OS HW3

資安所 宋哲頤 313560005

For the program source part (50%)

```
sudo ./sched_test.sh ./sched_demo ./sched_demo_313560005
Running testcase 0 : ./sched_demo -n 1 -t 0.5 -s NORMAL -p -1
Result: Success!
Running testcase 1 : ./sched_demo -n 2 -t 0.5 -s FIFO,FIFO -p 10,20
Result: Success!
Running testcase 2 : ./sched_demo -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30
Result: Success!
andy@andy-VirtualBox:~/HW2$
```

1.Describe how you implemented the program in detail. (10%)

參數解析：

透過 getopt 解析 command-line 參數來設置 thread 數量、每個 thread 的等待時間、排程策略（如 NORMAL 或 FIFO）以及優先級。

Thread Attribute Configuration：

使用 *pthread_attr_t* 結構來設置 thread 屬性。*sched_param param* 指定各個 thread 的排程優先級。*pthread_t thread* 儲存每個線程的 ID。*thread_info* 每個線程的基本資訊，例如線程 ID、排程策略和等待時間。

cpu_set_t 用於指定線程可以在哪些 CPU 上執行。*CPU_ZERO* 將所有 CPU 從集合中移除。*CPU_SET* 指定 threads 只允許在 CPU 0 上執行。*sched_setaffinity* 設置線程的 CPU 親和性。

```c
/* 2. Create <num_threads> worker thread_info */
pthread_attr_t attr[num_threads];       // thread attributes
struct sched_param param[num_threads];  // thread parameters
pthread_t thread[num_threads];          // thread identifiers
thread_info_t thread_info[num_threads]; // thread information

/* 3. Set CPU affinity */
cpu_set_t cpuset;
CPU_ZERO(&cpuset);
CPU_SET(0, &cpuset);
sched_setaffinity(0, sizeof(cpuset), &cpuset);
```

Thread Synchronization：

使用 pthread_barrier 確保所有 thread 在屏障處等待，直到所有 thread 準備好，再同時開始執行。

Set the attributes to each thread：

初始化 thread 屬性：根據排程策略來初始化每個 thread 的屬性。如果策略為 "FIFO"，則設置其屬性為 SCHED_FIFO，並明確指定線程不繼承父線程的排程設定（PTHREAD_EXPLICIT_SCHED）。

設置排程策略與優先級：對於 FIFO 線程，使用 pthread_attr_setschedpolicy 設置排程策略，並透過 pthread_attr_setschedparam 設置其優先級。優先級取自 priorities 陣列，讓不同的 FIFO thread 具備不同的優先級，以影響其排程順序。

Create thread：利用 pthread_create 函式創建 thread，並將設置好的屬性與 thread_info 傳入。

```c
/* 4. Set the attributes to each thread */

for (int i = 0; i < num_threads; i++) {
    thread_info[i].thread_id = i;
    thread_info[i].time_wait = time_wait;

    if (strcmp(policies[i], "FIFO") == 0) {
        thread_info[i].sched_policy = SCHE_FIFO;
        pthread_attr_init(&attr[i]);
        pthread_attr_setinheritsched(&attr[i], PTHREAD_EXPLICIT_SCHED);

        // set the scheduling policy - FIFO
        if (pthread_attr_setschedpolicy(&attr[i], SCHE_FIFO) != 0) {
            perror("Error: pthread_attr_setschedpolicy");
            exit(EXIT_FAILURE);
        }

        param[i].sched_priority = priorities[i]; // set priority

        // set the scheduling paramters
        if (pthread_attr_setschedparam(&attr[i], &param[i]) != 0) {
            perror("Error: pthread_attr_setschedparam");
            exit(EXIT_FAILURE);
        }

        // create the thread
        if (pthread_create(&thread[i], &attr[i], thread_func,
                           &thread_info[i]) != 0) {
            perror("Error: pthread_create (FIFO)");
            exit(EXIT_FAILURE);
```

Thread 功能實現：

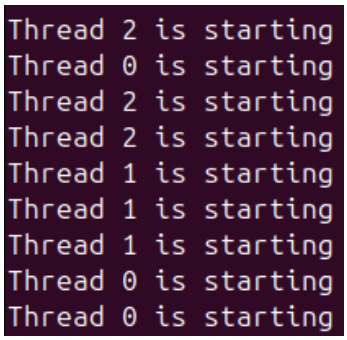
每個 thread 進行三 busy-wait 操作。使用 clock_gettime() 取得當前時間，持續比較直到達到指定的忙等待時間後退出迴圈。每次迴圈結束後，使用 sched_yield() 讓出 CPU 以便其他 thread 能夠運行。

程式結束：

主程式等待所有 threads 完成後，解除 barrier 並退出。

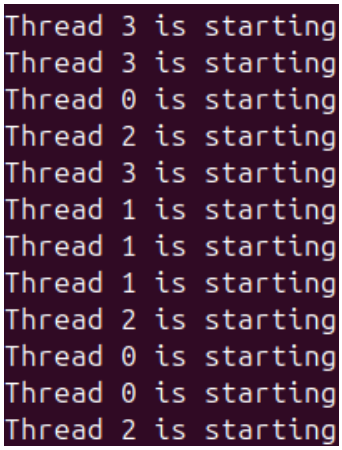
2.Describe the results of sudo ./sched_demo -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30 and what causes that. (10%)

第三個 thread（FIFO、優先級 30）會先完成，接著是第二個 thread（FIFO、優先級 10），最後是第一個 thread（NORMAL）。



3.Describe the results of sudo ./sched_demo -n 4 -t 0.5 -s NORMAL,FIFO,NORMAL,FIFO -p -1,10,-1,30, and what causes that. (10%)

FIFO 策略的第四個 thread（優先級 30）優先級最高，因此預期會先執行並保持控制，直到該 thread 完成或讓出 CPU。
第二個 FIFO thread（優先級 10）會在第四個 thread 完成後才執行。
NORMAL 策略的 thread（第一和第三個）只有在 FIFO thread 不執行時才會被調度，因此會最後完成。



4.Describe how did you implement n-second-busy-waiting? (10%)

使用 clock_gettime 取得 thread 開始執行時的時間 begin_time，並將其轉換為毫秒。

進入 while 迴圈，不斷使用 clock_gettime 取得當前時間 current_time，計算與 begin_time 的時間差。

當時間差達到指定的忙等待時間（wait_time_ms）後，迴圈結束。

```c
/* 2. Do the task */
int count = 0;
while (count < 3) {
    printf("Thread %d is starting\n", (int)thread_info->thread_id);

    struct timespec begin_time, current_time;
    clock_gettime(CLOCK_THREAD_CPUTIME_ID, &begin_time);
    int begin_ms = begin_time.tv_sec * 1000 + begin_time.tv_nsec / 1000000;

    int elapsed_ms = 0;
    do {
        clock_gettime(CLOCK_THREAD_CPUTIME_ID, &current_time);
        int current_ms = current_time.tv_sec * 1000 + current_time.tv_nsec / 1000000;
        elapsed_ms = current_ms - begin_ms;
    } while (elapsed_ms < wait_time_ms);

    sched_yield();
    count++;
}
```

5.What does the kernel.sched_rt_runtime_us effect? If this setting is changed, what will happen?(10%)

Effect：kernel.sched_rt_runtime_us 會限制 real-time threads 在每秒內允許的運行時間。超過這個時間後，系統會強制讓出 CPU 以便其他 non-real-time threads 能夠運行，防止 real-time threads 獨佔 CPU。

更改的影響：

增大 kernel.sched_rt_runtime_us 會讓 real-time threads 的允許執行時間更長，可能導致其他 non-real-time threads（例如 NORMAL threads）更少的執行機會。

減少 kernel.sched_rt_runtime_us 會讓 real-time threads 更頻繁地被迫讓出 CPU，使得其他 threads 有更多的執行機會，但可能影響 FIFO 和 RR threads 的 responsiveness。