

# OS Project #2

B03901116 王彥稀, B03901018 楊程皓

May 8, 2018

## 1 Part 1

Following the hint & using the code in part 2 as the template. First, set cpu affinity. Then, Invoke the FIFO scheduler. In the end, create two threads with descending priority so that they would follow the right execute order. To be busy for 0.5 second, I calculate the passed time in a while loop until it has been through 0.5 second. The details are as following:

1. Because this program runs multiple threads, if we don't set CPU affinity then the threads may be run on different hardware core. Use `sched_setaffinity()` to set cpu affinity.

```
cpu_set_t mask;
CPU_ZERO(&mask);
CPU_SET(0, &mask);
sched_setaffinity(0, sizeof(mask), &mask);
```

2. Use `sched_setscheduler()` to tell current thread use certain scheduling policy. Because the scope of this function is only on the thread that calls it, there are typically two methods to achieve our goal.

The first one, use pthread barrier to synchronize every thread.

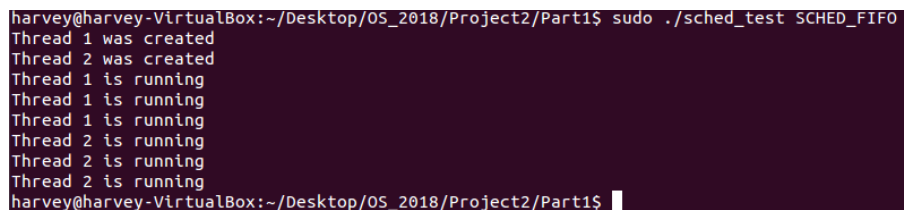
```
pthread_barrier_t barrier; // global variable
thread_function() {
    sched_setscheduler(0, SCHED_FIFO, &param);
    pthread_barrier_wait(&barrier);
    // run
}
main() {
    pthread_barrier_init(&barrier, NULL, 2);
    // create threads here
}
```

The second one, use inherited schedule. Set priority and inherited schedule before creating threads. I make the new threads' priorities lower than 99, which is the priority of the main thread, so that the new threads will be executed after all the threads are created.

```
// in main function
pthread_attr_setinheritsched(attr, PTHREAD_EXPLICIT_SCHED);
pthread_attr_setschedpolicy(attr, SCHED_FIFO);
for(int i = 0; i < 2; ++i)
    param.sched_priority = 98 - i;
pthread_attr_setschedparam(attr, param);
pthread_create(&tid[i], attr, thread_function, args);
```

### 3. Busy waiting for 0.5 second.

```
clock_t start;
while((clock() - start) / (double)(CLOCKS_PER_SEC) < 0.5);
```



```
harvey@harvey-VirtualBox:~/Desktop/OS_2018/Project2/Part1$ sudo ./sched_test SCHED_FIFO
Thread 1 was created
Thread 2 was created
Thread 1 is running
Thread 1 is running
Thread 1 is running
Thread 2 is running
Thread 2 is running
Thread 2 is running
harvey@harvey-VirtualBox:~/Desktop/OS_2018/Project2/Part1$
```

Figure 1: Part1 testing program, with correct executing order.

## 2 Part 2

1. `enqueue_task_weighted_rr`:  
First, add the task to the end of the queue with `list_add_tail`. Then, increase `nr_running` by 1 since one new task is in the queue.
2. `dequeue_task_weighted_rr`:  
First, update current `weighted_rr` with `update_curr_weighted_rr` since the current running task is being modified. Then, remove the head of the queue with `list_del`. In the end, decrease `nr_running` by 1 since the previous running task is no longer in the queue.
3. `yield_task_weighted_rr`:  
Just call the `requeue_task_weighted_rr` since the goal is the same as `requeue`: move the current running task to the end of the queue.
4. `task_tick_weighted_rr`:  
After updating the current task's runtime statistics, decrease `task_time_slice` by

1 and check if it is or was 0. If not, nothing to do and return immediately. If so, reset `task_time_slice`, set `task_need_resched`, and requeue the task with `requeue_task_weighted_rr`.

### 5. pick\_next\_task\_weighted\_rr:

First check the queue's size. If it's zero, return NULL. If not, choose the head of the queue as the next task to be executed, and set the task's `exec_start` as the current clock (time).

```
harvey@harvey-VirtualBox:~/Desktop/OS_2018/Project2/linux-2.6.32.60/test_weighted_rr$ ./test_weighted_rr weighted_rr 10 5 500000000  
sched_policy: 6, quantum: 10, num_threads: 5, buffer_size: 500000000  
abcdeabcdeabcabcabababababa  
harvey@harvey-VirtualBox:~/Desktop/OS_2018/Project2/linux-2.6.32.60/test_weighted_rr$
```

Figure 2: Part2 testing program, with correct weighted executed time slices.