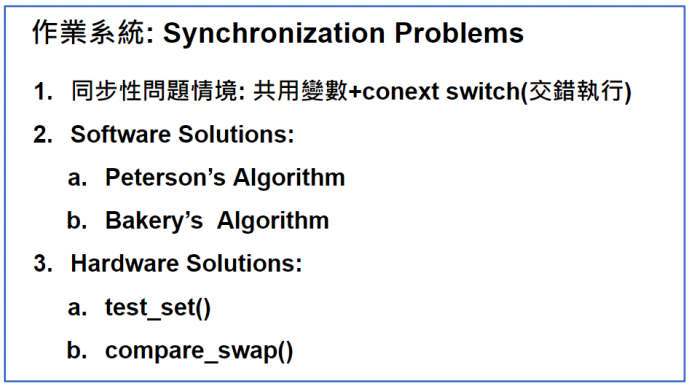
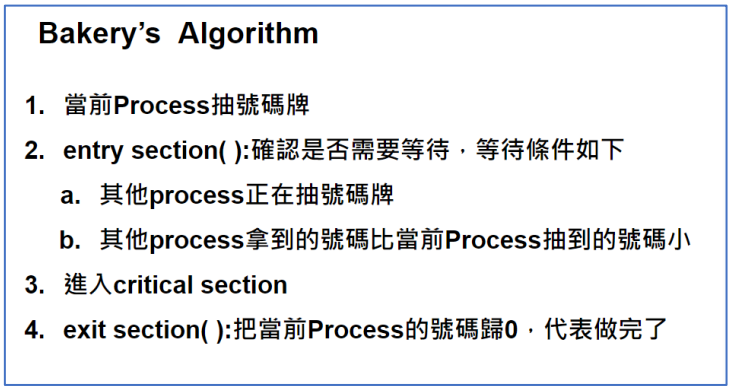
大家好，我今天的主題是想介紹 Synchronization 的解決方式，我們的教授教得非常清楚，所以大家肯定沒問題，今天想我介紹的算法有出現在課本上但沒有出現在投影片上的 Bakery algorithm，我主要閱讀的資料是 geekforgeek 以及網路上的教學影片



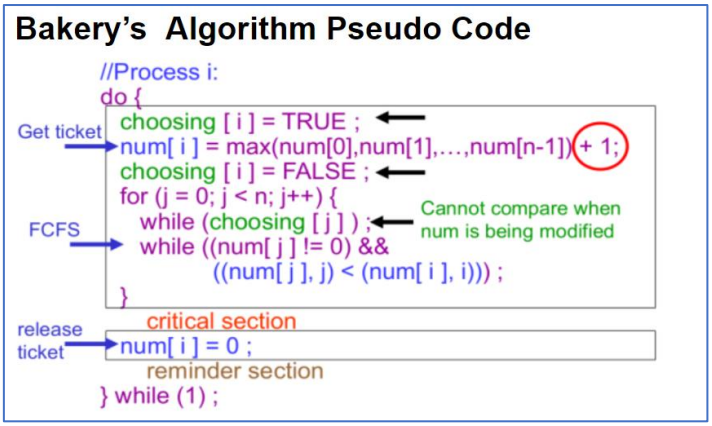
先來幫大家回憶一下會發生 synchronization 問題的情境，一定要是 process 或 thread 之間有共用變數，並且因為 context switch 的關係導致 access shared variable 的 code section 被 context switch 拆開後交錯執行，才會有 synchronization 問題，我們課本上舉的例子就是 procducer 和 consumer 的範例。

解決方法的部分，software 的方式最著名的就是教授教過的 Peterson's alogorithm，用於確保兩個 Process 不會同時進入 critical section，如果要確保 N 個 process 之間不會同時進入 critical section，可以參考今天的 Bakery algorithm

Hardware solution 就是在硬體端確保指令不會交錯執行，test_set 和 compare and swap 都是 hardware instruction.



Bakery algorithm 的重點就是抽號碼牌，讓同時想要取用 shared variable 的 process 抽號碼牌之後，排隊進入 critical section，從抽到號碼最小的 process 開始進入，最小的號碼是 1，process 如果做完 critical section，他的號碼牌就歸 0



1. 這邊的設定是 choose array 和 num array 都是共用變數，process 之間都是互相看的到
2. Choosing[i]=true 的部分就是標記當前的 process 正在抽號碼牌，如果這邊被 context switch 掉了，其他 process 也會知道 process[i]還在抽號碼牌
3. Num[i]=max(num[0],num[1],…num[j])+1 的部分就是在抽號碼牌，因為 num array 初始化成 0，所以號碼牌會從 1 開始往後抽
4. Choosing[i]=false 的部分就是標記當前的 process 抽完號碼牌了
5. 接下來就是條件判斷的部分，這邊在以下兩種情況會讓 Porcess[i]等待
   a. 第一種是如果當前有人在抽號碼牌就要等待，因為今天有可能 process[0]還沒抽完號碼牌就被 context swtich 到 process[1]去了，那當前的 num array 就是錯的，會額外導致 synchronization 的問題，所以要多個 while 等待
   b. 第二種是在其他 process 號碼牌不是 0 的情況下(只要 Process[j]的號碼牌為 0 就不用管他)，我們要確保當前的 Process，也就是 Process[i]所拿到的號碼牌是比其他所有 Process 都小，反之就要繼續等其他號碼更小的 Process 執行完，當前的 Process 才能執行。
6. 最後，從 critical section 出來後要把當前的 Process 的號碼牌歸 0，這樣他就能繼續去排隊抽號碼牌(但不一定會馬上拿到號碼，可能要等待，所以她的號碼會維持 0 一段時間)

```
void lock(int thread){
    choosing[thread] = 1; //Before getting the ticket number choose variable is set to be true
    int max_ticket = 0;
    for (int i = 0; i < THREAD_COUNT; ++i) { // Finding Maximum ticket value among current thread
        int ticket = tickets[i];
        max_ticket= ticket > max_ticket ? ticket : max_ticket;
    }
    tickets[thread] = max_ticket + 1;// Allotting a new ticket value as MAXIMUM + 1
    choosing[thread] = 0;
    for (int other = 0; other < THREAD_COUNT; ++other) {// Applying the bakery algorithm conditio
        while (choosing[other]) {
        }
        while(tickets[other]!=0&&(tickets[other]<tickets[thread]||
            (tickets[other]==tickets[thread]&&other<thread))){
        }
    }
}
void unlock(int thread){
    //MEMBAR;
    tickets[thread] = 0;
}
```

這邊就是 Code 的實作，這邊比較需要注意的，是讓當前 Process 通過 entry section 的條件，這一快比較 tricky 的當其他 Process 的號碼== Current Process 的號碼且其他 Process 的 ID 小於 current process 的 ID 的部分也需要等待，為何呢?

Bakery's Algorithm

1. num[i] 代表當前Process抽到的號碼牌
2. 為何entry section( )的條件要這樣設定?
   a. 如果其他process正在抽號碼牌可能會改動到num[ ]
   b. num[j]=1代表 porcess j已經做完不用管他
   c. num[i]=max(num[0],num[1],...)可能使不同Process拿到相同的號碼牌(一樣是synchronization的問題)

這邊一樣也是 synchronization 的問題
因為抽號碼的 max() function 實際上在執行的時候會被編譯成很多個 instructuon 執行，那可能執行到一半就會被 context switch 掉,但 num[i]是共用變數所以在 num array 可能會有多個 Process 拿到相同號碼，所以我們要特別注意這件事

接下來就是 demo 時間，我對一段會有 synchronization 問題的 code，分別測試有加 Bakery algorithm 所實現的互斥鎖以及沒有加 Bakery algorithm 實現的互斥鎖的測試。
這段 code 是分別讓八個 thread 去對一個共用變數++十萬次，那最後這個共用變數應該要是八十萬，但如果沒有用 Bakery algorithm 的互斥鎖，就會有同步性問題導致的錯誤產生，就不是八十萬。

使用Bakery's Algorithm實現mutex的lock和unlock
https://github.com/HsuSungTing/bakery_algo/blob/main/bakery.c

```
Thread 2 finished. Shared variable: 85981
Thread 0 finished. Shared variable: 81632
Thread 1 finished. Shared variable: 83447
Thread 3 finished. Shared variable: 94912
Thread 4 finished. Shared variable: 101376
Thread 5 finished. Shared variable: 131852
Thread 7 finished. Shared variable: 142340
Thread 6 finished. Shared variable: 183019
shared_variable: 183019:eason@LAPTOP-Q69P3F
```

使用Bakery's Algorithm實現mutex的lock和unlock
https://github.com/HsuSungTing/bakery_algo/blob/main/bakery.c

```
Thread 0 finished. Shared variable: 100000
Thread 1 finished. Shared variable: 200000
Thread 2 finished. Shared variable: 300000
Thread 3 finished. Shared variable: 400000
Thread 4 finished. Shared variable: 500000
Thread 5 finished. Shared variable: 600000
Thread 7 finished. Shared variable: 700000
Thread 6 finished. Shared variable: 800000
shared_variable: 800000:eason@LAPTOP-Q69P3F
```

點開 code，表達這裡是重點:

```c
// A simplified function to show the implementation
void* thread_body(void* arg){
    long thread = (long)arg;
    lock(thread);
    use_resource(thread);
    unlock(thread);
    return NULL;
}
```

然後就說接下來的 demo 會先執行有互斥鎖的部分，在執行把這兩行 Bakery 互斥鎖的 functuon 拿掉。明確證明 Bakeru 的做法真的是合理的