Writeup 2

Alex Hoffer
Nehemiah Edwards

CS 444
Spring 2017

Abstract

This second write-up details Alex and Nehemiah's work on Project 2. Project 2 consisted of a concurrency exercise, a Linux patch file intended to redefine the noop scheduler as a C-LOOK scheduler, and this write-up.

Contents

1 Design plan for SSTF algorithms

We planned to modify the Linux distributed noop-iosched.c file to implement a C-LOOK scheduler instead of NO-OP. The algorithm will use an insertion sort to sort new requests that get added, while servicing requests that are larger than the disk head. When the disk head reaches the end, it will return to the beginning on continue again. A merge sort will add request into the back of the queue. The algorithm only scan for pending requests on the queue in one direction. Whether data is being read or being written and the position that's assigned to the disk head will be printed to the Linux Kernel. From the noop scheduler clook_dispatch(), clook_dispatch(), and noop_init_queue() will all be modified and "noop" will be changed to "sstf".

2 What we think the main point of this assignment is

We think there were three points to this assignment. One is to continue enhancing our concurrency abilities through the completion of the Dining Philosophers problem, which is a classic problem of concurrency. The second point is to see how the C-LOOK scheduler looks in practice, and how we can modify a Linux distribution to apply this scheduler. The third point is to continue to familiarize ourselves with the Yocto kernel, specifically how to modify the build conditions of the kernel, such as changing the default scheduler and running files from within the kernel.

3 How we personally approached the problems

3.1 Concurrency

To prepare for this problem, we looked into the Dining Philosophers problem which was invented by Dijkstra. We found that there were many different solutions to this problem since it was a classic one. We were choosing between Dijkstra's Resource Hierarchy solution and the Chandy-Misra solution, but we landed on using the Resource Hierarchy solution since it made the most sense to us. Then, we needed to choose a language to implement it in. We tried C++ so we could implement it in an object oriented fashion, but ran into some problems with this, specifically because we wanted to use pthreads and the pthread create function, which takes a void function, was giving us grief. So, we changed to C, where we were most familiar with the environment. We like the pthread library a lot because we are so used to it. From here, our approach to coding was that we read the Resource Hierarchy algorithm and used traditional pthread library methods like pthread create, pthread join, and mutexes to implement it so that each philosopher first grabbed the fork to their left, then waited on the one to their right, and if were unable to get both, gave up and waited. We used characters from the Marcel Proust novel "Remembrance of Things Past" as our philosophers, which was an important design decision.

3.2 C-LOOK Scheduler

To prepare for this problem, we went to class, where Kevin McGrath talked about different schedulers and discussed the noop and C-LOOK schedulers. Then, we read the TopHat text to understand these schedulers even better. In this text, there were links to files that corresponded to the Linux kernel. There was even a link to the noop scheduling file, which we used to peruse how we needed to change this file to a C-LOOK scheduler. Our approach to coding, then, was to copy the noop file into a new file and then observe the methods in the file further. In order to understand how a C-LOOK elevator algorithm should function, we looked into various websites and some documentation that was recommended by the TAs. From this information we obtained we got a good idea of how we needed to modify the noop file. This helped us figure out how we could integrate the necessary

sorting algorithms to modify the queue, and how we needed to use the position of the disk head for this type of scheduler.

4 How we ensured solution was correct

4.1 Concurrency

To ensure our concurrency solution was correct, we implemented a lot of print statements within the code to tell us where the philosopher was, what the philosopher was doing, and what the forks were doing and where they were. We made sure the names were very different so that it was easy to tell who was who. Then, we executed the file five times and traced what was going on. We found that we could prove that two philosophers were eating at any given time by our print statements, and we also found via these print statements that other philosophers were also able to eat, thus avoiding starvation. It was easy to tell that deadlock did not occur, since the program never ran infinitely.

4.2 C-LOOK Scheduler

To make sure the solution for the C-LOOK listener was correct, we created a script to test the algorithm we created on the VM. We also did a little bit of research to ensure that it was doing would it is expected to do. Since our kernel used printk() to print the sector number being accessed, we merely had to write a script that we could run from the virtual machine that would have a sequence of random I/O operations, like printf() and scanf(). We did this and found that the sectors that were accessed by the kernel made sense due to the print out.

5 What we learned

We learned a lot in the course of this assignment. For the concurrency, we learned more about how to implement Pthreads in C and more about the canonical Dining Philosophers problem and the different ways to solve it. For the VM build, we learned more about what the different yocto flags meant, especially the networking flag which let us bring in files we needed. We also learned about how to change the scheduler in the virtual machine and how to configure the virtual machine in general. We also learned how to generate a patch file and what a patch file is. Finally, we learned how to implement the C-LOOK scheduler, specifically what we would need to do to institute an elevator algorithm in the place of a single queue (noop algorithm).

6 Version control log

History of versions

Version	Date	Author(s)	Changes
1	4.27.2017	Alex	Experiment with C++ implementation of dining philosophers
2	4.27.2017	Alex	Finish philosophers problem in C
3	4.27.2017	Alex	add comments
4	4.27.2017	Alex	add I/O to prove correctness
5	4.27.2017	Alex	add more comments
6	5.1.2017	Alex	reformat old latex file
7	5.1.2017	Alex	add sections to writeup

8	5.3.2017	Nehemiah	add sections to writeup
9	5.3.2017	Alex	Add scheduler files
10	5.3.2017	Alex	add test script, finish scheduler

7 Work log

7.1 Week 1

Alex started working on this project by looking up how to solve the Dining Philosophers problem. He struggled with implementing it in C++ so did it in C using Pthreads.

7.2 Week 2

Nehemiah worked on changing the noop scheduler into the CLOOK scheduler. He succeeded. Alex worked on changing the default scheduler in yocto and began writing the Latex file. Finally, Alex made the patch file, changed the makefile and Kconfig file, and got everything working.