間（毫秒）
P1	9
P2	18
P3	4
P4	8
P5	13

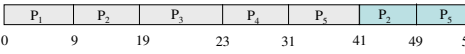


平均等待時間：(12+34+0+4+21) / 5 = 14.2 毫秒，平均回覆時間：(21+52+4+12+35) / 5

88

定性模擬 – RR(時間切片10毫秒)

行程	CPU 暴衝時間（毫秒）
P1	9
P2	18
P3	4
P4	8
P5	13



0 9 19 23 31 41 49 52

平均等待時間：(0+(9+(41-19))+19+23+(31+(49-41))) / 5 = 22.4 毫秒，平均回覆時間：(9+49+23+31+52)/5

89

定性模擬–FCFS、SJF、優先權、RR(切片10毫秒)

行程	CPU 暴衝時間（毫秒）	優先權	到達時間
P1	10	1	1
P2	29	2	0
P3	3	0	2
P4	7	1	3
P5	12	2	4

平均等待時間：?? 毫秒，平均回覆時間：?? 毫秒

90

排隊模式

- 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

91

排隊模式

- Little's formula(李特式公式)

  $n = \lambda \times W$

  n為平均的佇列長度
  W為行程在佇列中等待時間
  λ為新行程平均到達速率

- 一個行程在等待W時間內，就有λ*W個新行程進入佇列中。
- 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(λ)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。

92

模擬

- 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：監看系統中事件發生的順序並記錄下來
- 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。愈詳細的模擬所需的時間也就愈長

93

模擬

- 為了得到更正確的排班演算法之評估，一般都採用模擬方式。

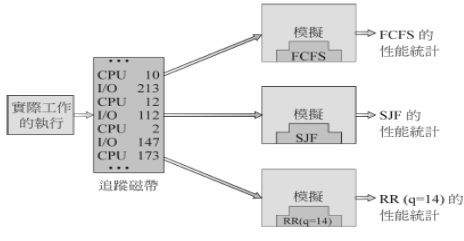


圖 5.15　以模擬方式評估 CPU 排班程式

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，使用者只求程式能很快地執行出來，經常改變作業系統及排程器，對使用者的工作幫助不大。
- 另一個困難點在於評估的環境是變動的，使用者可根據排程特性，將行程切割成適合很快執行的方式，如此會影響實際得到的評估結果。

95

摘要

- 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。

96

摘要

- 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。

97

摘要

- 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

98