Assignment 2 - I/O Elevators & Concurrency 2

Nickoli Londura & Benjamin Martin - Group 7
Fall 2018
CS 444
Oregon State University

October 29, 2018

Abstract

This is the assignment 2 write-up for group 7 of CS 444. This assignment combines the "I/O Elevators" section and "Concurrency 2" sections on Canvas. We explore ways to make an algorithm for the LOOK I/O scheduler in the yocto kernel, and run a VM on it on the OS2 server to see if the algorithm is working properly. We also try to solve the dining philosophers problem by running concurrent threads and applying mutual exclusion (mutexes) on the threads that need to share the forks.

1 Write Up

1.1 I/O Elevators

1. What do you think the main point of this assignment is? The main point is to understand how Linux kernel schedulers are implemented. By looking into LOOK and C-LOOK I/O schedulers, we were able to learn what kind of algorithms they use in order to execute processes in a distinct order for reading and writing to the hard drive.

2. How did your team approach the problem? Design decisions, algorithm, etc. We implemented the C-LOOK scheduler, but DONT BE DECEIVED! Everything says look, but we really mean clook. This was for ease of viewing on our own part. The C-look was easier to implement because we could just have the "elevator" run up and then reset back at the bottom once it has dispatched the uppermost request.

3. How did you ensure your solution was correct? Testing details, for instance. We started out by using print statements to see if the kernel was built and running properly, as well as knowing how the scheduler is adding and dispatching requests initially and at the end. We have a testing script, with a file named "yo", which was a test file with lorem ipsum text copied from here: `https://loripsum.net/` The test script was as follows:

```
#!/bin/bash

i="1"

while [ $i -lt 4 ]
do
cat yo > doy
done
```

It would run indefinitely until CTRL + C was hit.

4. What did you learn? We learned what the algorithms are for LOOK and C-LOOK I/O schedulers that are used to implement them, and also learning a bit on how they are coded. In general, we were able to learn some more about the Linux kernel, as well as getting used to working with the kernel more. We also learned about elevator algorithms and how they avoid the starvation problem, but are still more efficient than FIFO implementations.

1.2 Concurrency 2

1. What do you think the main point of this assignment is? The main point was to solve a concurrency problem using a different algorithm from the last one but still using the idea of concurrency. Learning more about different concurrency algorithms will help us be more familiar with the OS. This project helped us see concurrency in a different way, while still being able to use the C language to implement it.

2. How did your team approach the problem? Design decisions, algorithm, etc. This problem was initially tough because we had to ensure that philosophers would grab two forks simulatenously so that they all couldn't just grab one fork and lock each other out. This issue was resolved by having a "master" lock which would disable other philosophers from taking other forks (but they can still drop!) while a philosopher is grabbing forks.

3. How did you ensure your solution was correct? Testing details, for instance. Debugging was lots and lots of trace statements. Towards the end of the project, we had made a history and a table that would print out, showing the status of each thread and making sure the wait times are correct. We also checked mutex locks by commenting out unlock lines to see that the code would halt. This ensures that the resources are being locked by the threads.

4. What did you learn? We got more practice using mutexes and creating threads in C. We learned more on the idea of threads having to share resources but in a different scenario. Ideas like having all the threads share multiple items, rather than one item. We also learned about having every thread be able to run in a systematic way, where each thread is able to run when it is time to.

2 Version Control Log

3 Work log

Half of the project was completed on Thursday, October 25, 2018 for Dining Philosophers, and the other half on the due date (Monday, October 29, 2018). We saved the writeup and the coding for the I/O Elevators problem for today. That made our work log real spicy for the final day. Nickoli did most of the write-up and Ben did most of the coding for the I/O Elevators and Dining Philosophers problem. This worked out really well because Ben hates writing. Our group will continue this work pattern in the future.