OS Spring 2017, Project 2 Report

Team28 : B04902077 江緯璿 B04902079 甯芝蓓

Part 1: Invoke FIFO Scheduler

Ref: http://morris821028.github.io/2016/01/21/realtime-system-using-scheduler/

Compilation: `g++ sched_test.cpp -lpthread -lrt -o sched_test`

Execution:

1. Default: `./sched_test`
2. FIFO: `sudo ./sched_test SCHED_FIFO`

這次運用 pthread 進行 thread 的操作。首先利用 sched_setaffinity()將所有 thread 分配到 CPU0 進行運算，以確定不會因平行處理 thread 導致無法看出 FIFO Scheduler 的特性。接著使用 sched_setscheduler()來定下 scheduler，其中因為 FIFO 是 Real-Time Scheduler，故需要 root 權限執行才能成功 set scheduler。接下來利用 clock_gettime()取得 thread 的執行時間來進行 busy 0.5 sec 的操作，使用這個函式必須在編譯時加上-lrt 的參數。

執行後，default 會使得兩個 thread 接近交互運行，如 1 0 0 1 0 1 這種執行順序。FIFO 則穩定為 0 0 0 1 1 1，充分展現 FIFO Scheduler 的 Policy and Algorithm.

Part 2: Weighted Round Robin Scheduler

原本對這一部份有些摸不著頭緒，在助教公佈 hint 之後才找到了方向。花最多時間的部分反而是 trace code 以及尋找各種函式的正確用法。與 scheduler 有所牽涉的資料結構十分龐大複雜，其中不但有許多相似的命名（ex. rq, rq->weighted_rr_rq 所指不同），指標或物件本身也是另一個容易使人混淆的原因。後來決定在紙上把與此次作業比較相關的資料結構畫出來，便一目瞭然了。

另外還遇到一個滿有趣的問題：因為不清楚 list_first_entry(ptr, type, member) 的用法，去找了 reference，發現它只是被 define 到 list_entry((ptr)->next, type, member)，還是看不懂 XD 再查了 list_entry()，又發現他只是被 define 到另一個 function 了… 繞了一大圈，後來是在 sched_weighted_rr.c 的其他 function 中找到相同的函式，才順利推敲出他的用法。

Part 3: Bonus: OpenMP Parallelization of Part 1

Compilation: `g++ sched_test_OpenMP.cpp -fopenmp -lrt -o sched_test_OpenMP`

Execution:

1. Default: `./sched_test_OpenMP`

2. FIFO: `sudo ./sched_test_OpenMP SCHED_FIFO`

上課時曾提過 OpenMP 可以進行 Implicit Threading. 故試圖利用這個技術進行 2 個 thread 的模擬。首先，`omp_set_num_threads()` 可以設定總共有幾個 thread，而 set CPU affinity 的方法則與原本相同。最大相異處是從程式碼 Line 50 開始，利用`#pragma omp parallel for schedule(dynamic,1)` 平行化處理下一行 I = 0,1 的迴圈，因為 schedule 設成 dynamic，沒事做的 thread 會從 task_queue 中取一個工作來做，也就是一個 for 迴圈的 iterator，故確保 I = 0 一定會比 I = 1 先被執行，如此一來即可驗證 FIFO Scheduler 的正確性。