Project 4: CPU Profiler

- **Handed out:** Friday, April 15, 2021
- **Due dates:**
  - Part 1: Friday, April 23, 2021
  - Part 2: Friday, April 30, 2021

Introduction

The goal of this project is to design a CPU profiling tool. The tool will be designed as a kernel module which when loaded, keeps track of the time spent on CPU for each task.

Recommended Background Reading

- Linux /proc file system
- Sample /proc implementation: cifs_debug.c
- How to set up Kernel/QEMU
- Kprobe: documentation, examples
- x86_64 calling convention: documentation
- spinlock: API
- Jenkins hash: API
- Time measurement (rdtsc): API

Part 1. Monitoring task scheduling

In part 1, you will design a kernel module, named *perftop*, which will monitor the *pick_next_task_fair* function of Completely Fair Scheduler (CFS). To this end, You will use *Kprobe*, a debugging tool in linux kernel. With Kprobes, you can place a pre-event and post-event handlers (callback functions) on a certain kernel instruction address (similar to gdb's breakpoint). The module will display profiling result using the *proc* file system.

Program your module in `perftop.c` and `perftop.h` (as needed). Create `Makefile` that support *all*, *clean*, *install* and *uninstall* rules (and more as needed), similar to that of project 2.

Part 1.1. Setup procfs

[10 points] The first task is to setup a proc file (*procfs*) where the results of the profiler can be displayed.

- Review the Linux kernel documents and sample codes about *proc* file system. The links provided in the above *Recommended Background Reading* would be a good staring point.
- Write a kernel module named *perftop*
- The module should create a proc file named *perftop*
- *cat /proc/perftop* should display "Hello World"

Deliverables:

- Load *perftop* module
- Invoke *cat /proc/perftop*
- Take a screenshot of the output. Name your screenshot as `perftop1.png`

Part 1.2. Setup Kprobe

[10 points] We will set up Kprobe for the CFS's *pick_next_task_fair* function.

Tasks:

- Review the Linux kernel documents and sample codes about *KProbe*. The links provided in the above *Recommended Background Reading* would be a good staring point.
- Set a kprobe hook on the *pick_next_task_fair* function. Register a pre-event handler named *entry_pick_next_fair* and a post-event handler called *ret_pick_next_fair*.
- The event handler should increment its own counter, named *pre_count* and *post_count*, repsectively.
- The counter should be displayed by *cat /proc/perftop*.

Deliverables:

- Load *perftop* module
- Invoke *cat /proc/perftop* two times with some time gaps (e.g., 10 seconds).
- Take a screenshot of the output. Name your screenshot as `perftop2.png`

Part 1.3. Count the Number of Context Switches

[30 points] We will count the number of cases where the scheduler pick a different task to run: i.e., *prev task != next task*.

Tasks:

- Set up a kprobe hook on the *pick_next_task_fair* function (same as Part 1.2).
- On a pre-event handler *entry_pick_next_fair*, obtain the pointer of (prev) *task_struct* from *struct pt_regs *regs* using the register calling convention.
- On a post-event handler *ret_pick_next_fair*, obtain the pointer of (next) *task_struct* from *struct pt_regs *regs*. Check if the prev and next tasks are different. If so, increment the counter named *context_switch_count*.
- The counter should be displayed by *cat /proc/perftop*.

Deliverables:

- Load *perftop* module
- Invoke *cat /proc/perftop* two times with some time gaps (e.g., 10 seconds).
- Take a screenshot of the output. Name your screenshot as `perftop3.png`
- Make a folder named with your SBU ID (e.g., 112233445), put `Makefile`, `perftop.c`, `perftop.h` (if any), and the screenshots `perftop{1,2,3}.png` files in the folder, create a single gzip-ed tarball named `[SBU ID].tar.gz`, and turn the gzip-ed tarball to Blackboard.

Part 2. Print 10 most scheduled tasks

[50 points] In part 2, you will modify the kprobe event handlers in *perftop* to keep track of time each task spends on CPU and print the 10 most scheduled tasks using *proc*.

Preliminaries:

- We will measure time using *rdtsc* time stamp counter.
- Set up a hash table where a PID is used as a key and the start tsc (the time a task is scheduled on a CPU) is stored as a value.
- Set up a rb-tree that are ordered by the total tsc (the accumulative time) spent by a task on a CPU.

Tasks:

- On a post-event handler *ret_pick_next_fair*, if the prev and next tasks are different, we measure the time spent by each task on CPU as follows.
- (1) For the *prev* task: we read the current tsc and obtain the start tsc of the *prev* task from the hash table (using PID as a key). The difference between two timestamps (say, *elapsed time*) will represent the amount of the time the prev task has been scheduled on a CPU during the scheduling turn.
- (2) For the *prev* task: we remove the old entry from the rb-tree and add the new entry with the updated total tsc (the accumulative sum of elapsed times) to the rb-tree.
- (3) For the *next* task: we update the start tsc of the *next* task in the hash table with the current tsc.
- Modify the open function of proc file to print the top 10 most scheduled tasks. Print the PID and the time (total tsc) spent on a CPU.

Deliverables:

- Load *perftop* module
- Invoke *cat /proc/perftop* two times with some time gaps (e.g., 10 seconds).
- Take a screenshot of the output. Name your screenshot as `perftop4.png`
- Make a folder named with your SBU ID (e.g., 112233445), put `Makefile`, `perftop.c`, `perftop.h` (if any), and the screenshots `perftop4.png` file in the folder, create a single gzip-ed tarball named `[SBU ID].tar.gz`, and turn the gzip-ed tarball to Blackboard.