Virginia Tech ❖ Bradley Department of Electrical and Computer Engineering

ECE 4984 / 5984 Linux Kernel Programming
Fall 2017

Medium Project: Latency Profiler

1 Introduction

In this project, we will develop a *latency profiler*, which measures and accumulates time that a task starts sleep and wakes up. A task would sleep due to various reasons including blocking IO operations (e.g., disk, network) or contention on synchronization primitives (e.g., mutex, semaphore). In many cases, such long sleeping causes high latency in application behavior. Our latency profiler helps to find such latency bottleneck by measuring sleeping time of all tasks in a system.

This project is composed of an essential part and advanced part. The essential part is essential in functionality and takes 100% of score. All students should implement the essential part. In the advanced part, we will improve the modularity of the profiler and add additional features. By implementing the advanced part, you will get up to 75% bonus score.

The following concepts from the course will be put in practice in that project:

- Process scheduling
- Kernel debugging
- Kernel time management
- Kernel memory allocation
- Kernel synchronization
- `proc` file system
- `kprobe`

2 Project description

Our latency profiler should run in kernel space and measure sleeping time of all running tasks in a system. The profiling results should include the call stack information, where a task sleeps, accumulated sleeping time in CPU clock cycles in a descending order of sleeping time. *One restriction is that the modification of existing kernel data structures is not allowed.* All source code should be compiled with Linux Kernel v4.12. Following is the more detail description of the essential part and advanced part.

2.1 The essential part

Restriction : Do not modify existing kernel data structures.

Measuring latency : Measure and accumulate sleeping time of each task. The unit of measurement is CPU clock cycles.

Kernel call stack : Maintain sleeping time information for each `pid` and kernel call stack combination.

Start and stop of measurement : Start profiling when kernel boots and continue until kernel shuts down.

Print out results : Prints out the 1000 longest sleeping task information, including sleeping time, pid, process name, and call stack with function names, periodically (e.g., every 10 seconds or every 10,000 context switch operations). The period is not necessary to be accurate.

2.2 The advanced part

Restriction : Do not modify existing kernel source code at all and write the latency profiler as a kernel module.

Print out results : Create `/proc/lattop` in `proc` file system. Any user application should be able to get profiling results so far by reading the file. For example, `cat /proc/lattop`

User-space call stack : In addition to kernel space call stack, maintain user-space call stack. Print out just user-space addresses without function names.

3 Additional information

Kprobe example : `linux/samples/kprobes/kprobe_example.c`

x86_64 calling convention : `https://en.wikipedia.org/wiki/X86_calling_conventions`

stack trace : `linux/include/linux/stacktrace.h`

spinlock : `linux/include/linux/spinlock.h`

CPU clock cycle : `rdtsc`

Jenkins hash function : `linux/include/linux/jhash.h`

Proc file system : `linux/include/linux/proc_fs.h`

4 Results to be handed

The deadline is **10/25 11:59 PM EDT**. Following is expected to be submitted:

- Source code, which is either of patch file for the essential part or a module source code for the advanced part
- Report in PDF, which briefly describes your design and your analysis of results while cloning the Linux kernel repository.

All of this should be contained in a tarball should, with the following format: `<VT ID>.project2.tar.gz` (e.g., `johndoe.project2.tar.gz`)

5 Project TA office hours

- *TA*: Ho-Ren (Jack) Chuang
- *Date and time*: Monday:3-5pm, Wednesday:3-5pm
- *Email*: `horenc@vt.edu`
- *Location:* 460 Durham