Project 2: I/O Elevators

Courtney Bonn, Isaac Chan

Group #39

Abstract

For the second kernel assignment, we are tasked with building a new I/O Scheduler that is based off of the current NOOP Scheduler in the Linux Kernel. The algorithm that will be used for the new I/O Scheduler is CLOOK.

I. Design Plan

The algorithm we chose to implement is CLOOK. After examining the NOOP scheduler, we came to the conclusion that the only two methods to be altered are the dispatch and add_request methods. The dispatch requires an addition to tell if the request is for reading or writing. The add_request method requires an addition to put a new request into the queue. Because the CLOOK algorithm is circular, the request must be put in the correct spot in the queue. This can be done by: check the sector position while iterating and then insert the request if the sector is larger.

II. Version Control Log

III. Work Log

Date	Time	Person	Event
October 25, 2017	5:15pm	Courtney	Began Project 2 LaTeX file
	6:15pm	Courtney	Changes Tex Template to match Project 2 and made sure it compiled correctly with ”make”
October 29, 2017	11:30am	Courtney	Started reading Chapter 14 of Love’s Kernel book
	11:45am	Courtney	Copied noop-iosched.c to our working directory, renamed it sstf-iosched.c, and changed instances of noop to look
	12:00pm	Courtney	Continued research on NOOP, LOOK, and C-LOOK algorithms
	10:00pm	Courtney	Working on getting the VM run with the NOOP Scheduler
	10:30pm	Isaac	Examine NOOP scheduler and understand how it works
	11:30pm	Isaac	Design a plan to implement CLOOK algorithm
	11:45pm	Isaac	Begin implementation of CLOOK algorithm
October 30, 2017	12:30am	Isaac	Complete initial CLOOK implementation
	1:00am	Courtney	Working on Kernel error
	1:51am	Isaac	Recopied Yocto directory with changed files to rebuild the kernel
	1:55am	Isaac	Successfully built the kernel with the new scheduler
	2:05am	Courtney	Ran qemu with new scheduler and confirmed that clook scheduler is the default in the VM
	2:20am	Courtney	Design a plan to test the new scheduler
	3:45am	Courtney	Still working on how to test scheduler
	4:47am	Courtney and Isaac	Trying to figure out why CLOOK scheduler isn’t printing/working, not having much success
	5:30pm	Courtney and Isaac	Working through Kernel Panic, trying to determine where the error is
	7:30pm	Courtney and Isaac	Wrapping up assignment with final commits and documenting test plan

IV. Write Up

1) What do you think the main point of this assignment is?

   The main point of this assignment is to learn how to work with the disk scheduler, or I/O scheduler, on the kernel. Using the current I/O schedulers as base algorithms to work off of, we will build a CLOOK algorithm on the NOOP scheduler.

2) How did you personally approach the problem? Design decisions, algorithm, etc.

   This assignment took a lot of research ahead of time before actually sitting down to work on the code. First off, we needed to understand what the current algorithm was doing. NOOP is essentially a first-come-first-served (FCFS) or FIFO algorithm, meaning whatever request it receives first is the first one it is going to process. Additionally, it does not do any sorting. It does sort a new request with adjacent requests.

   CLOOK is a circular variant of LOOK. It only scans in one direction and starts at the beginning when it reaches the end. When a new request arrives it must be sorted to the correct spot in the queue.

   To implement our selected algorithm, we made use of the Kernel's Linked List implementation.

3) How did you ensure your solution was correct? Testing details, for instance. Our plan for testing was to run a python script that created a file and wrote to it. This triggers file/IO steps that we can then check the system log for clook printouts. However, because we were unsuccessful at getting the kernel to boot with our new I/O scheduler, we were unable to fully test the solution. Once the CLOOK algorithm was written, we were able to select it as the default scheduler when building the kernel and the kernel was able to compile. We made the necessary adjustments to the qemu command by disabling virtio and changing *root=/dev/vda* to *root=/dev/hda*; however, when the qemu command was ran, there was a kernel panic. We attempted for several hours to adjust our algorithm in order to avoid a kernel panic, but we were unable to determine the root of the problem. We believe the problem was with our add request function, as that ended up being the only one we modified. During the kernel boot, you can see "CLOOK add 2" which indicates that an add does occur, but immediately after an error occurs stating *BUG: unable to handle kernel NULL pointer dereference at 0000008a*. We are not 100 percent sure that this is the issue with the kernel panic, but we believe it may be attributed to it.

4) What did you learn? We learned about implementing an elevator algorithm and also with configuring a Linux kernel. We were able to change qemu flags to fit our needs. Even though we were not able to successfully apply our new scheduler and were unable to determine its correctness, we were able to learn quite a lot with regards to the Linux kernel.

5) How should the TA evaluate your work? Provide detailed steps to prove correctness

   - Apply kernel patch file and build the kernel with *make -j4 all*.
   - The build should ask you what scheduler to use; select CLOOK as the default.
   - Source the environment script.
   - Launch qemu with *qemu-system-i386 -gdb tcp::5539 -S -nographic -kernel ./linux-yocto-3.19/arch/x86/boot/bzImage -drive file=core-image-lsb-sdk-qemux86.ext4,if=ide -enable-kvm -usb -localtime –no-reboot –append "root=/dev/hda rw console=ttyS0 debug"*
   - Connect to qemu with gdb *target remote:5539* and *continue*
   - There will be many CLOOK messages as the kernel boots.