

作業說明

- 請在下列的 40 個問題中，挑選並完成其中 20 個題目。
- 其中每一章節需要完成至少 2/5 的題目(4, 2, 4, 3 題)。
- 其餘七題可以自行挑選要完成哪個章節的那些題目。
- 每一題請標明章節-題號(例如 6-3, 8-4)並按照章節-題號順序作答。
- 請上傳完成的作業 pdf 檔案，可以使用手寫或打字。
- 可以參考投影片內容回答，但請不要直接複製投影片內容。
- 繳交截止時間 2023/5/18 09:10。

Chapter 6

1. ■請舉例說明什麼是競爭情況(Race condition)。

A: 在 shared memory communication 中，若未對共享變數的存取提供任和互斥存取控制之 Synchronization 機制，則會造成共享變數的最終結果值，會因為 Process 間的交錯執行，有不同的結果值，這種 data inconsistency 的情況稱為 Race condition。

2. ■請解釋何為臨界區間(Critical Section)。

A: Critical Section(CS)是解決 Race condition problem 的方法之一。

Critical Section 是對”共享變數”之存取進行管制。當 Process P_i 取得共享變數存取權利時，在 P_i 尚未完成的期間內，仍和其他 Process 都無法存取共享變數，即使取得 CPU 也一樣。

Process 可以分成 2 個部分，Critical Section 與 Remainder Section。CS 的部分是 Process 中對共享變數進行存取敘述之集合，而 RS 則是除了 CS 之外的部分，在 CS 的前後，都需要由 Programmer 設計(加入)額外的控制程式碼，稱作 Entry Section 與 Exit Section。而 CS Design 其實是在設計 Entry\Exit Section Code。

優點:適用 Multiprocessor System

缺點:設計較複雜、較不適用 Uniprocessor System

Critical Section Design 需要滿足的 3 個性質:

1. Mutual Exclusion:在任何時間點，最多只允許一個 Process 進入他自己的 CS，不可有多個 Process 分別進入各自的 CS。
2. Progress:
 - (1) 不想進入 CS 的 Process (或在 RS 內活動的 Process)不能阻礙其他 Process 進入 CS(或不參與進入 CS 之決策)
 - (2) 從那些想進入 CS 的 Process 中，決定誰可以進入 CS 的決策時間是有限的(不可無窮)，即 No Deadlock(不可 Waiting forever，不能大家皆無法進入 CS)

3. Bounded Waiting:從某 Process 提出申請到核准進入 CS 的等待時間(次數)是有限的。即若有 n 個 Process 想進入 CS，則這一個 Process 至多等待 $(n-1)$ 次就可進入 CS(No Starvation，需公平)。

3. ■在某些情況下，可以將關閉系統的中斷(interrupt)以解決臨界區間問題，請問這個解法可能有什麼問題？

A: 此方法即為 Disable Interrupt，做法是 Process 在對共享變數存取之前，先 Disable Interrupt(關閉中斷)，等到完成共享變數存取後，才 Enable Interrupt(開啟中斷)，可保護 Process 存取共享變數期間，CPU 不會被 Preemptive。

而此做法的優點有：

1. Simple ,Easy to implement
2. 適用於 Uniprocessor System(單一 CPU)

缺點：

1. 不適用於 Multiprocessors System，因為 Disable 一顆 CPU 的 Interrupt，是無法防止 Race condition(其他 CPU 上執行的 Process，仍有存取共享變數的可能)，必須 Disable 「全部」 CPUs 的 Interrupt，才可以防止 Race condition，但此舉會大幅降低 performance(因為無法平行處理)
2. 風險高，必須信任 User process 在 Disable Interrupt 後，短時間內會 Enable Interrupt，否則 kernel 可能會拿不到 CPU，產生極大風險。

4. ■請描述兩個程序的 Peterson's Solution 解法。

A:

1. 共享變數宣告:

```
1.int turn;      // = i or j
2.Boolean flag[i...j]; // 初值皆為 false
```

2. Code:

```
[ $P_i$  部分]
while(1){
    flag[i]=true;      //表  $P_i$  有意願
    turn=j;            //權仗給對方
    while(flag[j] && turn==j);
    //對方有意願且權仗在對方，則卡住
    /*C.S.*/
    flag[i]= false; //表  $P_i$  無意願
    /*R.S.*/
}
```

```
[ $P_j$  部分]
while(1){
    flag[j]=true;
    turn=i;
    while(flag[i] && turn==i);
    /*C.S.*/
    flag[j]= false;
    /*R.S.*/
}
```

5. ■請解釋何為記憶體屏障(Memory barrier)

A: 在 Multiprocessors System 中，Memory model 可分為兩種:

1. Strongly ordered:當其中一個 Processor 對 Memory 中的資料作修改時，其他 Processors 可以立即看見此修改。
2. Weakly ordered: 當其中一個 Processor 對 Memory 中的資料作修改時，其他 Processors 可能不會立即看見此修改。

而記憶體屏障(Memory barrier)是一個指令，用於確保對記憶體中的任何修改都能傳播（可見）給所有其他 Processors。Memory barrier 可以處理在 Weakly ordered 可能出現的資料不一致狀況。當執行 Memory barrier 指令時，系統會確保在執行後續的 read/write 操作之前，所有對記憶體的 read/write 操作都已完成。即使指令因系統最佳化等原因而導致亂序，Memory barrier 仍然能夠確保操作完成並傳播給所有其他 Processors。

6. ■請解釋 test and set()指令為何可以作為 entry section 使用。

A: Boolean Test_and_Set(Boolean *target) 是一種 CPU 指令(硬體指令)，其指令功能為 return target 舊值，並將 target 設為 true，且保證此指令是 atomically execution(不可分割的執行)。

而這種特性很適合作為 entry section 的使用:在共享變數宣告時將 target 初始化為 False，當一個 Process 想進入 CS 時，先執行 Test-and-Set 指令，檢查 target 變數的值並將其設置為 true，如果 target 舊值為 true 表 CS 內有 Process 在活動(擋住目前 Process)，False 則無(可順利進入 CS)，而當 Process 要離開 CS 時，需要執行 target=False(表示 CS 內無 Process，放行擋住的 Process)。

7. ■在 Bounded-waiting with compare-and-swap 的解法中，請解釋為何可以滿足 Bounded-waiting 的條件。

A: compare_and_swap(CAS)的定義如下:

```
int compare_and_swap(int *value, int expected, int new_value){
    int temp = *value;
    if (*value == expected)
        *value = new_value;
    return temp
}
```

CAS 指令功能如下:

一律回傳舊 value 值，且如果 value == expected 成立，則 value 設成 new_value。並保證此指令是 atomically execution(不可分割的執行)。

Bounded-waiting with compare-and-swap 解法如下:

1. 共享變數宣告: `int lock=0; Boolean waiting[0...n-1]`; 初值皆為 false

2. Code:

```
while(true){
    waiting[i] = true;
    key=1;
    while(waiting[i] && key==1)
        key= compare_and_swap(&lock,0,1);
    waiting[i] = false;
    /* critical section*/
    j=(i+1)%n;
    while((j!=i) && ! waiting[j])
        j=(i+1)%n;
    if(j==1)
        lock=0;
    else
        waiting[j]=false;
    /* remainder section*/
}
```

此 Code 有滿足 CS Design 中的 Bounded-waiting。假設 P_0 到 P_{n-1} 這 n 個 Process 皆想進入 CS，即 `waiting[0...(n-1)]` 皆 True 令 P_0 是第一個執行 CAS 而率先進入 CS 者，其他 $P_1, P_2, P_3, \dots, P_{n-1}$ 都在等待中當 P_0 離開 CS 後，必會將 P_1 之 `waiting[1]` 設 false，讓 P_1 進入 (lock 仍為 1)，依此類推， $P_1, P_2, P_3, \dots, P_{n-1}$ 會依序 (FCFS) 進入 CS，所以 P_1 至多等 $n-1$ 次後即可進入 CS (No starvation)。

8. ■請解釋何為“busy waiting”。

A: “busy waiting”的定義如下: 當有 Process 在 CS 時，其他的 Process 嘗試進入 CS 會被持續的卡在迴圈(loop)內，也稱 Spin Lock。

9. ■請解釋使用“busy waiting”的好處與壞處。

A: “busy waiting”的好處(優點): 若 Process 卡在 loop 的時間很短 (i.e. 小於 2 個 Context Switch Time) 則 busy waiting 有利。

“busy waiting”的壞處(缺點): 等待中的好處(優點): 會與其他 Process 競爭 CPU，將搶到的 CPU Time 用在毫無實際進展的迴圈測試上，因此若 Process 要等待長時間才能跳出迴圈，則此舉非常浪費 CPU Time。

10. ■請解釋何為 Monitor?

A: Monitor 是一個用來解決同步問題的高階結構，是一種 ADT (Abstract Data Type)，Monitor 之定義主要有 3 個組成:

1. 共享變數宣告區
2. 一個 local functions(or procedures)的集合，供外界呼叫使用
3. Initialization(初始區)

Monitor 內之共享資料只能被 Monitor 內的 functions 直接存取，外界不可直接存取，只能透過呼叫 Monitor 提供之 functions 來間接使用(類似 private)。

特性: Monitor 本身保障 mutual exclusion，即在任何時間點最多只允許 1 個 Process 呼叫 Monitor 內的某個 function 執行，不會有多個 Process 同時呼叫 Monitor 內的 functions 執行(保障共享資料不會 Race Condition)，所以 Monitor 相較於 Semaphore 更讓 Programmer 易於使用。

Condition Type : Condition 變數型別是用在 Monitor 鐘，提供給 Programmer 解決同步問題使用，令 x 是 Condition Type 變數，在 x 上提供 2 個操作:

1. x.wait:執行此運作的 Process 會被 Blocked，且置入 x 變數所屬的 Waiting Queue 中(預設是 FCFS Queue)。
2. x.signal:如果先前有 Process 在 Waiting Queue 中，則此操作會從此 Waiting Queue 中移出一個 Process 並恢復(resume)其執行，否則無任何作用。

Condition Wait:通常 Condition 變數所附屬的 Waiting Queue 預設是 FCFS Queue，但有時候我們需要的是 Priority Queue。令 x 是 Condition Type 變數，x.wait(c)操作就可以將執行此操作的 Process 置入 x 變數所屬的 Priority Queue 中，其中 c 代表其優先權 Level。

Chapter 7

1. ■在 Bounded Buffer Problem 中，有三個 Semaphore，分別是 mutex，full 和 empty，請解釋 mutex 的用途。

A: Semaphore mutex 的用途是用來對 Buffer 變數做互斥控制，防止 Race condition。

2. ■在 reader-writer problem 中，有一般整數變數 read count，請解釋該變數的用途。

A: 在 reader-writer problem 中，一般整數變數 read count 是用來統計目前的 reader 個數，且因可能有多個 reader，所以該變數需要 Semaphore mutex 來做互斥控制。當 read count=1 時，代表目前此 reader process 是第一個 reader，所以要執行 wait(rw_mutex)，確認有無 writer 正在寫入，若有則需等待寫入工作完成，若無則可以順利執行讀檔工作(非同時間的第一個 reader 就不需執行 wait(rw_mutex)，因為也有其他 reader 在讀，不會 Race condition)。當 read count=0 時，代表沒有 reader，執行 Signal(wrt)，將檔案存取權釋放，有 writer 需要執行寫入工作就可以順利執行。

3. 請描述哲學家餐桌(Dining-Philosophers)的問題設定。
4. 在最初的嘗試中，使用了 wait(chopstick[i])和 wait(chopstick[(i+1)%5])依序檢查能否拿起左右的筷子，這種解法可能會有什麼問題？為什麼？
5. 在 Dining Philosopher 的解法中，test 函式在 if 條件時，會呼叫 slef[i].signal，為什麼需要這個操作？

Chapter 8

1. ■有 4 個條件同時成立時，會造成 Deadlock 的發生，請列出這四個條件的名稱。

A: Deadlock 成立的 4 個必要條件有:

1. Mutual Exclusion(互斥存取)
2. Hold & Wait(持有並等待)
3. No Preemption(不可搶先)
4. Circular Waiting(循環等待)
- 5.

2. ■請解釋 Deadlock 發生條件中的 Hold and wait 是什麼意思。

A: Deadlock 中的 Hold and wait 指的是 Process 持有部分資源，並且又在等待其他 Process 所持有的資源。

3. ■請解釋 Deadlock 發生條件中的 No preemption 是什麼意思。

A: Process 不可任意剝奪其他 Process 所持有的資源，必須等待對方釋放資源後，才有機會取得。

4. ■可以利用 resource-allocation graph 檢查系統的狀態有無 deadlock，請問判斷的標準是什麼？

A: 我們可以用 RAG 觀察並得到以下結論:

1. No Cycle 則 No Deadlock
2. 有 Cycle 不一定 Deadlock
3. 除非每一類型資源皆是 single-instance(單一數量)，則有 Cycle \Leftrightarrow 有 Deadlock

5. 請解釋 Deadlock Prevention 和 Deadlock Avoidance 的差別。

6. 請解釋何為 Safe State？

7. 在銀行家演算法(Banker's Algorithm)中，使用的變數有 Available/Max/Allocation/Need，請分別解釋四者的用途。

8. 請解釋 Safety Algorithm 的運作原理。

9. 請解釋 Resource-Allocation graph 和 wait-for graph 的差異。

10. 請解釋如何使用 Safety Algorithm 檢查有無 Deadlock。

Chapter 9

1. ■請分別解釋 base register 和 limit register 的用途。

A: base register 中紀錄的是 Process 在記憶體中的起始位置，相對位址加上 base register 中的值即是實體位址，在 MMU 中 base register 則稱為 relocation register。而 limit register 則是記錄 Process 的大小，用來判斷存取記憶體位址的範圍大小是否合法。

2. ■請解釋 Logical address 和 Physical address 的差別。

A: Logical address 是由 CPU 生成的，而 Physical address 則是實際去實體記憶體(RAM)存取的位置。而 Logical address 轉為 Physical address 需要 MMU 介入。

3. ■請解釋何為動態連結(Dynamic linking)。

A: 在 Execution Time，若 Module 被呼叫到，才將之載入，並與其他 Modules 進行 Linking 修正(外部符號之解決)，適用在 Library Linking(ex Dynamic Linking, Library, DLL)。

優點:節省不必要之 Linking time

缺點: Process 執行時間較久

4. ■請解釋何為連續記憶體配置(Contiguous Allocation)

A: 即 OS 在為 Process 配置記憶體空間時，必須配給 Process 一個連續的 Free Memory Space，而每個 Process 所占的 Memory Space 稱為一個 Partition。

5. ■請解釋記憶體管理中的洞(hole)的定義。

A: 在連續記憶體配置下，Memory 中會有一些 Free Memory Space(or Block)稱為 Hole。通常 OS 會用 Link-List 觀念管理 Holes，稱為 AV-List(Available-List)。

6. ■在選擇合適的記憶體位置配置一個程序的資料時，採用 best-fit 的好處是什麼？

A: best-fit 的作法是:

當 Process 大小為 n 時，check AV-List 中所有的 Holes，找出一個 Hole，滿足其 Hole size $\geq n$ 且 Hole size - n 的差值最小，將此區塊配置給 Process。

採用 best-fit 的好處是空間利用度會是最佳的

7. ■請解釋何為外部碎裂(external fragmentation)

A: 在 Contiguous Allocation 要求下，AV-List 中任何一個 Hole size 均小於 Process size，但這些 Hole size 總和卻大於等於 Process size。因為這些 Hole 並不連續，因此無法配置給 Process，造成空間閒置不用，Memory 利用度低。

8. ■請解釋何為記憶體 Compaction。

A: Compaction 是解決外部碎裂方法中的其中一種，做法是移動執行中的 Process，使原本非連續的 Hole 可以聚集形成一個夠大的連續 Free Memory Space。Compaction 的困難點在於不易製定最佳的 Compaction 策略、Process 必須是 Dynamic Binding 才可在執行期間移動。

9. ■請解釋何為分頁表(page table)

A: page table 是在 Page Memory Management 下，因為採取非連續性配置，OS 會替每個 Process 建立 page table，紀錄各個 Process 配置於哪個 Frame 之 Frame number(Frame 不一定要連續)，若 Process size = n 個 Pages，則他的 page table 就會有 n 個 entry。

10. 請解釋 Translation lookaside buffer 的用途。

11. 請解釋 Valid-invalid bit 的用途。

12. 在分頁表的實作中，其中一種是使用階層式分頁(hierarchical paging)，請解釋使用此方法的理由。

13. 在分頁表的實作中，其中一種是使用雜湊分頁表(Hashed page table)，請解釋使用此方法的理由。

14. 請解釋反向分頁表(Inverted Page Table)的運作原理。

15. 請解釋何為 swap 空間及使用 swap 的理由。