Project 2: Taking an I/O Elevator to Dante's Inferno

CS 444: Operating Systems II **Oregon State University**

Group 31

Spring Term, May 7th, 2018

Prepared by: Kevin Talik, Austin Sanders, and Zachary Tusing

Abstract

This paper will explain the functionality of an IO Scheduler (or "Elevator") algorithm in context to the current No-Op scheduler in the linux-yocto project (version 3.19.2). Following the background, is an explanation of how to implement a different scheduler, CLOOK. CLOOK is a scheduling algorithm that avoids starvation and "Hard Drive Time Bias", because it uses less needle seeks on average [1].

1 INTRODUCTION

In this project, we were asked test the stability of the I/O elevators of the linux-yocto kernel git.yoctoproject.org/cgit.cgi/linux-yocto/tree/drivers?id=v3.19.2 Instinctively, we took turns testing a new elevating device with our body weight until it broke. After taking out the ”if=virtio” tag in the command that launches the virtual machine, the kernel asks to specify a file as to what scheduler to use. We then specified our scheduler that we wrote called *sstf-iosched.c*.

The primary data structure for the IO elevator algorithm that we are using is the: http://git.yoctoproject.org/cgit.cgi/linux-yocto/tree/include/linux/list.h?id=v3.19.2. Using this predefined linked list data structure in the linux kernel, our team implemented a CLOOK algorithm.

This paper will first cover the functionality of an I/O elevator, and a few examples of modern input/output algorithms. Next, this paper will illuminate the current I/O elevator (FIFO NO-OP) that is in the file system specified above. With this information, our team will describe our experiences implementing the LOOK I/O Scheduler, and what to expect when swapping out the *virtual* device that reads the input from your *physical* device. Then we will describe how virtual operating systems work

Concluding this paper is a synopsis our what each member of our group worked on during this project, and how we felt about it.

2 VIRTUAL ELEVATORS/SCHEDULERS

Elevators, or schedulers, are the abstract machines that organize input and output to other devices. If a program would like to write some characters to the hard drive, the program has to wait until the computer has processed the other programs that wanted to write first. The kernel shares the virtual resource (known as a ”block”) through scheduling. This is the premise of IO scheduling.

The virtual blocks are placed into a data structure, and it is up to the developer to decide what algorithm to use; your algorithm may vary. The No-Ops scheduler is a simple, yet powerful algorithm for reading and writing to random access memory. This uses a FIFO queue. This algorithm may not be a good option for HDD, since the needle needs to spin to get to the right sector.

The LOOK scheduling algorithm puts requests into list, and then scans back and forth to get to process the requests. This algorithm is found inside of the file /linux/list.h. However, this algorithm still has a time bias with the time sharing on a device that takes time to spin (the hard drive).

CLOOK is a clever change to the LOOK scheduler, as it used a circularly linked list. This saves time by scanning in one direction without having to check data structure conditions. This is also helpful to the programmer, who only has to think about requests in one direction while designing an algorithm.

3 QUESTIONS

In the section below we will answer the list of questions assigned to us on canvas.

3.1 Purpose of the Assignment

The purpose of the assignment, in our opinion, was to gain a better understanding of Linux kernel modules worked. This assignment had us learning about how structures are generally implemented across many different branches of the OS. This knowledge helped us gain a better understanding of how each subsection of an operating system works. This assignment then drew us closer to the assignment of how I/O schedulers work within Operating systems.

3.2 Plan of Attack

The first thing I did when approaching this problem was to try and get a better idea of the big picture when it comes to I/O schedulers before I got lost in the details of actual implementations. I read the text book *Linux Kernel Development, 3rd ed.*[2] and listened to the professors lectures. After understanding how the I/O scheduler worked I then went and learned the different implementations (No-Op, deadline, fair scheduling, etc.) and learned their respective advantages and disadvantages. After I got a handle on that I then had to learn how to write and actual module for a Linux Kernel. I found the online resource *The Linux Kernel Module Programming Guide*[3], and found it very useful for understanding the code I had to write. While using the guide I constantly was looking at the noop-ioschded.c implementation so I could understand what the author wrote and why.

Once I had a grasp of what I/O schedulers are and how to write them I then started diving into the code to understand what previous authors wrote and why. I walked line through line of the NOOP I/O scheduler and sourced each function call or data reference to its library using *Elixir Cross Referencer*[4]. With bootlin I could find the actual line where the functions or data type was declared and written to fully understand its purpose and implementation.

After having completed all these steps I was ready to write the code. I had several points of reference open when writing and just wrote the functionality one line at a time. I was constantly compiling to make sure my code had no syntax errors, and just wrote the module till it was fully implemented and done.

3.3 Ensuring the Results.

The first thing I did before even thinking of running the program was making sure my completed module could compile with no errors. Pulling a make file from the Module Programming Guide [3] I wrote a new makefile to ensure I could compile my module with no syntax error or warnings. Once I compiled the module I then moved on to compiling the entire Kernel again with modified Kconfig.iosched and Makefiles to include my newly created I/O scheduler.

I allotted four cores to its compilation and recompiled the entire Kernel. After a few minutes of compilation the kernel recompiled with no new errors or warning. After that it was time to boot the new operating system using QEMU.

3.4 What We Learned

There was a lot to take from this assignment. We only barely scratched the surface of writing modules and block I/O schedulers as well. The big take aways from this assignment was more big picture stuff.

We learned the difference between between scheduling algorithms. LOOK is a scheduler that is very similar the SSTF scheduler except that it avoids starvation. It will travel back and forth across the disk organizing the list in a way where the scheduler will always hit the next closest sector on the disk minimizing the time it spend searching. CLOOK is similar to LOOK except for the fact it travels in only one direction. This means that if there is a cluster of requests along the edge the arm won't allow the other requests to be slowed down when servicing a cluster of edge requests. This means that CLOOK gets a more equally balanced service time than LOOK does.

Deadline's focus is to make sure starvation does not happen. It has two queues which contain read and write requests respectively. Then it has one last queue which contains all the requests with respect to the sector number. It then services requests according to which process in the queue is closest to it's deadline. This means that if processes have certain required time constraints they will be serviced according to those deadlines. This allows for a psudo-fixed priority schedulers.

Finally I learned about completely fair queueing. This algorithm places requests into queues according to the process that requested it. It then allocates a certain amount of time for each queue to spend reading or writing. Different processes can be assigned different priorities increasing or decreasing the amount of time allocated to the process for reading or writing to the disk. For example if you had a high priority process you could assign it a high priority, so when the scheduler assigns time slices to each queue the queue with the higher priorities will be allowed to spend more time with the disk. Ensuring the everything is fair and all processes get the correct amount of relative time on the disk.

This then brings us to how the Qemu commands were changed. We removed the command ”if=virtio”. This command installs the Virtio drivers. Virtio is an I/O virtualization system. This is a system meant to interact with the hypervisors, hypervisors run the virtual machines. When using Virtio the operating system running on the virtual machine will no longer have ”direct contact” with the hardware virtaulization. This means that Virtio will be adding a level of abstraction in between the hypervisor and the host machine. They call this a guest-hypervisor connection. This adds what virtio calls ”para-drivers”. These para-drivers allow for virtualized operating systems to get better performance. In this case however, we end up losing control over the virtualized hardware. This effect causes us to not handle exactly how we handle our virtual devices. Which in turn causes our I/O scheduler to break.

Virtio has a driver for each device that is being virtualized by the operating system. The Virtio system also implements it’s own buffer system for how the front-end drivers(Operating system running on the virtual machine) communicates with hypervisor(the program that is simulating hardware). This means that we would need to specially design our I/O scheduler to work with Virtio. This would remove from our learning about Operating Systems and provide a level of abstraction that isn’t really there when working with non-virtualized operating systems.

3.5 How to Grade

The first thing I would look at would be the source code for the Shortest Seek Time First I/O scheduler, since that was a majority of the implementation. Look at how we implemented each function for the operations struct inside the elevator type struct. I would also take a look at the slightly modified Kconfig.iosched and Makefile. If you do not know what I changed specifically you can check the hw2.patch for a list of modifications to each file.

Once you can see the all the changes you should move the files into a source tree of yocto and recompile the kernel with the new modifications. Once that is complete you can run it on QEMU and use built in commands to read and write files and then check the Kernel log to see the module printing its printk messages.

After that you should read through this entire document to make sure we understand the concepts associated with this subject. You can also double check all of our work by going over our version control logs and work logs. If you also desire evidence that our work logs are true and complete you can go to the github repository that we were using for this assignment[5]. That is the approach I would take to grade this and understand our work.

4 Concluding Thoughts

Traversing the filesystem of the linux kernel is always a special moment in a developers life. Generally, if a solution for a modern software problem involves changing the IO scheduler, you are looking in the wrong place. However, this was a fantastic moment to compile a kernel with our custom CLOOK algorithm. There is no better way to understand the nature of the linux kernel then by swapping our their words with our files. Austin wrote most of the code for implementing the circularly linked list in the *sstf-iosched.c* file. Kevin focused on compiling and managing the *tex files for the group.

5 Version Control Log

commit 2196b4e0312896a1882c8c7224ac98e5ab0264d9

Author: Zachary Tusing ¡zttusing@protonmail.ch¿

Date: Tue May 8 19:19:15 2018 -0700

Update project2CS444.tex

commit 2196b4e0312896a1882c8c7224ac98e5ab0264d9

Author: Zachary Tusing ¡zttusing@protonmail.ch¿

Date: Tue May 8 19:19:15 2018 -0700

Update project2CS444.tex

commit 26ed53a5b8929bb5a88ca68271c6a7aac5d075b8

Author: Zachary Tusing ¡zttusing@protonmail.ch¿

Date: Tue May 8 19:02:20 2018 -0700

Update project2CS444.tex

Author: Austin Sanders ¡sanderau@oregonstate.edu¿

Date: Tue May 8 17:40:50 2018 -0700

Add files via upload

commit 5d1cf152f24116e8cfe2f1d1d5de12d0b98ce785

Author: Zachary Tusing ¡zttusing@protonmail.ch¿

Date: Tue May 8 17:22:12 2018 -0700

updating tex

commit 231ffc4b95e28e17744ebbd745d380e6507ce385

Author: sanderau ¡sanderau@oregonstate.edu¿

Date: Tue May 8 14:46:20 2018 -0700

added the sections: plan of attack, ensuring the results, and what we learned. Still more to add and edit

commit 633abc0d6b4ec07d6a8980b770c47f4b95d8c1a1

Merge: a2bff8e b4adb80

Author: Kevin Talik ¡Kevin Talik¿

Date: Tue May 8 12:50:00 2018 -0700

Merge branch 'master' of https://github.com/sanderau/OS2_group31

commit a2bff8e362dee19a485bf69b2a508caa0b05620e

Author: Kevin Talik ¡Kevin Talik¿

Date: Tue May 8 12:49:56 2018 -0700

Added my files for the individual writing assignments

commit b4adb80ce6bca025ac482f8bf109dea7e21af72a
Author: Zachary Tusing ¡zttusing@protonmail.ch¿
Date: Mon May 7 22:12:41 2018 -0700
Update project2CS444.tex

commit 93bb0fc27951e5c9ee77cc8bb8e9f88d11ccdb4d
Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿
Date: Mon May 7 21:46:44 2018 -0700
Adding the last two functionalities. Module now compiles with no errors and warnings. Should work. Just going to adjust some formatting

commit 99ec277e707c63d2e7d341df295f15359c02968f
Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿
Date: Mon May 7 21:27:35 2018 -0700
dispatch meow works

commit 50da0fe21d4cdbe5c32574a45a81ef30dc0ad799
Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿
Date: Mon May 7 21:14:14 2018 -0700
merge now works

commit f1d35b64b10646235f8a67e75b323ae6b9c42f93
Merge: 25ae9c2 4c2eca0
Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿
Date: Mon May 7 21:05:17 2018 -0700
Merge branch 'master' of https://github.com/sanderau/OS2_group31

commit 25ae9c260c94a64ee24c3195df069fee7bbb6699
Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿
Date: Mon May 7 21:04:57 2018 -0700
Can now successfully add a request to the elevator

commit 4c2eca013b6de8dd6c07fb45815831b947a56817
Merge: 9eb4f72 209a7d7
Author: Kevin Talik ¡Kevin Talik¿
Date: Mon May 7 18:51:54 2018 -0700
Merge branch 'master' of https://github.com/sanderau/OS2_group31

commit 9eb4f72db049415e4f13387ede958f71f008349f
Author: Kevin Talik ¡Kevin Talik¿
Date: Mon May 7 18:51:51 2018 -0700
Made some teeny tiny edits

commit 209a7d7963294a5690a85bdbafe703ad5ffa9f7e
Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿
Date: Mon May 7 18:46:12 2018 -0700
finished the exit function for the elevator

commit 247cb4492f43a6c73bcda17fa980be24ba44906f
Merge: 6684065 b3676dd
Author: Kevin Talik ¡Kevin Talik¿
Date: Mon May 7 18:38:30 2018 -0700
Merge branch 'master' of https://github.com/sanderau/OS2_group31

commit 66840658047164fa2a09787cabc5a942c961a698
Author: Kevin Talik ¡Kevin Talik¿
Date: Mon May 7 18:38:26 2018 -0700
Completed a full page of framing for the document

commit b3676dd9d4fded7f7df474a8348340eec43a061f
Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿
Date: Mon May 7 18:35:33 2018 -0700
init function is implemented and compiles

commit fbd64d7f416d57b81fcea997b9caba4a04aaab62
Merge: 8804004 71ca84f
Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿
Date: Mon May 7 17:56:41 2018 -0700
Merge branch 'master' of https://github.com/sanderau/OS2_group31

commit 88040043bd7be30b250171728ff139700529a9aa
Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿
Date: Mon May 7 17:56:17 2018 -0700
The Kconfig.iosched and Makefile should include the needed changes. All that is left is writing the functionality for the new elevatr

commit 71ca84ff187d26f744ade57f470fd4aa475dd23c
Author: Kevin Talik ¡Kevin Talik¿

Date: Mon May 7 17:31:08 2018 -0700

Filled out background on linux scheduling

commit 9a800595c58893ed786c9734676aeed0f9fb59bd

Merge: 2be0030 1170f3d

Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿

Date: Mon May 7 16:20:26 2018 -0700

Merge branch 'master' of https://github.com/sanderau/OS2_group31

commit 2be0030575e2b6d4d0a0e04ada3950dc54b3b583

Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿

Date: Mon May 7 16:19:47 2018 -0700

I added the initiate and exit functions for the module. Now all that is left is implementing all of the functionality for the elevator

commit 1170f3df20f33df96f36d3ad313e9dc43dcd972a

Author: Kevin Talik ¡talikk7@github.com¿

Date: Mon May 7 15:32:20 2018 -0700

Commiting my code so I can figure out copy and pasting functionality into putty

commit 8784eaad9e96c4bcf047881c4ace72dad41830bf

Author: Kevin Talik ¡talikk7@github.com¿

Date: Mon May 7 15:15:20 2018 -0700

Filling out the sections of the document

this is an attempt to get my personal github to push code to the repo

commit eeeb51e440b44dd6076f2e147c2aae3edb15a905

Merge: b7aa988 1355fe7

Author: Kevin Talik ¡talikk@flip1.engr.oregonstate.edu¿

Date: Mon May 7 14:42:27 2018 -0700

Merge branch 'master' of https://github.com/sanderau/OS2_group31

Attempting to merge with my flip account changes, lets see where this goes

commit b7aa988b728fa330d84e58bdac226f0979c4172c

Author: Kevin Talik ¡talikk@flip1.engr.oregonstate.edu¿

Date: Mon May 7 14:41:51 2018 -0700

Updated the README manifesto

commit 1355fe71f97c0e2542c2d38c07d55bce5abe742f

Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿

Date: Mon May 7 14:29:10 2018 -0700
Added the init and exit macro definitions

commit 30aaf60b490e4a192aea59db5a2165cfe513bce0
Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿
Date: Mon May 7 11:51:14 2018 -0700
adding and cleaning up the beginning of the sstf

commit 80f573d786841810a8ed1abb3f787a4cf2b31389
Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿
Date: Sun May 6 21:37:38 2018 -0700
The other two files we will need

commit b948543d4ce4708fdc3184ded73691888cc99b3b
Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿
commit 1355fe71f97c0e2542c2d38c07d55bce5abe742f
Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿
Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿
Date: Mon May 7 14:29:10 2018 -0700
Added the init and exit macro definitions

commit 30aaf60b490e4a192aea59db5a2165cfe513bce0
Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿
Date: Mon May 7 11:51:14 2018 -0700
adding and cleaning up the beginning of the sstf sched

commit 80f573d786841810a8ed1abb3f787a4cf2b31389
Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿
Date: Sun May 6 21:37:38 2018 -0700
The other two files we will need

commit b948543d4ce4708fdc3184ded73691888cc99b3b
Author: Austin Sanders ¡sanderau@os2.engr.oregonstate.edu¿
Date: Sun May 6 21:34:59 2018 -0700
Adding the scheduler, just the needed libraries right now

6 Work Log

Austin Sanders
Did all of the program implementation and helped write latex document

Kevin Talik

Helped with understanding and laying the foundation of the latex document

Zachary Tusing

Worked on the latex document

REFERENCES

[1] "Disk scheduling algorithms." [Online]. Available: http://www.cs.iit.edu/~cs561/cs450/disksched/disksched.html

[2] R. Love, *Linux Kernel Development, 3e*. Addison-Wesley, 2010.

[3] O. P. Peter Jay Salzman, Michael Burian. The linux kernel module programming guide. [Online]. Available: https://www.tldp.org/LDP/lkmpg/2.6/html/lkmpg.html

[4] Elixir cross referencer. [Online]. Available: https://elixir.bootlin.com/linux/v3.19.2/source

[5] K. T. Austin Sanders, Zach Tusing. Our github repo. [Online]. Available: https://github.com/sanderau/OS2_group31