OSPJ2_Team4_Report

Chun-Ju Wu B03202040, Meng-Jin Lin B04901009

May 10, 2018

1 Part I: Invoke FIFO Scheduler

In part I, we need to implement a code to invoke FIFO scheduler, and test if the program correctly invoke FIFO scheduler by producing two threads running simultaneously.

1.1 Implementation

1. First obtain the current using CPU by sched_getcpu(). Then set the cpu_set_t mask by CPU_ZERO and CPU set.
2. Check the argument input number, if there is no other argument, use default scheduler (do nothing). If the argument is "SCHED_FIFO", the use sched_setscheduler to set the scheduler to FIFO scheduler.
3. Use pthread_create to produce two threads executing function "thread_func", and print the message "Thread # is created".
4. The function "thread_func" will print the thread number of itself, and busy waiting for 0.5 seconds. The busy waiting is achieved by using clock.h. One timer is recorded first as start, and use the while loop to check if the time exceed 0.5 second by subtracting by the current time (another clock_t object). If exceed 0.5 second, then jump out the while. The function will be busy waiting for 0.5 seconds three times.
5. After all the thread functions finished, use thread_join to end up the thread.

Note that the function sleep() can not be used because sleep does not provide busy waiting. Instead, the process (thread) will give up the CPU and go to sleep.

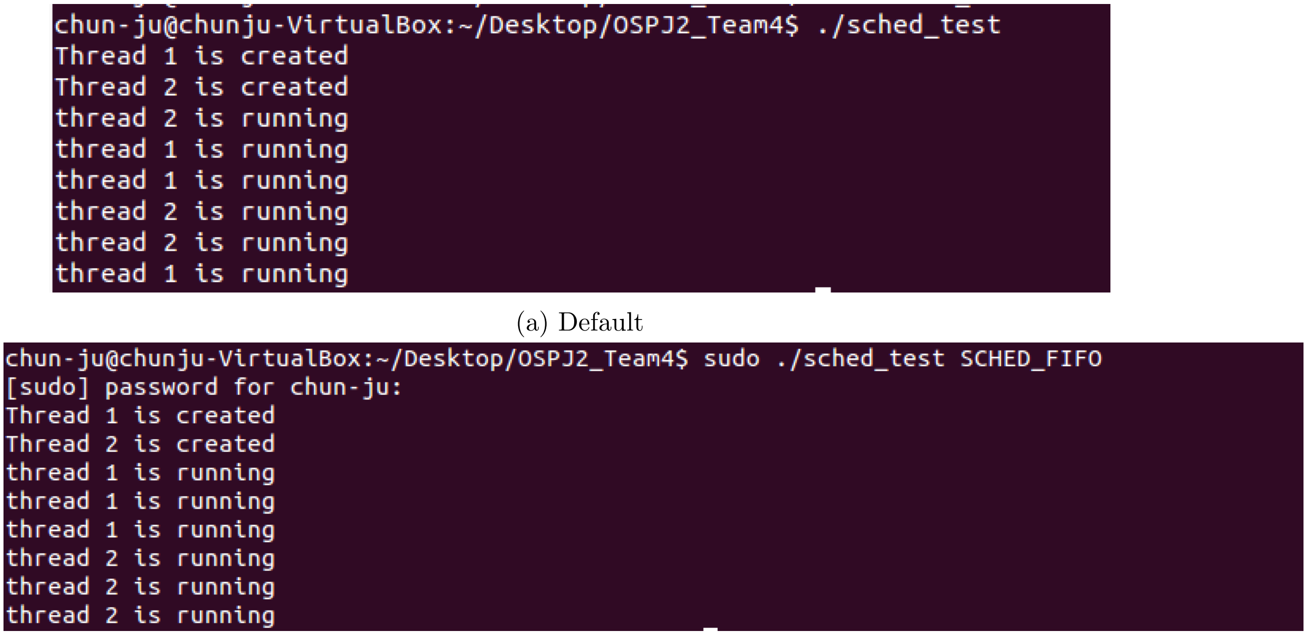
1.2 Result and Discussion

In Figure 1, the output of the Program using default scheduler and FIFO scheduler are shown.

In the default scheduler, thread 1 and 2 are alternating. In the FIFO scheduler, thread 1 is first executed, then thread 2 is executed. This is the expected result, because the default scheduler in linux is not FIFO. FIFO scheduler let the process be done first if it is first started. Thus the printed result follows the expectation.

2 Part II: Weighted Round Robin Scheduling

In part II, we need to finish some functions to implement weighted round robin scheduling and use the testing program(test_weighted_rr.c) to see if it is correct.



```
chun-ju@chunju-VirtualBox:~/Desktop/OSPJ2_Team4$ ./sched_test
Thread 1 is created
Thread 2 is created
thread 2 is running
thread 1 is running
thread 1 is running
thread 2 is running
thread 2 is running
thread 1 is running
```

(a) Default

```
chun-ju@chunju-VirtualBox:~/Desktop/OSPJ2_Team4$ sudo ./sched_test SCHED_FIFO
[sudo] password for chun-ju:
Thread 1 is created
Thread 2 is 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
```

(b) FIFO

Figure 1: The output of default scheduler and FIFO scheduler.

2.1 Implementation

1. enqueue_task_weighted_rr():
   Use list_add_tail() to enqueue task_struct* p and decrease rq ->weighted_rr.nr_running by one.

2. dequeue_task_weighted_rr():
   Use list_del() to dequeue task_struct* p and increase rq ->weighted_rr.nr_running by one.

3. yield_task_weighted_rr():
   Assign rq ->curr ->weighted_time_slice to rq ->curr ->task_time_slice and use list_move_tail() to put the current task_struct* (rq->curr) to the end of the run list.

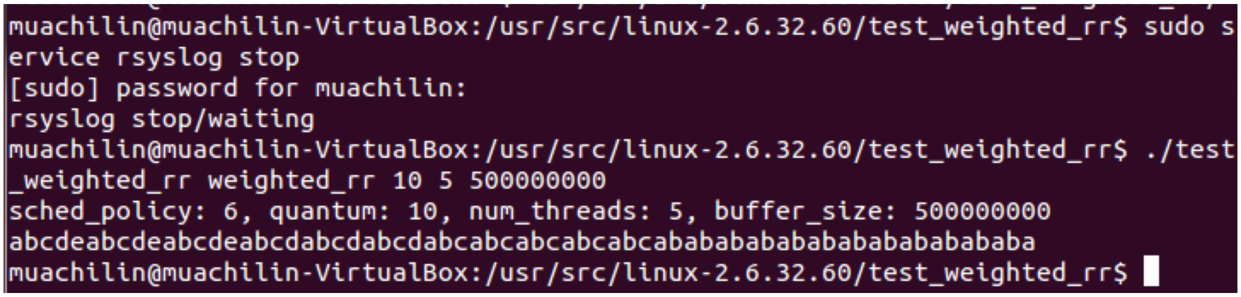
4. task_tick_weighted_rr():
   Minus p ->task_time_slice by one. If the value is zero, Assign rq ->curr ->weighted_time_slice to rq ->curr ->task_time_slice and check if curr ->weighted_rr_list_item.prev is equal to curr ->weighted_rr_list_item.next. If not, call set_tsk_need_resched() and requeue p.

5. pick_next_task_weighted_rr():
   If rq ->weighted_rr.nr_running is zero, return NULL because there is no task to run.

Otherwise, use list_first_entry() to obtain the first task in rq ->weighted_rr.queue and return it.

2.2 Result and Discussion

In Figure 2, the process and result of running test_weighted_rr.c are shown. The daemon process continues executing and thus consumes resources. When running the test program, if daemon process exists, it will not run correctly because the priority of the program is lower than daemon process's. Before executing the test program, use ”sudo service rsyslog stop” to stop daemon process. After executing the test program, a sequence of characters are printed. From the sequence, we can see that each character's finish time are different according to their weights. ”e”'s weight is the biggest so it finishes first. The finish sequence is therefore ”edcba”. On the other hand, from the other program files in the kernel directory, we found that in function task_tick_weighted_rr(), the task_struct* p is the same as rq ->curr in the execution of weighted round robin scheduling.



```
muachilin@muachilin-VirtualBox:/usr/src/linux-2.6.32.60/test_weighted_rr$ sudo s
ervice rsyslog stop
[sudo] password for muachilin:
rsyslog stop/waiting
muachilin@muachilin-VirtualBox:/usr/src/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
abcdeabcdeabcdeabcdabcdabcdabcabcabcabcabcababababababababababababa
muachilin@muachilin-VirtualBox:/usr/src/linux-2.6.32.60/test_weighted_rr$ 
```

(a) Result

Figure 2: The output of test program.