COMP 4511- System and Kernel Programming in Linux
Programming Assignment #5

Weighted round-robin scheduler

Description

In this assignment, you are going to implement a weighted round-robin scheduler. You are not required to start from scratch. Please refer to the following 3 lab exercises: MyFIFO scheduler, Round-robin scheduler, and syscall

Fair-share scheduling

As we learned in the lecture, Linux scheduler repeatedly switches between all the running tasks on the system, attempting to give a fair amount of CPU time to each task. Fair-share scheduling is a strategy that shares the CPU among sets of processes according to the groups that own those processes.

For example, Jason, Shining, and Ying belong to different groups. They are using a machine which implements fair-share scheduling. Jason has 1 runnable process, Shining has 3, and Ying has 6. Fairshare scheduling views these 10 runnable processes in three groups, and each group receives 1/3 of the CPU cycles to allocate to the processes that it owns.

In this example, Jason would get about 33% of CPU time for the single process. Shining would get around 11% of CPU time for each of the 3 processes. Ying would get roughly 5.5% of CPU time for each of the 6 processes. Suppose James belonges to Jason's group and he logs in to the machine and creates 2 extra processes. Then Jason and James together would have 3 processes that in total still receive 1/3 of the CPU, so that each process would get 11% of the CPU. No change happens to Shining and Ying.

Weighted round-robin scheduling

Weighted round-robin (WRR) supports a **different** version of fair-share scheduling. WRR uses fair-share scheduling based on each process's identification. That is, WRR shares the CPU among sets of processes at the granularity of individual processes, instead of group as we've seen above. The WRR scheduler assigns each process a time quantum proportional to the weight of the process.

For example, if process A has weight 1 and process B has weight 2, A will receive 33% and B will receive 67% of the CPU time.

Important notes

- Please review the following labs: syscall, MyFIFO and Round-robin. They guide you how to implement a syscall and integrate a scheduler to the kernel.
- Most scheduler functions are the same as the round-robin scheduler, except `task_tick_wrr` and `get_rr_interval_wrr`

Problem Statement

- Implement the new WRR process scheduler. Please provide the default implementation within the section: `#ifdef CONFIG_SMP` `#endif`. These functions can be found in the round-robin lab exercise
- The new scheduler can run individual processes (e.g., top displays that your WRR processes are running) and can proportionally assigns CPU time to individual processes based on their weight (e.g., top displays CPU usage proportional to process weight).
- The process weight will be set by a new syscall: `sys_set_wrr_weight`
- All processes are executed on a single core CPU environment. You can reset the number of CPU to 1 in VirtualBox

Entity class

You should add the following entity class in an appropriate file inside the Linux kernel source directory (please refer to the "MyFIFO scheduler" lab and the "Round-robin process scheduler" lab.

```
struct sched_wrr_entity {
    struct list_head run_list;
    int wrr_weight;
    unsigned int time_slice;
};
```

In `sched_wrr_entity` struct, a new attribute called `wrr_weight` is added. You can implement a similar run queue struct as the round-robin lab.

System call implementation

Implement a system call `sys_set_wrr weight` with the following prototype:

```
long sys_set_wrr_weight(pid_t pid, int weight);
```

Hint: use `find_task_by_vpid` to find a task struct pointer by the given pid.

User-level test program

A user-level test program (`wrr_test.c`) is provided. To make grading easier, you MUST set the syscall number of `sys_set_wrr_weight` as `359` and the scheduler policy number of the weighted round-robin as `7` when you modify the kernel source codes:

```
/* You need to define them in kernel. Please refer to the
FIFO scheduler and the syscall labs */
#define __NR_set_wrr_weight 359
#define SCHED_WRR 7
```

The following part is the main logic for testing the weighted round robin scheduler:

```
ret = sched_setscheduler(getpid(), SCHED_WRR, &sp);
if (ret == -1){
   fprintf(stderr,
    "[Error]: Child-%d failed to change to WRR scheduler\n",
     getpid());
   exit(1);
}
ret = syscall(__NR_set_wrr_weight, getpid(), weights[i]);
```

Please use `sudo` to run the test program

Submission

- `sched_wrr.c`
  - The implementation of the weighted round-robin algorithm.
  - For simplicity, the syscall should also be implemented in the same file
  - You can set the default time slice as `WRR_TIMESLICE`
    - `#define WRR_TIMESLICE  (100 * HZ / 1000)`
    - Basically, it is the same as `RR_TIMESLICE`, the default time slice for defined in `<linux/sched/rt.h>`. HZ is architecture dependent. In i386 and x86-64 architecture, HZ is defined as 1000.
- To make grading easier, you are expected to place the file in the following path:
  - `kernel/sched/sched_wrr.c`

```
/* file: kernel/sched/sched_wrr.c */
/* implementation of the weight round-robin scheduler */
/* implementation of set_wrr_weight syscall */
asmlinkage long sys_set_wrr_weight(pid_t pid, int weight) {
}
```