COMP 530H
Fall 2014

Programming Assignment 4
Module for user-weighted round-robin scheduling
Date assigned: October 10, 2014
Date due: October 31, 2014

Implement a service module to provide user-weighted round-robin (UWRR) scheduling of processes. The module will essentially extend the Linux Real-Time scheduling class for processes that use the SCHED_RR policies. Each process should have an associated weight that determines its proportional share allocation of processor time allocated in units of a time quantum. For this assignment, assume that all processes calling your module are essentially CPU-bound.

Relative weights of processes can range between 1 and 20, inclusively. The actual time quantum for a process is determined by multiplying the weight by 10 milliseconds. Thus a weight of 10 corresponds to the normal Linux time quantum of 100 milliseconds. As an example, suppose that process A has a weight of 5 and process B has a weight of 20, both with the same priority. Then, on average process B should receive four times as much CPU time as process A over a fixed interval of time. Note that in the original Linux implementation, all SCHED_RR processes have the same fixed default time quantum so all processes with the same priority in this class receive approximately equal amounts of CPU time over a fixed interval of time.

Specifically, your module should implement one call:

- ***sched_uwrr*** (parameter is the relative weight for this process; returns 0 or error indication). The scheduling class for the calling process is to be set to SCHED_RR and the priority for the calling process is to be set to the lowest value (1) for Real-Time scheduling (which is still higher priority than any Linux non-Real-Time process). See the man pages for **sched_setscheduler** for details. Note that sched_setscheduler is a symbol exported in kernel/sched.c

Note: the process that makes this call to your module must run as root (or sudo) in order to request setting its priority value and setting its scheduling class to SCHED_RR.

Your extension of the Linux Real-Time scheduling class can be accomplished by making function call substitutions for two calls contained in the **sched_class** "object" pointed to by **task_struct** of the calling process (***after the process priority and SCHED_RR class have been successfully set by your module***). The necessary steps are:

- On the first call to your module, allocate a local copy of the structure of type sched_class (this structure will be shared by all processes that call sched_uwrr()). Initialize the structure by copying (with memcpy()) the sched_class structure from the calling process (current->sched_class) to your local copy. Replace the function pointers in .task_tick and .get_rr_interval with pointers to your local functions (that implement the weighted scheduling).
- On each call to sched_uwrr, save the pointer to the process original sched_class and replace it with a pointer to your local copy.

Study the kernel implementation of the functions pointed to by .task_tick and .get_rr_interval to determine how your replacement code should extend them to implement user weighted round robin scheduling. *Hint: don't try to make this harder than it needs to be!. You can arrange to call the original task_tick() function as part of your implementation.*

Your module should deal with any errors that can occur (but not be burdened with tests for errors

that cannot occur). The calling program should receive sufficient information so that it can determine if the call has been successful or has failed.

The important kernel files related to this assignment are include/linux/sched.h, kernel/sched.c, and kernel/sched_rt.c

A significant part of the grade for this assignment will be based on your experiments to evaluate the results of assigning different weights to a set of processes in the SCHED_RR class you implement. Use the printk function to log useful data for each process such as a timestamp when a quantum begins and ends. The kernel function ktime_get() can be used to obtain a nanosecond precision timestamp like this:

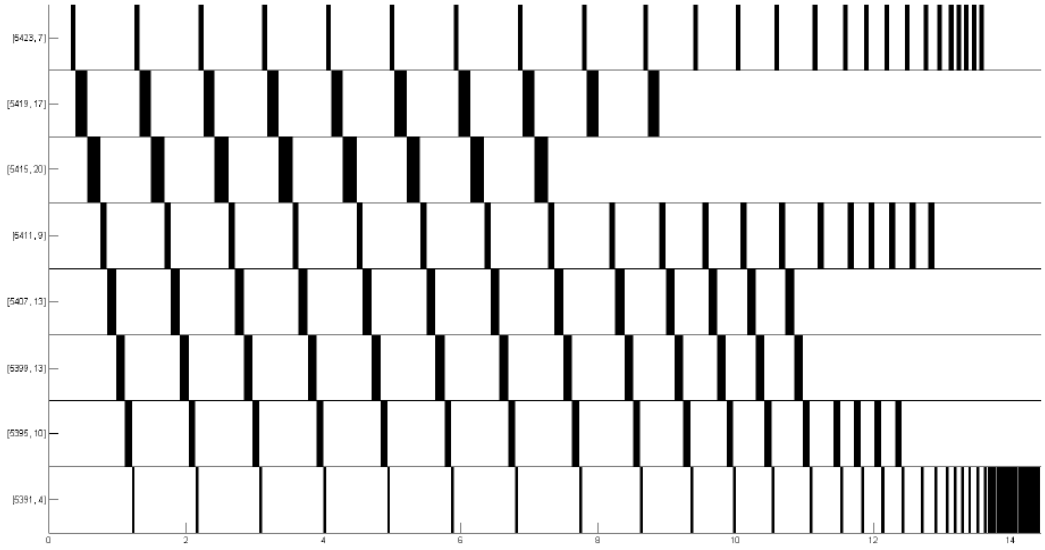
```
U64 time_stamp;
  ……
time_stamp = (u64) ktime_to_ns(ktime_get());
```

A suggested experiment is to have several processes that call sched_uwrr() with different weights specified and then block on the same named event using event_wait() from the previous assignment. Once all those processes are waiting, have a process that signals that named event non-exclusively so the blocked processes all become schedulable at approximately the same time. The weighted processes should all execute the same compute intensive loop and run for 20 or more seconds when running alone.

In addition to submitting your program as usual, also submit a short report (3-4 pages) that describes your experimental methods and gives the quantitative results you obtained from your experimental logs. Some examples of how you could display or plot your results are given below. These are intended only to stimulate your thinking about how you will report your results.

Submitting your program:

Follow the instructions from assignment 1 for submitting your programs and report.



Time →

PID	Priority	Allotted Time Slice (ms)	Total Ticks	CPU Allotments	Average Time Slice Length (ms)	Total Run time (ms)
3443	10	100	16730	168	99	91911
3439	9	90	16729	186	89	99988
3435	8	80	16715	209	79	108254
3431	7	70	16715	239	69	116629
3427	6	60	16718	279	59	124995
3423	5	50	16705	335	49	133341
3419	4	40	16679	417	39	141530
3415	3	30	16692	557	29	149902
3411	2	20	16681	835	19	158212
3407	1	10	16682	1669	9	166533