綜合整理

各種排程器的比較表：

排程器類別	主要工作內容	主要目的	工作時間的限制
長程排程器	長程排班程式會從disk中的行程池（pool），選出行程，並且將它們載入到記憶體內以便執行。	長程排班程式掌控了多元程式度。	有最長的時間
中程排程器	當系統因過多或過少行程而降低效能時，搬出或搬進一些行程。	搬出（swap out）那些在記憶體中停留很久而仍未完成工作到儲存裝置，已空出的記憶體供其他人使用，以提升記憶體的使用率。等到記憶體有空時，再將其搬入（swap in）繼續執行，通常用在分時系統中。	介在兩者之間
短程排程器	從ready queue中選出行程到running queue中	提高CPU的使用率	時間最短

五、行程排程演算法

行程的排程法的優劣，將會影響電腦系統的效能。下面將介紹幾種常見的排程演算法。

1. 先到先服務法（First-Come First-Served, FCFS）

FCFS演算法十分簡單，它採用一個先進先出的佇列，誰先進入等待佇列，誰就能先占用CPU，且一旦將CPU分配給某行程後，該行程就會一直占用CPU，直到該行程結束或該行程執行等待I/O事件時而釋放出CPU的使用權，才會交給下一個在等待佇列中的行程。我們可以用FIFO（First In First Out）佇列來執行FCFS的工作。先到先服務法的缺點是：

在此法下，行程的平均等待時間經常是很長的，而且FCFS的平均等待時間會隨著CPU分割時間的大小變化而差異甚大，另外會產生所謂的護送效應（convoy effect）現象。所謂護送效應就是說，有很多程式必須等待某個大程式的完成之後，才能繼續工作的現象。日常生活中常會發生護送效應，例如：上、下班時間，在速度較慢且體積大的公車後面，都會擠滿了過不去的小客車或摩托車的情況。

2. 輪流法（Round Robin, RR）

輪流法則的效能完全取決於時間量（time quantum）的長短。若是這個時間量非常大的話，那麼RR就跟FCFS排班是一樣的；若是時間量非常短，則內容轉換（context switch）會經常發生，這對於系統的執行效率會大打折扣，因此一般要求時間量的長短應該要比內容轉換所需的時間要來的長些。

3. 最短優先法（Shortest Job First, SJF）

這種演算法就是不斷地指定給下一個需要CPU burst最短的行程，如果兩個行程具有相同長度的CPU burst，那麼就採用FCFS的方法來決定。

4. 優先權排班法（Priority Scheduling, PS）

每一個行程都有它自己的優先順序，CPU將分配給具有最高優先權的行程，若具有相同優先順序的行程，則按照FCFS來排班即可。SJF其實就是優先權排班演算法的特例。優先權排班演算法會有饑餓現象發生，解決方式就是採用變老（aging）技術，行程的變老技術是提高等待時間較久的行程的優先權，進而改變其使用CPU的優先順序。

綜合整理

排程演算法比較表：

排程演算法	方法說明	缺　點	優　點	改進方案
FCFS	1. 先進入FIFO佇列的行程先執行。 2. 行程終止或執行I/O時才轉換為下一個行程。	1. 不適用於分時系統。 2. 會發生convey effect。 3. CPU的使用率低。	最簡單。	無。
SJF	1. CPU burst最短的先執行。 2. CPU中的行程執行完畢時才換下一個行程。	1. 無法製作真實的SJF。 2. 會發生饑餓現象（starvation）。	有最小的平均等待時間。	無。
Priority	1. 有高優先權的行程優先執行。 2. 高優先權的行程執行完時才會換下一個行程。	會發生饑餓現象。	可讓緊急事件優先處理。	用變老技術（aging）可以解決饑餓現象的問題。
RR	1. 以FCFS佇列輪流使用CPU。 2. 每個行程分配相等的時間量。 3. 時間用完或執行I/O就換下一個行程。	若時間量太大，則RR成為FCFS，反之時間量太小，則context switch的負擔過重。	1. 適合分時系統。 2. 較公平。	選擇適當的時間量的長短，一般來說應該要比內容轉換的時間要來的長些。

備註

有一些參考書會以是否為可強奪的（preemptive）來進一步細分，例如：FCFS是不可強奪的（non-preemptive）的演算法，RR是可強奪的（preemptive），SJF和PS則可以是可強奪的或是不可強奪的。