主因: shared Data同時會被多個程式修改 (data inconsistency)

解決方法: 確保執行順序是對的(例如:不能交錯)

Ex: consumer & producer problem

我們之前使用buffer，會浪費一格空間

那我們何不使用counter紀錄buffer資料數?

Ans:會發生synchronization問題，counter是共用資料

比如使用counter++

在組合語言其實分為三道指令

Move a counter

Add a 1

Move counter a

Counter—同理

Move a counter

Sub a 1

Move counter a

如果今天不確保順序

Move a counter

Add a 1

Context switch

Move a counter

Sub a 1

Context switch

Move counter a counter = 6

Context switch

Move counter a counter = 4

此程式碼counter++ 、 再 counter--後，可能依執行順序會發生4、5、6的結果

Race conditon

最後(修改)執行的程式會決定shared data的值

(synchronized 同步化才能解決)

解法1: 暴力禁止interrupt => 使用者不可能這樣(OS會炸掉)、OS才能這樣做

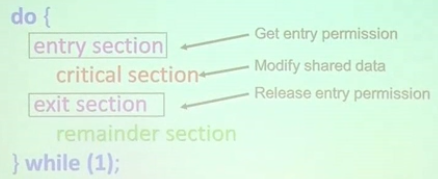
解法2: Critical section problem 解法 (常用)

Critical section

1. 一個對共同執行的程式的protocol
2. 定義: N個process要使用同個shared data

共用的那段即稱為Critical section

1. 演算法: 決定進入與出去 Critical section



1. 三大性質要求: 1. Mutual exclusion

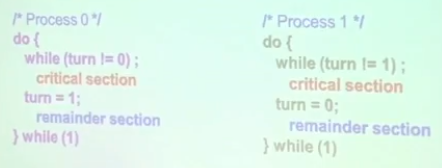
2. progress (有人有權利使用但沒有意願，要讓給有意願的人)

(有意願要可以搶走無意願者的權利 => 不能讓資源鎖住)

3.bounded waition (有限等待時間，有意願的人必須一定等的到) (不會被插隊)

考慮兩個process:

Naïve 解法



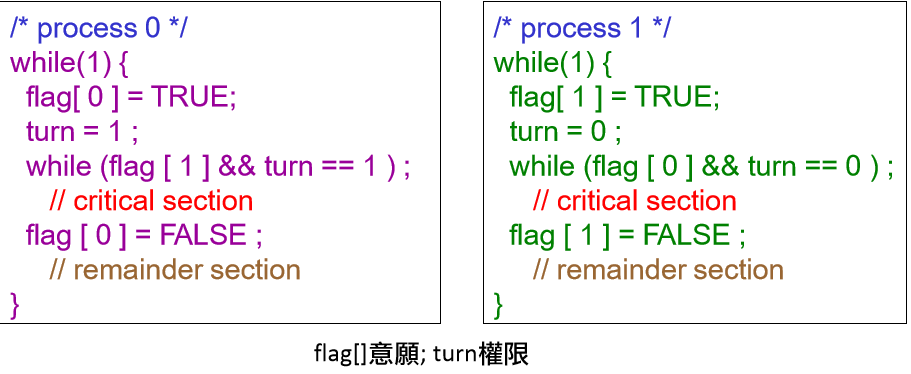
ME: check，turn不可能同時是1且0

Progress: No，progress出去後，把turn調成別人，但有可能別人卡在remainder section(有權利但是不用)

Bounded wating : check，結束後會把turn給別人，無法插隊

範例解法 peterson`s solution

給定變數 權力 與 意願



在進去前會先有意願，把權力給別人

然後如果別人有意願且有權力，就要在外面等

離開時要把意願關掉

ME:turn 不可能同時為1 and 0

Progress: 出去時會把意願放掉，別人一定進的去 (不會讓系統因為無意願者鎖住)

Bounded waiting 有意願時會先把權力給別人 (防止插隊)

ME 要有不重疊的權利

Progress 要有意願 (解決dead lock)

BW 要有輪流、排隊 (解決starve)