實驗七 More Synchronization – Semaphore & rwLock

9617145 資工 4C 許晏峻

1. 實驗目的

練習使用 semaphore and rwlock。

2. 步驟過程

2.1. semaphore

首先，在 global 宣告一個 sem_t 型態的 semaphore。

在 main 函式中，呼叫 sem_init 函式來將宣告的 semaphore 初始化，並且將 semaphore 設定同時有簽名簿數量(sheet_num)個資源可用。

在 student 函式中，在原本 mutex lock 的地方(也就是在 barrier 之後)，將呼叫 pthread_mutex_lock 改成呼叫 sem_wait；並且在原本 mutex unlock 的地方(也就是 student 函式最後幾行)，將呼叫 pthread_mutex_unlock 的地方，改成呼叫 sem_post。

以上完成 semaphore 的使用，但是為了保護 critical section (cqueue)，需要在每次呼叫 cqueue_pop_front 和 cqueue_push_back 的區段加上 mutex lock，以避免 race condition。

2.2. multi-threading even_odd_sorting

首先宣告 thread 需要傳遞參數的 struct，其中包括 int tid, n, start, end 和 int *a；n 代表總數(也就是 main.c 中的 student_num)，start 和 end 則代表在 Odd_even_iter 函式中迴圈使用的起始值和邊界值，a 則代表指向要 sorting 陣列的指標。

在 Odd_even_sort 函式中，初始化一個 global 的 pthread barrier，將 barrier 的 count 數設成 n/2，然後將原本的做 n 次 iteration 區段刪掉，加上迴圈來建立 n/2 條 thread，並且設定正確的參數(int tid, n, start, end 和 int *a)，傳遞進入 RunPhase 函式；最後，再依序呼叫 pthread_join 等待所有 thread 完成。

在 RunPhase 函式中，是原本在 Odd_even_sort 函式中的 n 次 iteration 區段(上面刪掉的部分)；不同的是在每次 iteration 開始時會先做 pthread_barrier_wait 以確保上一次 iteration 的 thread 都已經完成，接者呼叫 Odd_even_iter。

在 Odd_even_iter 函式中，只需要改變傳遞進來的參數(int a[], int start, int end, int phase)，把迴圈的起始值設成 start，將邊界值設成 end 即可。

2.3. rwlock

將原本 semaphore 版本的程式碼中的 pthread_mutex_lock 改成 pthread_rwlock_wrlock，並且將最後計算 signup student 區段加上 pthread_rwlock_rdlock 即可。

3. 數據結果

以下測試環境為：

- 實體機器
- CPU：AMD Phenom(tm) II X6 1055T Processor
- Core：6
- Memory：8G
- OS：Ubuntu 10.04.02 TLS x86_64

```
signup_count:1000
Check Ok!!!
4.390u 40.640s 0:27.37 164.5%   0+0k 0+0io 0pf+0w
```

圖 1 執行./main 1000 2 (執行時間約 27.37 秒)

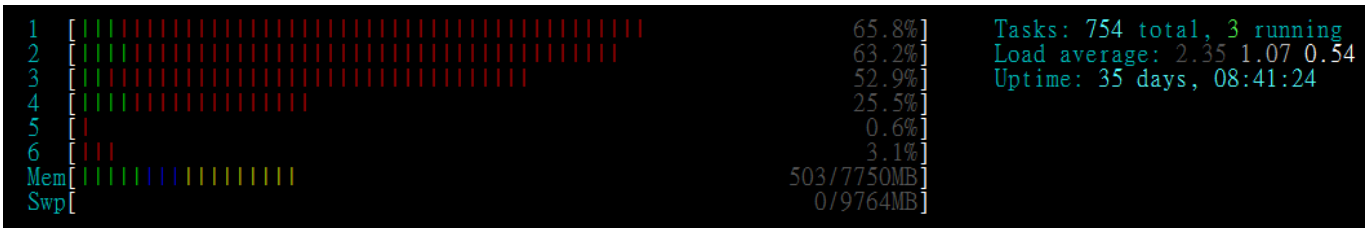

圖 2 執行./main 1000 2 下的 htop 指令

以上可看出在只有 2 個簽名簿時，CPU 資源沒有被耗盡，因為大多數 thread 都在等待 semaphore post 出來釋放資源。

```
signup_count:1000
Check Ok!!!
5.710u 69.070s 0:20.00 373.9%   0+0k 0+0io 0pf+0w
```

圖 3 執行./main 1000 32 (執行時間約 20.00 秒)

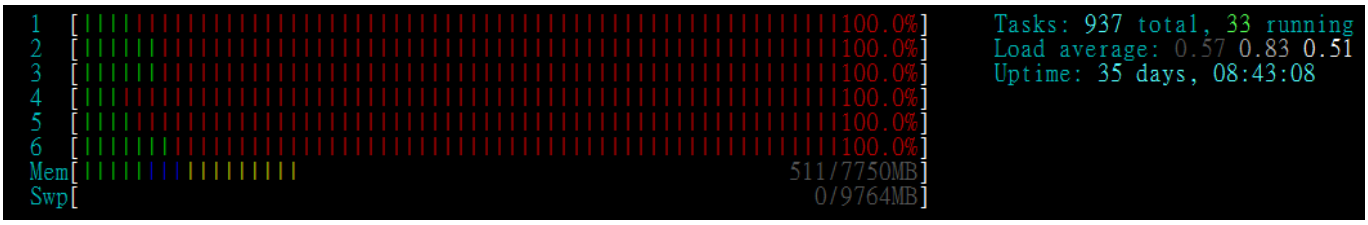

圖 4 執行./main 1000 32 下的 htop 指令

以上可看到執行時間有縮短為 20 秒左右，並且從 htop 指令可看到 CPU 資源已經被完全使用到了，因為 32 個簽名簿同時可讓 32 條 thread 來工作，只有在某些 critical section 需要控制流程為 sequential 執行順序。

4. 結論心得

這次實驗中，學習到了 semaphor 和 rwlock 的使用方式，並且在課堂中也瞭解了從一開始教的 busy waiting lock、mutex lock、conditional variable、barrier 到今天實驗的 semaphore 和 rwlock，這

些流程控制的技術之間有什麼異同、什麼時機該使用什麼…等。

而在 odd even sorting 部分，則讓我又有機會再次將期中考寫得不夠完美的 multi-thread 程式有機會寫得更好，加上了考試時沒想到的 barrier 觀念，讓 thread 可以只建立一次就好，而不需要每次 iteration 都重新建立，理論上應該可以讓 CPU 的 loading 減輕不少。

在實驗課中，也從助教的現場 demo 知道了 htop 這項好用的 Unix 指令，比起原本系統內建的 top 指令，整個 monitor 畫面又更豐富了點。