OS_110_Ch6

Background

- Concurrent access data 造成 data inconsistency
- 所以需要 ensure the orderly execution
- 當有 share memory 的時候，就會有可能有同步問題

Race Condition

- 結果的值會 depend on 最後一個修改值的人，則會造成中間的過程可能不正確

- 為了解決同步
  - 在 single-processor machine，可以用 disable interrupt or non-preemptive CPU scheduling
  - 大多用 critical section problem

Critical Section

- 為一個 protocol
- 問題：有多 processes 競爭 shared data
- 寫一個區塊，讓區塊只有一個 process 執行

Requirements

1. Mutual Exclusion：只有一個 process 可以在 CS 執行
2. Progress：要確保空的時候可以進去，且可以正常執行
3. Bounded Waiting：等待的時間是要有限的

Software Solution

- Peterson's Soloution for Two Processes 雙方同意
  - 有 flag 跟 token 可以確認對方要不要用

- Bakery Algorithm for N processes 像是抽號碼牌
  - FCFS
  - 有可能會抽到同樣的號碼牌，會比較 PID
  - 要有一個 choosing 的功能，判別是否再抽號碼牌
- Pthread Lock / Mutex Routines
- Condition Variables
  - wait() block
  - signal() wake up one thread
  - broadcast() wake up all thread
  - 代表可以告知某些事情發生，可以開始執行其他事
  - thread pool

Synchronization Hardware

- atomic instructions
  - atomic : 必須一次做完
  - as one uninterrupt unit
  - TestAndSet (var.), Swap (a,b)
- 方法是硬體燒進去的，所以很快
- 因為是硬體電路，所以不會有撞車的情況

TestAndSet

- 會回傳當下的 lock 狀況，且把呼叫的人鎖起來，下一個會收到已經鎖起來
- 離開的時候會解鎖

Swap

- 會將鎖頭交換給別人，給別人有機會執行
- 先 Call 的人就會拿到鎖

Semaphores

- 為一工具
- 是一個 Counter，對應到一個數字，數字代表有多少個單元可以用
  - if record == 1 : binary semaphore, mutex lock
  - if record > 1 : counting semaphore
- accessed only through 2 atomic : wait & signal

Spinlock Implementation

- busy waiting
- 如果執行時間很短，就可以用
- 長的話會消耗 CPU 資源

```c
wait(S){
        while (S <= 0);
        S--;
}
```

```c
signal(S){
        S++;
}
```

POSIX Semaphore

- 可以控制 record 的值
- 不一定 Pthread 才可以用

Non-busy waiting Implementation

- wait and signal → use system call block() and wakeup()，所以會慢
- 盡量為執行時間長，短的話，每次都會很慢
- must be executed atomically

Atomic Operation

Classical Problems of Synchronization

Monitors

Atomic Transactions