OS assignment2

1. Describe how you implemented the program in detail

A. Parse program arguments

我首先利用 getopt()這個 function，來將 terminal 傳入的 argument 讀取進來，接著我再利用 atoi()、atof()，來將傳入的字串轉換成 integer(number of threads)還有 float(timewait)，然後我再利用 strtok()，這個功能是將字串有指定符號的地方切割開來，這邊是指定“,”，用來將輸入進來的 priority 以及排程方式切割開來，然後我會將各個 thread 的 priority 以及排程方式分別存成兩個長度為 thread 數量的陣列。

B. Set CPU affinity

我這邊用 pthread_setaffinity_np()來設定讓 thread 都在同一個 CPU 上跑，我這裡設定在 index 1 的 CPU 跑，達到競爭 CPU 的目的。

C. Set the attributes to each thread &

Create <num_threads> worker threads

```
/* 2. Create <num_threads> worker threads */
thread_info_t my_threads_data[thread_num];
pthread_t my_threads[thread_num];
pthread_attr_t attr[thread_num];
```

thread_info_t 是用來存取各 thread 的資訊的一個 struct，
pthread_t 的作用有點像是代表 thread 的 id
pthread_attr_t 是用來存取各 thread 的屬性的，像是排程方式以及 priority 等等。

將需要的陣列以及其資料型態宣告完畢後就可以創建 thread 了，我這邊是利用 for 迴圈並且用 pthread_create()將 thread 分別建立起來，除此之外，為了避免讓先建立好的執行續先執行，需要使用 pthread_barrier_init()來宣告需要等待多少個執行緒完成後再一起放行(通常設定 n 個執行緒+1 個)，然後用 pthread_barrier_wait()來等待執行緒，等到 pthread_barrier_wait()被執行了我們指定的數量之

後，在等待的執行緒就會被放行，一起競爭 CPU。

D. thread_func

thread_func 裡面包含了顯示第幾個 thread 正在 running 以及 busy wait 的操作，這邊我是將所有的 thread_info_t 裡面所包含的資料傳入(包括 priority、thread id、time wait)，這樣就可以從 time wait 知道需要 busy wait 多久，然後我再利用 clock_gettime()來操作 busy wait。

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

```
(base) james@james-Victus-by-HP-Laptop-16-d0xxx:~/NYCU/111fall/OS/LAB2$ sudo ./sched_demo -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30
Thread 2 is running
Thread 2 is running
Thread 2 is running
Thread 1 is running
Thread 1 is running
Thread 1 is running
Thread 0 is running
Thread 0 is running
Thread 0 is running
```

因為 thread 2 的排程是 FIFO，且優先度最高，所以會優先執行完畢，接著是 thread1，因為 thread1 的排程方式也是 FIFO，會優先於 NORMAL 執行，最後才會是排程方式 NORMAL 的 thread0 執行。

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

```
(base) james@james-Victus-by-HP-Laptop-16-d0xxx:~/NYCU/111fall/OS/LAB2$ sudo ./sched_demo -n 4 -t 0.5 -s NORMAL,FIFO,NORMAL,FIFO -p -1,10,-1,30
Thread 3 is running
Thread 3 is running
Thread 3 is running
Thread 1 is running
Thread 1 is running
Thread 1 is running
Thread 2 is running
Thread 0 is running
Thread 2 is running
Thread 0 is running
Thread 2 is running
Thread 0 is running
```

因為 thread3 的排程方式是 FIFO 且優先度最高，所以會優先執行完畢，接著是 thread1，因為 thread1 的排程方式也是 FIFO，但是優先度比較低，所以會讓 thread3 先執行完畢才執行，最後 thread2 跟 thread0 會交錯執行的原因是因為他們的排程方式一樣，且沒有優先度問題，執行一次之後不會被優先放回去執行，所以才會是交互執行。

4. Describe how did you implement n-second-busy-waiting

我再 thread_func 裡面使用了 clock_gettime()這個 function，我先獲得一個開始時間，再利用迴圈不斷重新獲得新的時間，直到獲得的時間扣掉開始時間等於我傳入的 time wait 時間後，再跳出迴圈並且用 sched_yield() 將 CPU 讓出來。除此之外需要下一個指令來確保 CPU 在執行時不會被 preempted，就是 sysctl -w kernel.sched_rt_runtime_us=1000000。