並行計算

EP06: 同步（Synchronization）初探

很難教菜鳥上而下（**top-down**）程式寫作，
因為他們根本不知道哪邊是「上」。

Tony Hoare

本集內容

- 為什麼需要同步（Synchronization）
- 競爭狀況（Race Condition）
- 執行序列（Execution Sequence）
- 臨界區（Critical Section）
- 互斥（Mutual Exclusion）
- 臨界區的入口和出口
- 解決臨界區問題的「好」解法
  - ✓互斥（Mutual Exclusion）
  - ✓有進展（Progress）
  - ✓有界等待（Bouded Waiting）
- 其它相關論題

請看影片的說明，那兒有取得投影片和程式的連結

頭痛時間開始

同步（Synchronization）初探

和同步（synchronization）有關的課題

- 競爭狀況（race condition）
- 臨界區（critical section）

本講主題（觀念部份）

- 純軟體解法
- 硬體支援
- 訊號機（semaphores）
- 再探競爭狀況
- 監督程式（monitors）

需要同步! 1/6

```
int  a[3] = { 3, 4, 5};
```

Process 1

```
a[1] = a[0] + a[1];
```

Process 2

```
a[2] = a[1] + a[2];
```

```
a[3] = { 3, ?, ? }
```

整道高階語言敘述交錯執行

需要同步! 2/6

```
int  a[3] = { 3, 4, 5};
```

Process 1	Process 2
`a[1] = a[0] + a[1];`	`a[2] = a[1] + a[2];`

- 若process 1先更新 a[1]， a[1]為7，而 a[ ] = { 3, 7, 5} 。
- 然後，process 2使用 a[1] 的新值計算 a[2] ，於是 a[ ] = { 3, 7, 12} 。

- 若process 2先使用 a[1]， a[2]為9，而 a[ ] = { 3, 4, 9} 。
- 然後，process 1計算 a[1] ，於是 a[ ] = { 3, 7, 9} 。

結果是不確定的！

實際情況可能更糟。

需要同步! 3/6

```
int  Count = 10;
```

Process 1 **Process 2**

```
Count++; Count--;
```

Count = **9, 10 or 11?**

高階語言的敘述會被轉譯成機器指令，
而不是被CPU從頭到尾一氣呵成執行完成的！
換言之，在CPU中執行一道敘述的機器指令時
中間可能會被interrupt打斷!

需要同步! 4/6

```
int  Count = 10;
```

Process 1

```
LOAD  Reg, Count
ADD   #1
STORE Reg, Count
```

Process 2

```
LOAD  Reg, Count
SUB   #1
STORE Reg, Count
```

轉譯成機器指令執行時，可能會被打斷，結果變成不確定。

需要使用機器指令交錯執行

需要同步! 5/6

Process 1		Process 2		
指令	暫存器	指令	暫存器	記憶體
LOAD	10			10
		LOAD	10	10
		SUB	9	10
ADD	11			10
STORE	11			11
		STORE	9	9

新值9蓋過舊值11

記得用機器指令交錯執行解釋競爭狀況

需要同步! 6/6

Process 1		Process 2		
指令	暫存器	指令	暫存器	記憶體
LOAD	10			10
ADD	11			10
		LOAD	10	10
		SUB	9	10
		STORE	9	9
STORE	11			11

新值11蓋過舊值9

記得用機器指令交錯執行解釋競爭狀況

競爭狀況

- 若下列條件同時滿足，就會出現競爭狀況（Race Condition）：
  - ✓兩個process或thread並行使用一個共用的資源，
  - ✓結果取決於兩個process/thrad使用該資源的順序。
- 得用同步的技巧避免競爭狀況發生。
- 同步是一個有相當難度的課題，不要錯過此地的任何討論細節；有必要時不妨一再觀看、並且留言發問。

細說執行序列: 1/3

- 必須要用機器指令的交錯執行證明競爭狀況的存在，因為：
  1. 高階語言的敘述會被轉譯成機器指令，而interrupt會在機器指令間發生、從而產生環境切換；這樣，一道敘述在執行中途就變打斷而交由另一個process/thread執行。
  2. 機器指令交錯執行會很清楚地說明若干process/thread並行地共享某個資源。
  3. 因為競爭狀況會從不同執行順序而產生不同結果，因此需要兩條交錯執行的執行序列（execution sequence）分別得到不同的結果。

細說執行序列: 2/3

```
int  a[3] = { 3, 4, 5};
```

Process 1

```
a[1] = a[0] + a[1];
```

Process 2

```
a[2] = a[1] + a[2];
```

執行序列之一

Process 1	Process 2	Array a[ ]
a[1]=a[0]+a[1]		{3, 7, 5}
	a[2]=a[1]+a[2]	{3, 7, 12}

執行序列之二

Process 1	Process 2	Array a[ ]
	a[2]=a[1]+a[2]	{3, 4, 9}
a[1]=a[0]+a[1]		{3, 7, 9}

此地並沒有並行分享行為，
這不是一個正確的證明
有競爭狀況的例子，
雖然結果不同。

細說執行序列: 3/3

```
int  Count = 10;

Process 1                    Process 2
LOAD  Reg, Count             LOAD  Reg, Count
ADD   #1                     SUB   #1
STORE Reg, Count             STORE Reg, Count
```

變數count被並行地共享
就很清楚地展示出來

Process 1	Process 2	Memory
LOAD Reg, Count		10
	LOAD Reg, Count	10
	SUB #1	10
ADD #1		10
STORE Reg, Count		11
	STORE Reg, Count	9

臨界區

- 一個**臨界區**（**critical section**，簡稱**CS**）是一個程式中會使用（未必是更新）某個共用資源的地方。

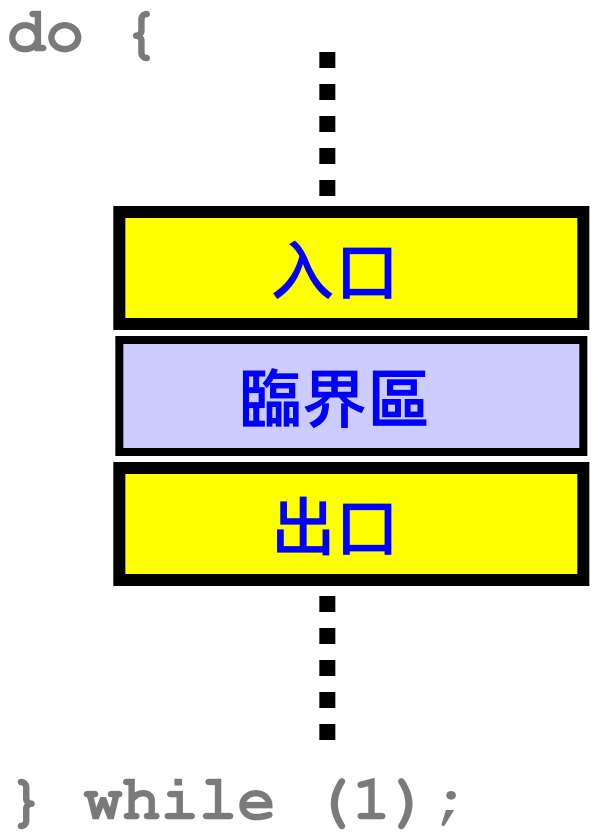
這幾個臨界區都是有關共用資源count的地方

每一個共用資源都有它的臨界區

互斥（Mutual Exclusion）

- 為了要避免競爭狀況，進入臨界區時必須互斥，也就是在任何時刻最多只會有一個process/thread會在某個共用資源的臨界區中執行。
- 臨界區問題（critical-section problem）就是如何設計進入臨界區和離開臨界區的程式，使得臨界區的使用方式是互斥的。

臨界區的入口/出口的執行方式



- 臨界區的執行方式需要一個入口和一個出口。
- 入口和出口之間就必須要滿足互斥的臨界區。

解決臨界區問題的「好」解法

- 一個解決臨界區互斥問題的好解法得滿足下列三個條件：
  - ✓互斥（mutual exclusion）
  - ✓有進展（progress）
  - ✓有界的等待（bounded waiting）
- 更重要的是，這個解法不能依賴CPU的速度、調度程式的做法、執行時間等等的外在因素。
- 還有，一個解答未必能滿足所有條件，但卻一定得做到互斥、因為這是最低限度的要求。

互斥（Mutual Exclusion）

- 如果一個process已經在臨界區中執行，其它任何想進入臨界區的process都不可以進入為同一資源而設的臨界區。
  - ✓臨界區入口部份必須要能阻檔想進入臨界區的process（如果該臨界區已經有process在執行）。
  - ✓在臨界區執行的process到達出口時，必須要有什麼方法讓入口知道，以便入口讓一個在該處等待的process可以進入臨界區。哪個process可以進入並不十分重要，只要有一個可以進入就行，於是互斥條件就滿足。

有進展（Progress）

- 如果臨界區中並沒有process在執行、而且又有一些process正在等待進入臨界區，於是：
  - ✓只有那些正在等待進入臨界區的process可以參與競爭進入臨界區，其它的process不可以影響誰可以進入的決定。
  - ✓在那些等待進入臨界區的process中挑一個可以進入的決策時間不可以無限延遲；也就是說，在有限時間內，入口部份必須要能從等待的process挑一個、讓它進入臨界區。

有界的等待（Bounded Waiting）

- 當一個process（對入口）提出進入臨界區的要求時，在它提出要求到它進入時，最多只能有不超過某個既定值的process可以進入。換言之，這個既定值限制了在要求進入、到實際進入之間最多有多少個其它process可以進入的界限。
- 有界和有限的差異。有限指的不是無窮，任何可以寫下來的值（譬如幾兆）都是有限，但有界的意義是這個值不可以比某個既定的值來得大。這個既定值就是界限（bound）。

有進展 vs. 有界等待

- 有進展（progress）並不表示有界等待（bounded waiting）：有進展指的是在等待（進入臨界區）的**progress**中在有限時間內（如果臨界區中是空的）會做成挑一個可以進入臨界區的決定，但是卻不保證一個**process**會等多久，因為有進展指的是做（挑出一個**process**的）決定的時間、並不是某個**process**要等多久。
- 有界等待也不保證有進展：縱使我們有一個界限，所有在等待的**process**很可能會被困在入口處（無限的決策時間）。
- 所以，有進展和有界等待是兩個相互獨立的條件，不可以混為一談。

幾個相關的名詞: 1/6

- **Deadlock-Freedom**（或**Deadlock Free** — 無死鎖）：若有兩個或兩個以上的**process**等待進入臨界區，有一個（而且僅容許一個）最終可以進入；換句話說，挑選一個可以進入臨界區的決策不會無限延遲。
- 在有進展的前提下，不但決策不會無限延遲，而且還要求不打算進入臨界區的**process**不能干擾這項決策；換句話說，決定一個可以進入臨界區的**process**的決策只和等待進入的**process**有關、和目前不打算進入的**process**無關。
- 所以，無死鎖條件就是把有進展條件中的有限決策時間部分保留、而刪掉了非等待**process**不能影響決策的條件。是故，無死鎖條件是比有進展條件來得弱。

幾個相關的名詞: 2/6

- **Starvation-Freedom**（或**Starvation Free** — 無恆久等待）:若一個**process** 正在等待進入臨界區，它最終一定可以進入。
- **問題**：
  1. 無恆久等待（**Starvation Free**）導致無死鎖（**Deadlock Free**）？
  2. 無恆久等待導致有界等待？
  3. 有界等待導致無恆久等待？
  4. 有界等待加上無死鎖導致無恆久等待?

幾個相關的名詞: 3/6

- 問題1 ： 無恆久等待（Starvation Free）導致無死鎖（Deadlock Free）？
- 是的。若每一個等待的process最終都可以進入臨界區（雖然它可能會等很久），這表示在有限時間內入口部份終究可以選出一個可以進入的process，這表示決策部份所用的時間是有限的，當然就沒有死鎖。
- 要不然（無限決策時間）根本就選不出可以進入臨界區的process，所以等待進入的process就得等待無限久，這就違反了無恆久等待的題設了。

幾個相關的名詞: 4/6

- 問題2 ： 無恆久等待（Starvation Free）導致有界等待?
- 非！ 無恆久等待只表示一個process終究可以進入臨界區，但並不保證它的等待時間不超過某個既定值（等待的界限）。

幾個相關的名詞: 5/6

- 問題3 ： 有界等待（Bounded Waiting）導致無恆久等待?
- 非！有界等待並不保證一個希望進入臨界區的process一定可以進入，它只保證了在該process進入之前只會有不超過某個定數的process會進入。好比說，如果所有等待進入臨界區的process都無法進入（也就是有進展條件中的有限決策時間不成立），於是恆久等待就發生了。
- 我們需要有進展+有界等待才能有無恆久等待。事實上，有進展+有界等待是一個比無恆久等待強的條件！ 為什麼? 請想一想。

幾個相關的名詞: 6/6

- 問題4 ： 有界等待加上無死鎖導致無恆久等待?
- 是！無死鎖保證有限時間內一定有一個process可以進入，有界等待指出一個等待中的process只需要等待不足某個定數的process「插隊」後就可以進入，這表示沒有恆久的等待。
- 因為無死鎖是有進展的一部份，所以無死鎖+有界等待是比有進展+有界等待來得弱的。

想一想

簡單的問題

問題: 1/2

Process A

```
----------
x += 2;
x++;
```

Process B

```
----------
x = 2*x;
```

答案： 0, 1, 3, 4, 5, 6 (但沒有2).

- 問題1 ：假設x是Process A和Process B的共用變數，初值是0（見左邊程式）。請問：這兩個process執行完後，x可能的值為何?

問題: 2/2

Process A

```
-----------
for (i = 1; i <= 2; i++)
     x++;
```

Process B

```
-----------
x = 2*x;
```

答案： 0, 1, 2, 3, 4.

- 問題2 ：假設x是Process A和Process B的共用變數，初值是0（見左邊程式）。請問：這兩個process執行完後，x可能的值為何?

我們學到了什麼?

- 什麼是同步、為什麼需要同步
- 什麼是競爭狀況
- 臨界區和它的用途
- 解決臨界區問題的三個重點
- 其它若干名詞

結束，謝謝收看！
期望您再次觀看下一集

請看影片的說明，那兒有取得投影片和程式的連結