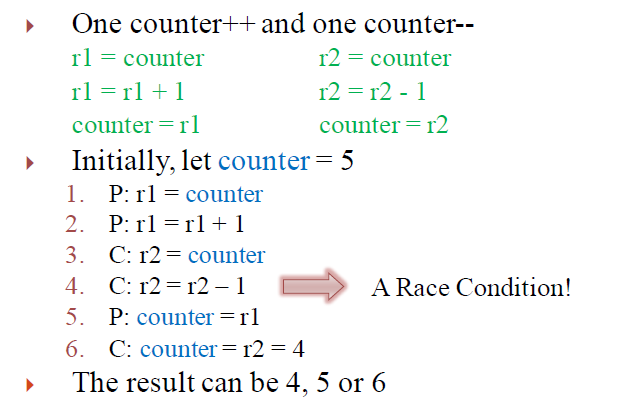
3CH06-Synchronization

Race Condition (競爭情況)



↑當兩個行程並行處理counter這個變數(count++ and count--)，像這種數個行程同時存取和處理相同資料的情況，而且執行的結果取決於存取時的特殊順序，就為競爭情況。

Process Sychronization

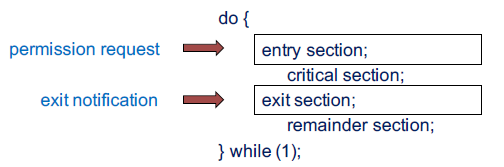
* A Race Condition：

執行結果取決於存取時的特殊需求的情況

* The Critical-Section(臨界區間) Problem：

設計一個協定，使得各行程能夠互相合作

* 每個行程都有一段code稱為critical section，行程間這一段code必須互相排斥(當一個行程在臨界區間內執行，不允許其他的行程在他們的臨界區間內執行，也就是指沒有兩個行程同時在他們的臨界區間執行)
* 臨界區間設計的基本結構



Solution of the Critical Section Problem

Three Requirements：

* Mutual Exclusion(互斥)

只有一個行程能在它的臨界區間內執行

* Progress(進行)

如果沒有行程在臨界區間內執行，同時某一行程想要進入其臨界區間，只有不在剩餘區間內執行的行程才能決定誰將進入臨界區間，且這個選擇不得無限地延遲

* Bounded Waiting(限制性的等待)

一個已要求在等待進入臨界區間的行程，其需等待其他行程進入臨界區間的次數有一個限制

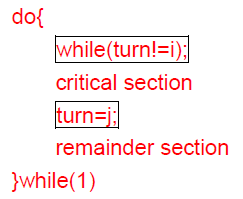
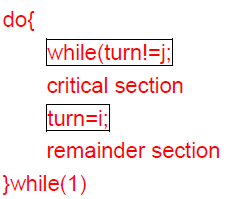
Peterson’s Solution

* Process Pi , Pj
* 假設所有指令的執行都為一單位

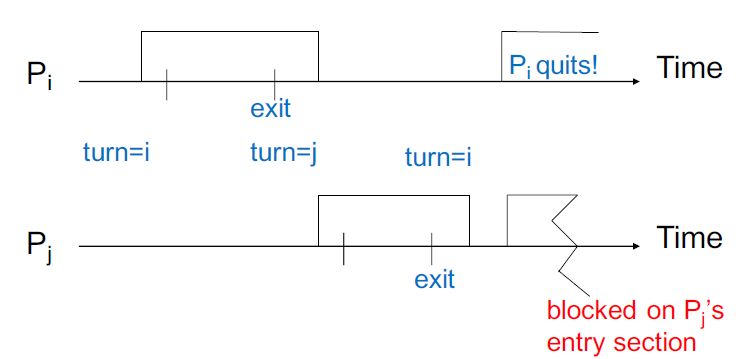
1. Algorithm 1

Idea：記住哪個process可以進入臨界區間，若為Pi可進入，則turn=i。

↓Pi ↓Pj

Algorithm 1 fails the progress requirement：

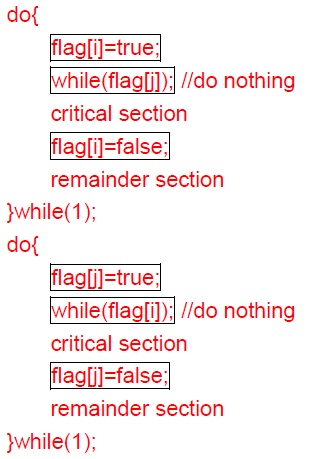
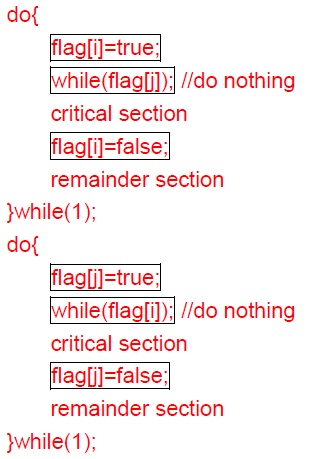


↑不滿足progress，假設turn值為i，但是Pi並不想要進入，也就沒有機會將turn值改為j，這時如果Pj想進去也沒有辦法。

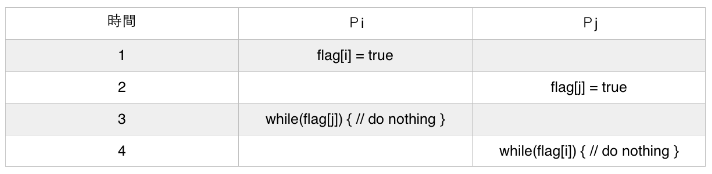
1. Algorithm 2

Idea：記住每個process進入的意願，flag[i]==true，表示Pi準備好要進入臨界區間。Initially，flag[i]=flag[j]=false

↓Pi ↓Pj

Algorithm 2 fails the progress requirementl：



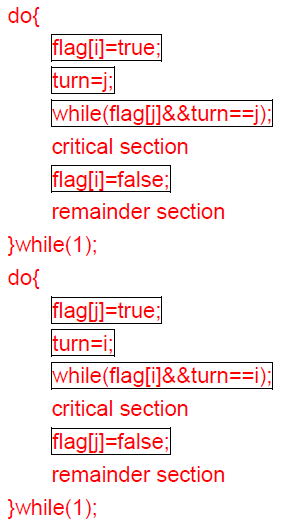
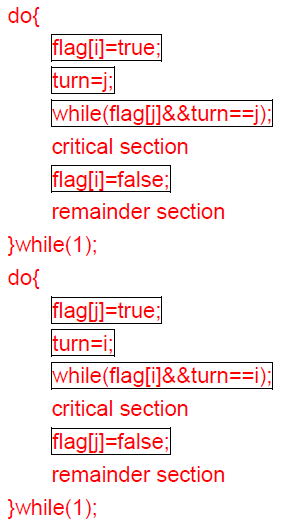
↑不滿足progress，當flah[i]=flag[j]=true，大家都進不去

1. Algorithm 3：

Idea：結合Algorithm 1 和 Algorithm 2，當 (flag[i] && turn=i), Pj

must wait，Initially，flag[i]=flag[j]=false，and turn = i or j

↓Pi ↓Pj

滿足三個條件！

1. Mutual Exclusion：

滿足，若Pi與Pj皆想進入自已的臨界區間，代表 flag[i] = flag[j] = true，而且會分別執行到 turn = i 及 turn = j 之設定 (只是先後次序不同)。因此，turn 値僅會是 i 或 j，絶不會兩者皆是。 ∴ Mutual Exclusion得以確保。

1. Progress：

滿足，若Pi不想進入臨界區間，則表示 flag[i] = false。此時若Pj想進入自已的臨界區間，必可進入，而不會被Pi阻礙。若Pi與Pj皆想進入自已的臨界區間，則在有限的時間內，Pi與Pj會分別執行到 turn = i 及 turn = j 之設定 (只是先後次序不同)，turn値必為 i 或 j。∴Pi(或Pj) 會在有限的時間內進入自已的臨界區間。

1. Bounded Waiting：

滿足，假設兩個 Process 同時都想進入自己的臨界區間，如果最後是Pi 順利進入而Pj 等待。假如Pi在執行完臨界區間之後想馬上再進去，但因為在進入前會把 turn 設成 j 且因為Pj在等待中而無法進入，相當於無法搶先Pj，換句話說，每一個 Process 至多等待一次就可以進入。

Synchronization Hardware

* 動機：硬體上的特殊性質可以使寫程式的工作變得比較容易，並且增進系統的效率
* 方法：

不能中斷(共用變數被更改時不讓中斷發生)→無搶先

* + - 在多處理器的系統並非可行
    - 對由中斷驅動的系統時鐘有潛在的影響

單元硬體說明

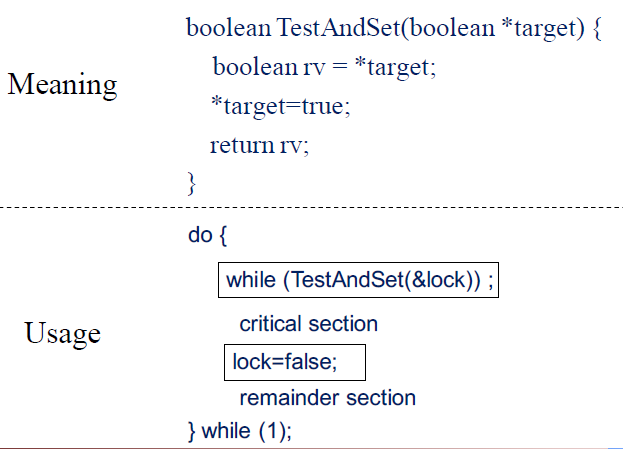
* Test-and-set, Swap

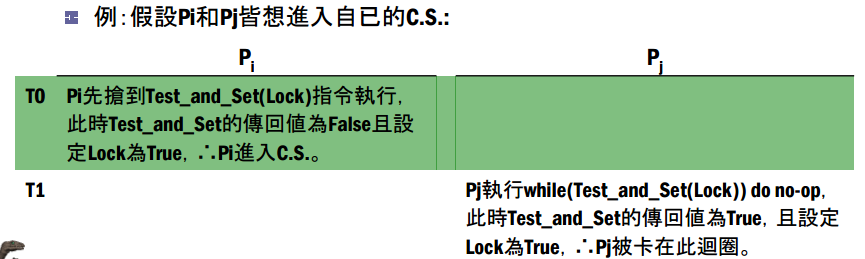
Test-and-Set

可以在一個不可中斷的單位時間中被執行

藉著布林變數lock的初值為false來塑造出互斥性

滿足互斥，不滿足有限等待





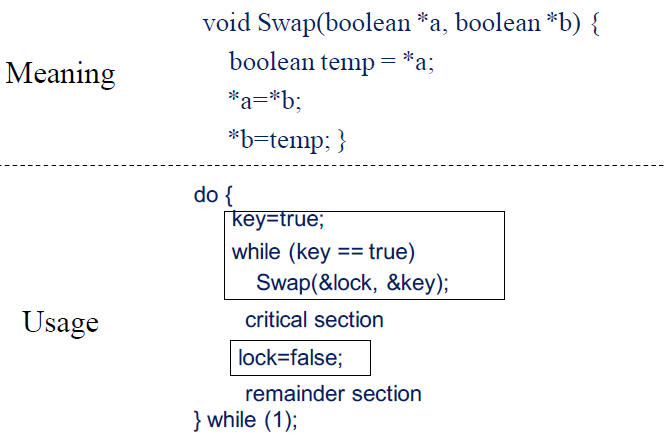


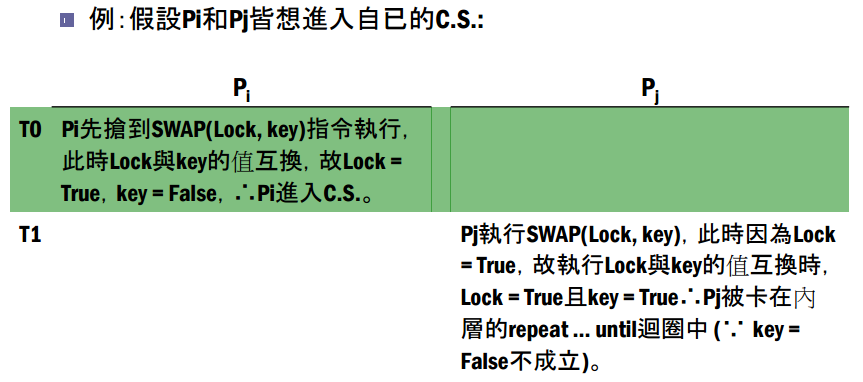
Swap

可以在一個不可中斷的單位時間中被執行

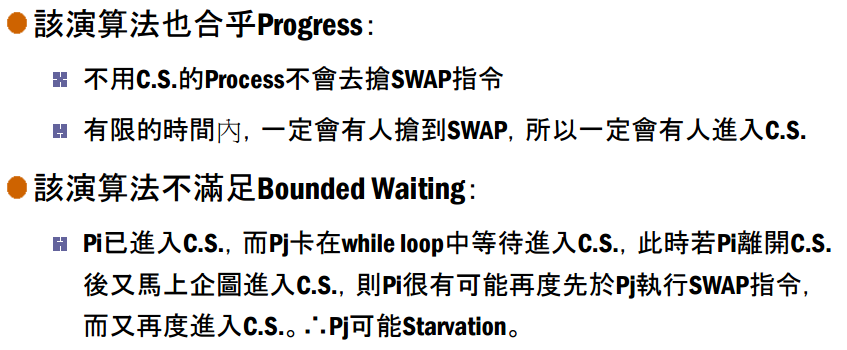
lock和key初值皆為false

滿足互斥，不滿足有限等待



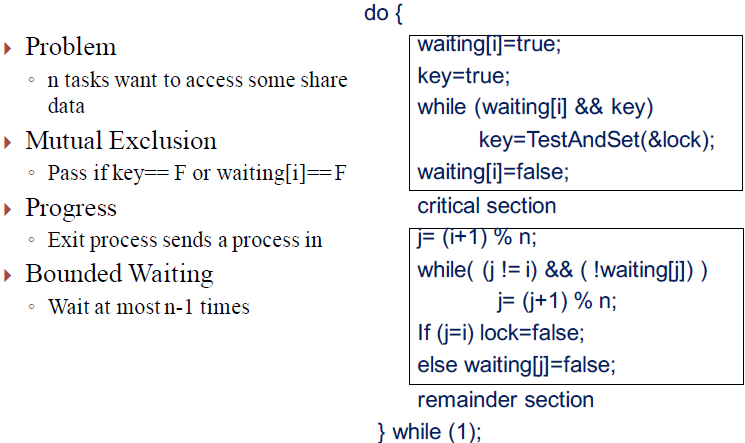






Case Study of Test and Set

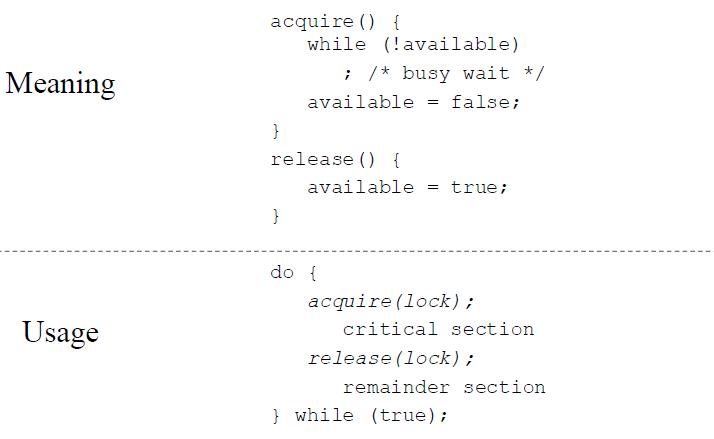
可以在一個不可中斷的單位時間中被執行



Mutex Locks(互斥鎖)

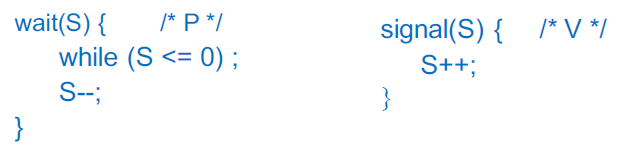
* 行程進入臨界區間時需要取得鎖acquire()，離開前要釋放鎖release()
* acquire()和release()要不可分割的執行
* 缺點：此做法需要busy waiting(當一個行程正在臨界區間執行時，任何試圖進入臨界區間的行程必須在呼叫acquire()的地方不斷地執行)

這種lock與被稱為spinlock(自旋鎖)



Semaphore(號誌)

* 動機：對更複雜的問題有更高階的解決方法
* Semaphore：變數S只被兩種單位操作取得↓



Precedence Enforcement

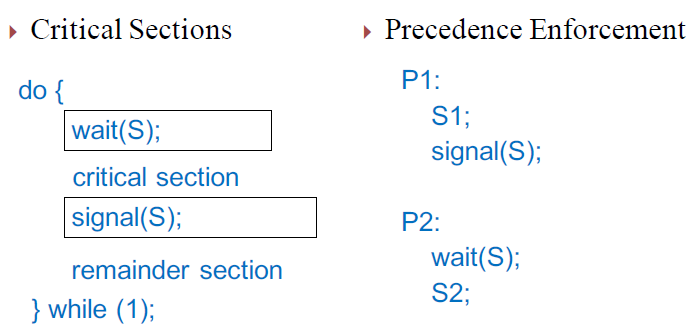
規定S1的敘述一定要在S2敘述之前執行

S：Semaphore初值=0

Semaphore初值定義：

作為互斥控制之用：1

作為強迫暫停之用：0



* 實作：

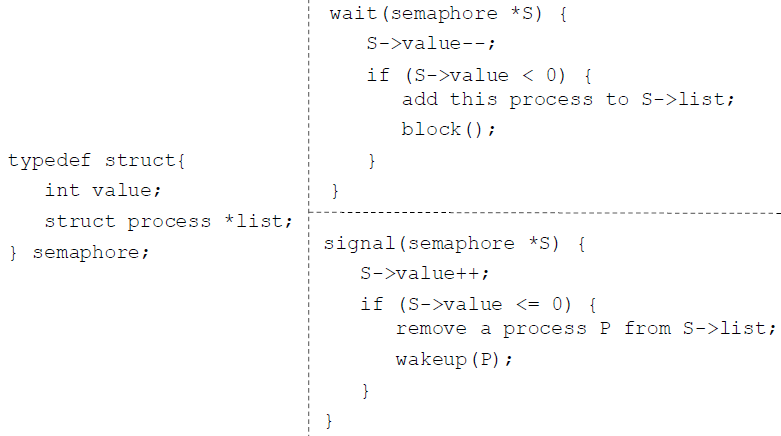
Spinlock：A Busy-Waiting Semaphore

* While(S<=0)造成CPU cycles的浪費
* 優點：當一個鎖被把持很短的時間時，spinlock是很有用的，因為他不需要內容轉換

Semaphore with Blocked-Waiting

* 進入臨界區間時不需busy waiting

Semaphores with Block Waiting

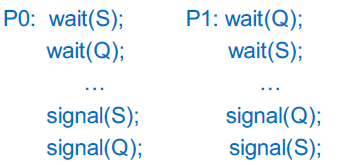


* + block()運算暫停呼叫他的行程
  + wakeup(P)運算則恢復被閉鎖的行程P的執行

Deadlocks and Starvation

* + Deadlock

當一個佇列裡的行程皆需等待來自另一個行程的一個事件( signal() )發生才能夠執行時，就會造成死結



假設P0執行wait(S)，然後由P1執行wait(Q)。當P0執行Wait(Q)時，它必須等到執行完signal(Q)後才能執行，同樣地，當P1執行wait(S)時，它也必須等到P0執行完signal(S)後才開始。P0和P1就進入死結。

* + Starvation

LIFO(last-in,first-out)後進先出

Classical Problem of Synchronization

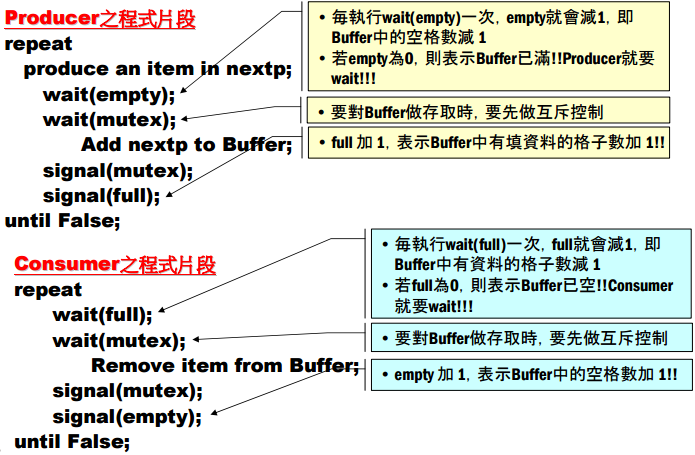
* Bounded-Buffer Problem
* Readers and Writers Problem
* Dining-Philosophers Problem

Bounded-Buffer Problem

↓Producer和Consumer的對稱性，我們可解釋為：生產者對消費者生展滿的緩衝區or消費者對生產者產生空的緩衝區

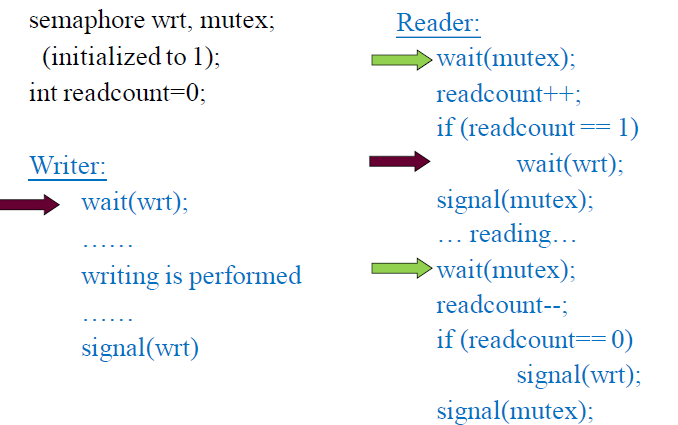
三個共享的Semaphore：

* mutex:Semaphorel：對buffer存取提供互斥控制，初值=1
* empty:Semaphorel：紀錄Buffer內現有的空格數，初值=n
* full:Semaphore：紀錄Buffer內有資料存在的格數，初值=0

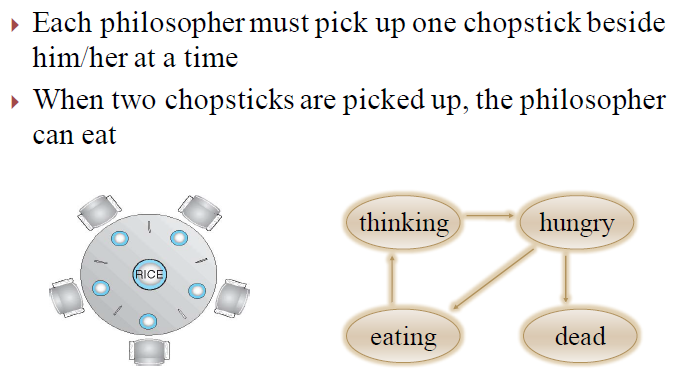
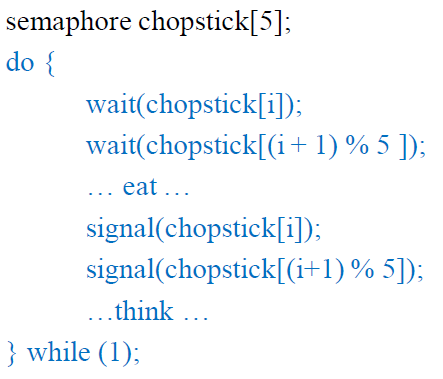


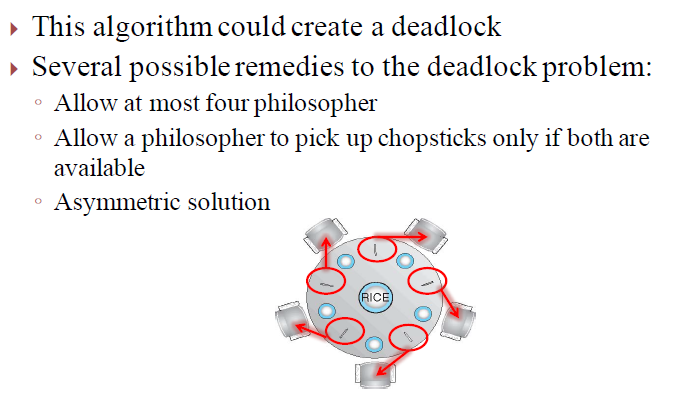
Readers and Writers Problem

* 一個資料集可以被許多並行行程所共用
  + Readers只能讀data set，不能做任何修改
  + Writers可以讀和寫
* 問題
  + 允許多個Readers同時讀取
  + 只有一個Writer有共用資料的存取權



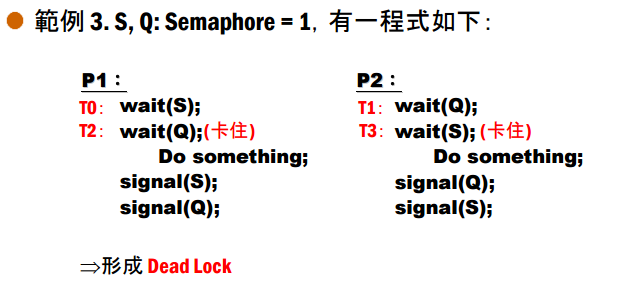
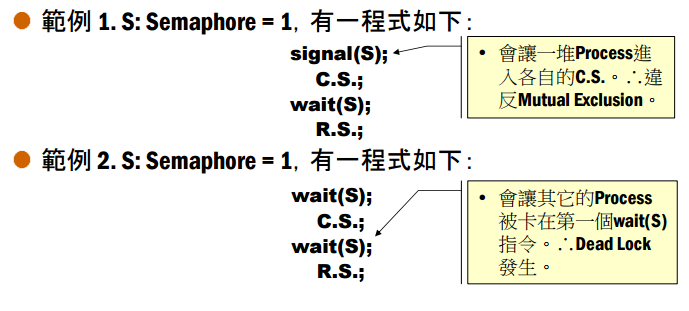
Dining-Philosophers Problem



Problems with Semaphores

* 錯誤使用wait和signal指令

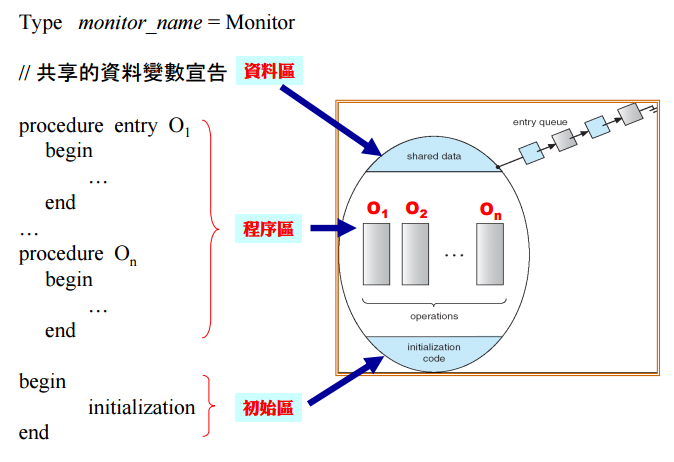
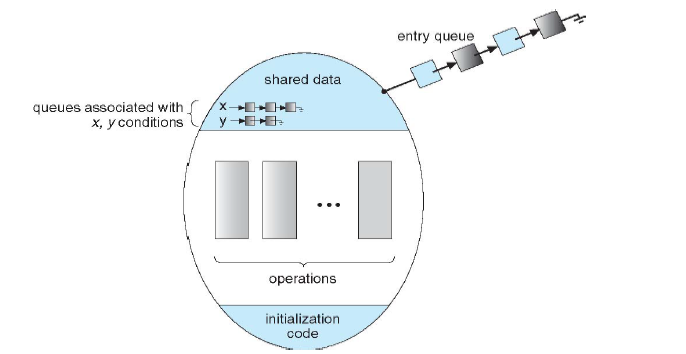


* 造成deadlock和starvation

→使用一種高階的同步結構→監督程式monitor

Monitor

* 由三部分構成：
* 共享資料區：宣告一些共享變數，此變數只提供給montitor內的程序使用
* 一組Procedures：讓外界呼叫用來取得share data內的資料
* 初始區：設定共享資料的初值

* Condition Variables
  + 假設宣告x為Condition型態變數→x:Condition
  + 提供兩個Atomic Operations：
    - x.wait：強迫process暫停，把process放到waiting queue中，直到x.signal()
    - x.signal：如果有process卡在waiting queue中，則呼叫x.signal，會從waiting queue中wakeup第一個process，若waiting queue中沒有被x.wait暫停的prcess，則x.signal無任何作用