OS project 2

b0XXXXXX Goodman A
b0XXXXXX Goodman B

前言

本次project要求在linux kernel實作simple-round-robin的scheduler. 並實作weighted-round-robin作為bonus.

Implementation and results

1. Ubuntu 14.04 LTS 32bit

與Proj 1相同，想要在14.04 ubuntu上完成本次作業，然而在修改任何檔案之前編譯便出現了Compilation Error：

```
  CC      arch/x86/kernel/ptrace.o
arch/x86/kernel/ptrace.c:1472:17: error: conflicting types for ‘syscall_trace_enter’
 asmregparm long syscall_trace_enter(struct pt_regs *regs)
                 ^
In file included from /usr/src/linux-2.6.32.60/arch/x86/include/asm/vm86.h:130:0,
                 from /usr/src/linux-2.6.32.60/arch/x86/include/asm/processor.h:10,
                 from /usr/src/linux-2.6.32.60/arch/x86/include/asm/thread_info.h:22,
                 from include/linux/thread_info.h:56,
                 from include/linux/preempt.h:9,
                 from include/linux/spinlock.h:50,
                 from include/linux/seqlock.h:29,
                 from include/linux/time.h:8,
                 from include/linux/timex.h:56,
                 from include/linux/sched.h:58,
                 from arch/x86/kernel/ptrace.c:11:
```

看起來是編譯器認為asmregparm long這個type是不consistent的。我們猜想是gcc的版本不同而使預設的標準不同，因為改用ubuntu 12.04便能成功編譯。

2. Ubuntu 12.04 LTS 32bit - Simple Round Robin

a. syscall of set_quantum

i. 仿照Proj 1，定義set_quantum(int)的system call

```c
#include <linux/kernel.h>
#include <linux/linkage.h>
#include <linux/sched.h>
asmlinkage long sys_sched_simple_rr_setquantum(int q){
  simple_rr_time_slice = q;
  return 0;
}
```

ii. 藉由getquantum(void), setquantum(int), getquantum(void)可以確認set有設置成功

iii. 在trace kernel code時發現有透過SYSCALL_DEFINE來簡單定義syscall的方式，嘗試過兩者都能成功定義syscall

```c
SYSCALL_DEFINE1(sched_simple_rr_setquantum, int, q)
{
        simple_rr_time_slice = q;
        return 0;
}
```

b. Implementation of simple RR

i. enqueue / dequeue / yield_task / pick_next都是相當簡單的，這裏僅附上task_tick的截圖

```
189 static void task_tick_simple_rr(struct rq *rq, struct task_struct *p,int queued)
190 {
191         //+ first update the task's runtime statistics
192         update_curr_simple_rr(rq);
193         if (!task_has_simple_rr_policy(p))return;
194         if(simple_rr_time_slice == 0) return; //means no limit
195         p->task_time_slice--;
196         if(p->task_time_slice <= 0) {
197                 p->task_time_slice = simple_rr_time_slice;
198                 requeue_task_simple_rr(rq,p);
199                 set_tsk_need_resched(p);
200         }
201         //+ OS Proj 2: implement here
202         //...
203 }
```

ii. 需要注意的是如果 simple_rr_time_slice為0時代表RR的時間是無限制的，因此要特別判斷。

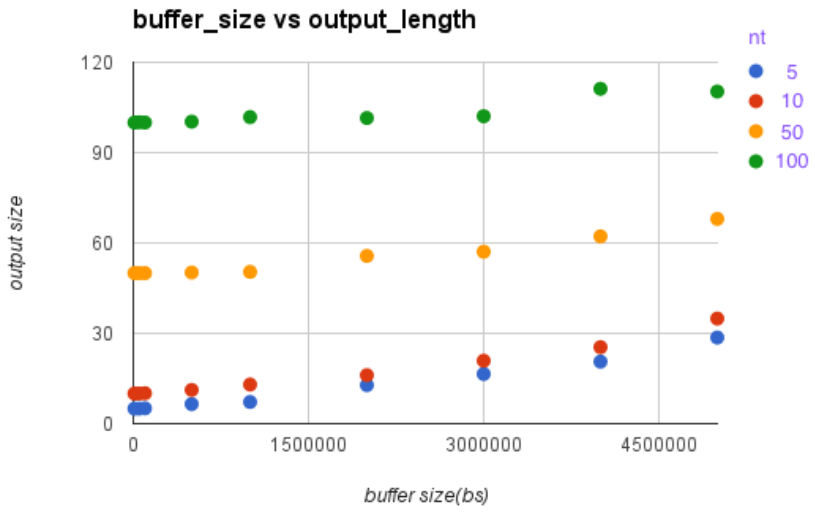
c. Results and Charts

i. ./test_simple_rr執行截圖

```
> ./test_simple_rr simple_rr 0 5 10
sched_policy: 6, quantum: 0, num_threads: 5, buffer_size: 10
abcde
```

```
> ./test_simple_rr simple_rr 1 5 10000000
sched_policy: 6, quantum: 1, num_threads: 5, buffer_size: 10000000
abcdeabcdeabcdeabcdeabcdeabcdeabcdeabcdeabcded
```

ii. 不同buffer size與threads#對平均輸出長度的影響(ts=1)



d. 討論

i. 在進行統計分析時，發現輸出的字串不是完全只有字母

```
david942j@david942j-VirtualBox:~/hw2$ ./ana.rb
"sched_policy: 6, quantum: 1, num_threads: 5, buffer_size: 1000000\nabcde\000abd\000a\n"
```

出現了null byte參雜在輸出中，因此使用bash是看不出來的！研究了test_simple_rr.c的寫法，應是race condition造成val_buf_pos被增加了而未寫入character。

ii. 從圖表中看出，當buffer_size小於100萬時，輸出長度與threads數相同。並且如預期的buffer size越長，輸出的重複次數就越多。

iii. 當#threads越大，輸出的重複次數變得越少，這應該是buffer size較小，使得nchars較小造成的。等比例提高buffer size至5000萬，的確就使nt=100的情形有7倍的重複率(平均輸出長度770)。

3. Weighted Round Robin as bonus

a. 我們實作了Weighted Round Robin，對於priority較大的具有較容易先完成的設計。

b. 我們實作方式很簡單，在tick中對於task_time_slice進行reset的時候，不要直接assign成simple_rr_time_slice，而是根據該task的priority做點加權，我們實作使用simple_rr_time_slice+priority*10，這樣一來具有較高priority的task就有較久時間被CPU連續的執行，因而容易較早完成。

```
189 static void task_tick_simple_rr(struct rq *rq, struct task_struct *p,int queued)
190 {
191         //+ first update the task's runtime statistics
192         update_curr_simple_rr(rq);
193         if (!task_has_simple_rr_policy(p))return;
194         if(simple_rr_time_slice == 0) return; //means no limit
195         p->task_time_slice--;
196         if(p->task_time_slice <= 0) {
197                 p->task_time_slice = simple_rr_time_slice+p->rt_priority*10;
198                 requeue_task_simple_rr(rq,p);
199                 set_tsk_need_resched(p);
200         }
201         //+ OS Proj 2: implement here
202         //...
203 }
```

c. rt_priority預設為0，因此如果不設置rt_priority的話就與原本的simple RR無異。

Obstacles

1. Simple Round Robin

我們在助教公布hint之前就已經完成scheduler的程式撰寫。一開始僅知道每個函式大約的用處，不確定實作上有哪些細節要考慮，例如enqueue是只要把task加入queue中呢，還是需要先判斷該task的policy是否正確。因此我們花了很多時間在trace kernel code，並利用printk進行debug。

a. 首先將pick_next設置成永遠回傳queue的頂端，不考慮time_slice，利用printk觀察enqueue和dequeue被呼叫的情形：

```
[   72.012221] calling enqueue, #=1 f0908000
[   72.012224] end calling enqueue, #=2
[   72.012273] calling enqueue, #=2 f090cc20
[   72.012275] end calling enqueue, #=3
[   72.012434] calling enqueue, #=3 f0908cb0
[   72.012436] end calling enqueue, #=4
[   72.012451] calling enqueue, #=4 f090f230
[   72.012452] end calling enqueue, #=5
[   72.012577] calling enqueue, #=5 f3388cb0
[   72.012579] end calling enqueue, #=6
[   72.012586] calling dequeue, #=6
```

觀察到enqueue的數量只有#threads+1，可以推論只有test_simple_rr中的task才會被加入queue中，因此enqueue與dequeue應當只要單純的進行list_head的操作即可。

b. 關於list_head的使用我們也是trace include/linux/list.h找到適合用的macro，而且我們寫好的與之後助教release的hint完全相同～

c. 誤會time_slice的用途，在我們一開始實作的scheduler中，將task_time_slice與sum_exec_runtime比較，如果sum_exec_runtime較大就認為此task必須被requeue。
這樣其實也的確實作了Round-Robin演算法，執行較久的thread就會被替換。不過這樣一來ts會在10000左右的數量級，因為我們觀察到大約每20000個clock才會呼叫一次tick，但這數值可能會與CPU時脈有關，並不適當作為與time_slice的比較。在助教release hint之後方才恍然大悟。

```
[   66.611718] curr:f36858d0 start=66611713478 sum_exec_runtime=12955 time_slice=20000
[   66.611721] ret :f3683f70 start=66610626628 sum_exec_runtime=35555 time_slice=20000
```

利用printk看出sum_exec_runtime的數量級。

2. Weighted Round Robin

我們的Weighted Round Robin需要設置rt_priority，為了測試就必須修改助教的test program。經過survey後找到使用pthread_attr_setschedparam這個library call可以產生attr包含priority的資訊。

然而直接使用pthread_attr_setschedparam時卻會有error產生，而且奇怪的是，只要priority設為0(default值)就會成功，非0就會失敗。經過code trace之後，發現sched.c中有一處

```
6584         if (rt_policy(policy) != (param->sched_priority != 0))
6585                 return -EINVAL;
```

也就是說，如果使用的schedule policy不是SCHED_FIFO或是SCHED_RR的話，priority就必須為0，否則會設置錯誤。

因此我們修改sched.c如下：

```
6585         if (!simple_rr_policy(policy) &&
6586                 rt_policy(policy) != (param->sched_priority != 0))
6587                 return -EINVAL;
```

這樣就可以繞過檢查。

然而修改之後，使用pthread_attr_setschedparam後整個OS會直接hang住！trace code之後還看不出來為何這樣修改會造成此情形，但往下看code中有goto語句，猜想可能是因而產生無窮迴圈了。

附上的kernel source是並未修改過sched.c的版本，避免造成助教也產生相同問題。

壯烈的51次編譯kernel!

```
Root device is (8, 1)
Setup is 14940 bytes (padded to 15360 bytes).
System is 3922 kB
CRC 6e5c8fc7
Kernel: arch/x86/boot/bzImage is ready  (#51)
```