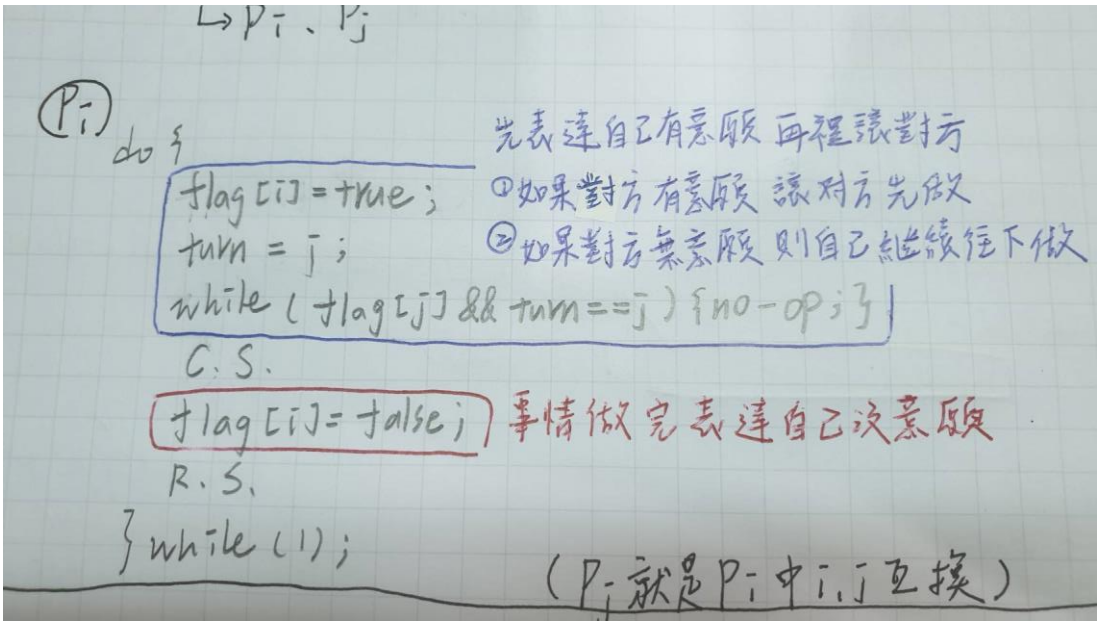
Peterson's Solution 在現代硬體在優化執行流程時為什麼會有問題 1081420 王藝喬

1. 定義什麼是 CS 問題

假設一個情況，一張能存錢提款的銀行卡可以讓兩個人使用(例如有兩個使用者 A 和 B)，有一天 A、B 兩人同是去存錢或是提款，但是他們的存款數目是共用的資料(share data)，那若是在 A 提款的操作中參雜 B 提款的操作就會造成裡面共用的存款的數目發生錯誤的存取，若此狀況發生在 processor 中兩個或是多個 process 有共用資料，就叫做 critical section problem。在 process 中有一段程式碼是負責更改共享資料的地方，叫做 critical section 簡稱 CS，當程序跑到這裡時，其他 process 不能進入，且不能被中途打斷。CS 問題有三個部分要解決，第一個是 mutual excusion(互斥)，在一段時間點只能讓一個 process 進入它自己的 cs 活動;第二個是 progress(前進)，不想進入 cs 的 process 不能阻礙和影響其他 process 進入 CS，並且要在有限的時間中選出下一個要進入 CS 的 process，關於這點也會牽扯到 no deadlock 的問題(系統中有一組 process 互相等待對方所擁有的資源，造成其他 process 無法繼續往下跑，降低了 cpu utilization，容易發生在 non-preemptive 的環境中)；第三個是 bound waiting(有限等待)，從 process 申請進入 cs 到獲取資格進入 cs 的等待時間是有限制的，避免其他 process 無限等待得不到資源(no starvation)，若有 n 個 process，任一個 process 最多等待 n-1 個有限時間就可以進入 cs。若用上廁所來解釋，mutual excusion 就是同一間廁所只能一個人上，progress 是當廁所沒人使用時要找下一個人進入，bound waiting 是一個人不能上太久會害其他人不能上。

2. 解釋 Peterson's Solution

有兩個 process 分別為 Pi 和 Pj，共享兩個變數，第一個 int turn(初始值為 0，turn=i 或者 turn=j 表示 Pi 和 Pj 誰可以進入 CS)，第二個 boolean flag[2](初始都為 false，代表想不想進入 CS)。程序的大概流程用下圖表示。



3. Peterson's Solution 在現代硬體為何可能出錯

在只有一個 CPU 的系統中，process 有照正確順序執行，Peterson's Solution 是可以正常運作的。但在多個 CPU 的系統中使用 Peterson's Solution 可能會因為要提高效率使用 out-of-order execution，或是多個 CPU 之間引起 out-of-order execution，而出現兩種狀況導致在不同 CPU 的執行順序不同步。第一種，可能會提早進入 CS，但是這個 CS 正在被使用；第二種，可能會提早退出 CS，但是關於共享資料的工作還沒有做完。

4. 解決方案

為了確保 CPU 內部的事件執行順序和外部其它 CPU 一樣，就需要用到一種需要硬體上的協助的技術叫做 Memory Barriers。

Memory Barriers 是一種 Barrier (computer science) instruction 可以分成兩種，Compiler Barrier 和 CPU Barrier。硬體為了減少讀寫 memory 而產生 cache。而 cache 要同步 cache 之間的資料 ，同時同個位置只有一個值，用 memory barrier 確保共享的資料有依正確的順序更新。