Assignment 6: Design and Implement a CPU Profiler (Part 1)

Introduction

This project aims to design and implement a CPU profiling tool. The tool will be developed as a single kernel module that, when loaded, tracks statistics for each task on the system, such as total CPU time and the number of times a task is scheduled in and out. The module will display profiling results using the `proc` file system. This project is divided into two parts (`A6` and `A7`) but will use and update the same kernel module code created in Part 1 (`A6`).

Recommended Background Reading

- Kprobes: documentation, examples
- x86_64 calling convention: documentation
- stack trace: source code, example
- spinlock: API
- Jenkins hash: API
- Time measurement (rdtsc): API
- Symbol look up: API

Part 1: Counting per task scheduling

[40 points] In part 1, you will design a kernel module that counts how many times a task has been scheduled onto the CPU. To achieve this, you will use `Kprobes` , a debugging tool in the Linux kernel that allows you to break at any kernel address. `Kprobes` can be configured to trigger when a specific function is executed, transferring control to an event handler routine.

Part 1.1. Setup procfs

[10 points] The results of the profiler tool should be displayed using the **/proc** file system. The first step is to create a **/proc** file for the profiler.

Tasks:

- Write a kernel module named `perftop` ;
- `perftop` should create a **/proc** file named `perftop` ;
- `cat /proc/perftop` should display "Hello World".

Deliverables:

- Load your `perftop` kernel module;
- Invoke `cat /proc/perftop` ;

- Add a screenshot of the output from the above execution.

Part 1.2. Setup Kprobes

[15 points] Next, we will count the number of times the proc file we created in Part 1.1 (i.e., `/proc/perftop`) is opened using `Kprobes`.

Tasks:

- Understand the API for using `Kprobes`;
- `Kprobes` should call an event handler every time `cat /proc/perftop` is invoked;
- The event handler should increment a counter;
- The counter should be displayed by `cat /proc/perftop`.

Deliverables:

- Load your updated `perftop` kernel module;
- Invoke `cat /proc/perftop` 3 times;
- Add the screenshot of the three executions in the same window.

Part 1.3. Count the number of times a PID is scheduled in

[15 points] Next, we will count the times a **PID** has been scheduled. This will now track all PIDs on your Linux system.

Tasks:

- Setup a hash table with key as PIDs and value as the number of times that a specific PID has been scheduled;
- Setup a `Kprobes` hook on `pick_next_task_fair` function;
- Using the register calling convention, get the pointer of `task_struct`;
- Using the `task_struct`, extract the associated PID of the task;
- If the PID already exists in your hash table, then increment the value; otherwise, create a new hash table entry and set the value to 1;
- Modify the `open` function of the proc file to print (with `cat /proc/perftop`) the PIDs and their corresponding values (the number of times that PID was scheduled).

Deliverables:

- Load `perftop` module
- Invoke `cat /proc/perftop`
- Add screenshot of the output
- Upload the source code tarball