Homework 1 - System Calls and Virtual Memory

Due: October 15th, 2018, 11:00pm

GitHub page: https://github.com/SudarsunKannan/CS519/blob/master/homework1/homework1.md

In the second homework, we will explore the cost of system calls and the page fault cost. The homework will consist of three parts: part 1 and part 2 will require you to develop a benchmark to measure the cost of a system call, whereas the part 3 will involve reducing the cost of page fault handling by modifying the OS virtual memory management.

Linux 4.17 kernel source code can be found here

```
https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/?h=linux-4.17.y
```

PART 1

The first step would be to write a simple benchmark to measure the cost of the system call; you will add a new dummy system call (hello_kernel) to your OS in the Linux kernel's mm/mmap.c file. A couple of references below show how to add a new system call; there are thousands of other references online.

Reference 1 Reference 2

Once you have added a system call, measure the cost of invoking a system call from userspace by invoking the system call few hundred times and measuring the average latency.

NOTE: Make sure that your system call is getting called by adding a **printk()** within your dummy system call. You should remove the print statement after confirming that the new system is getting invoked inside the kernel.

PART 2

In the second part, we will study the cost of page fault. Remember that Linux OS allocates pages **on demand** (the first time a page is accessed (or touched) after allocation). First, you will allocate a large 2GB memory-mapped region which is also page-aligned. Next, you will touch (write) the first byte of each page sequentially and measure the avergage cost of accessing a page (i.e., page fault cost).

Note: For allocating memory-mapped region

Use `mmap()` system call that allocates page-aligned memory. When using `mmap()`, you should map an anonymous memory and not a file backed memory. See the following link for more details how to allocate anonymous memory.

Reference 1 Reference 2

PART 3

The part 3 of homework is an attempt to reduce the cost of page fault handling cost. Every time a page fault occurs, instead of allocating only one page, you are required to allocate two or more pages after a page fault. For example, one page for the actual faulting address (say, addr X) and the next page (say, addr X + 4096). For this, you will modify the OS virtual memory fault handler.

The page faults for anonymous memory are handled in the following function inside the mm/memory.c source file.

```
int do_anonymous_page(struct vm_fault *vmf)
```

This function first checks whether the page fault is read or write fault (due to read or write access), then (1) allocates a page, (2) creates a new page table entry (PTE), and (3) adds the PTE to the page table.

We will walk through this function in the class.

Resources

In this class, we will use the QEMU-based virtual machine to test the modified OS. QEMU VMs run either on a bare-metal OS or even inside another virtual machine. More details on QEMU can be obtained [here] (https://www.qemu.org/).

For students new to QEMU or hacking kernel, we have created a set of instructions about how to compile a custom kernel (OS) and how to run the OS using QEMU. Detailed step-by-step instructions can be found here.

```
https://github.com/SudarsunKannan/CS519
```

Computing Resource

If you need access to a development environment and cannot use your laptop, please send me an email (sudarsun.kannan@cs.rutgers.edu).

Starting Early

This a significant-but-essential homework for understanding the basics of OS virtual memory. Please start working on this homework early. If you have questions, make sure to ask them during office hours.