Assignment 7: Design and Implement a CPU Profiler (Part 2)

Part 2: Record the stack trace and calculate the task execution time

[40 points + 20 bonus points] In Part 2, you will modify your kernel module, `perftop`, to: **(a)** track the time that both user and kernel **tasks** spend on a CPU and **(b)** print the 20 most scheduled tasks on the system using `/proc`.

Part 2.1. Storing kernel stack trace

[20 points] Modify your kernel module from Part 1 to track **tasks** instead of PIDs. A **task** is defined as a unique user or kernel **stack trace**. The hash table key will now be a hash of a task stack trace, and the value will be the number of times the unique kernel stack trace is scheduled on the CPU. Particularly, referring to background reading *stack trace* and *Jenkins Hash* in Part 1 of the project (`Assignment 6`).

Tasks:

- In the `Kprobes` event handler, get a given task's stack trace
  - Use `stack_trace_save` function for a kernel task
  - Use `stack_trace_save_user` function for a user task
  - If the function you want to use is not exported, you can obtain a function pointer with ~~`kallsyms_lookup_name()`~~ the following `kprobe` workaround:

```
struct kprobe kp = {
   .symbol_name = "function_name"
};
register_kprobe(&kp);
function_ptr = ([function type])kp.addr;
```

- Modify the kernel modules hash table to store the **stack trace** instead of **PID** as the key (you can pass the stack trace buffer to Jenkins hash)
- Increment the schedule count of each **task** (stack trace) in the hash table
- Modify the `open` function of the `proc` file to print with `cat /proc/perftop` the stack trace dump *and* its corresponding number of times it has been scheduled

Deliverables:

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

Part 2.2. Store schedule time

[20 points] Calculate the time spent by each kernel task scheduled on a CPU and store it.

Task:

- Modify the `Kprobes` event handler to measure the time spent by each task on CPU (the time the task is actively running *TASK_RUNNING*)
    - Modify the hash table value to store task and corresponding time spent on CPU
    - Measure the time using `rdtsc` counter
    - When the event handler is invoked: 1) A task is scheduled *out* and a new task is scheduled *in*; 2) Calculate the time spent on the CPU for task being scheduled *out* and update the hash table; 3) Start the timer for the new task being scheduled *in*
- Modify the open function in the `/proc` file to print the stack trace and the cumulative time spent by the task on the cpu. The time can be in rdtsc ticks; please denote this unit of measurement in your print line.

Deliverables:

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

Part 2.3: Use rb-tree to get 20 most scheduled tasks [Optional]

[20 points] Modify the kernel module to print the 20 most scheduled tasks.

Note: This is an optional task. Credits are added to your total homework points, provided they do not cause your total to exceed the maximum points for the homework section.

Tasks:

- Maintain a rb-tree with the key as the *cumulative* time spent by a task scheduled on the CPU and value as the stack trace;
- Whenever a task is scheduled *out*, remove the old entry of the task from rb-tree and add the new cumulative time spend and stack trace to the rb-tree;
- Modify the open function of proc file to print the **top 20 most scheduled tasks**. Specifically: print the rank of the stack trace (rank 1 scheduled task: ..., rank 2 scheduled task: ...) stack trace jenkins hash,

the total time spent on CPU, and the stack trace dump with only max depth of 4 frames.

Deliverables:

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