Sethu Senthil
sns5787@psu.edu

Extra Credit for Assigment 5

My internship was truly remarkable! I gained extensive knowledge in C programming, mastered the intricacies of pointers, and became proficient in Linux command-line operations I would love to get a return offer after I graduate and intern here again next summer.

Reading

Implementing reading was difficult at first as it was my first task in this environment and my first task like this in C. However I was able to eventually get it. I initially struggled to get reading across the disks working because I was not keeping track of the current dick or current block properly. However, after I figured out the problem about why my implementation was not working I was able to get it to work with no problems. Although it took me some extra time, beyond the due date for this part, I still ended up figuring it out at the end of the day. I did lose out on some points, however, there was little I could do considering the number of commitments I had with other classes and things beyond my control. One thing that helped me figure out the logic and algorithm for the code was to write it out in English. For MDAM Read it was as follows:

- **Validate input parameters**: ensure the JBOD is mounted, the starting address and read length are within valid ranges, and the read buffer is consistent with the read length.
- **Calculate disk and block indices**: determine the starting and ending disk and block indices based on the provided starting address and read length.
- **Perform data read operation**: iterate through the blocks, switching between cases based on the relative positions of the starting, current, and ending blocks.
- **Copy data to read buffer**: for each block, read the data into a temporary buffer, copy the relevant portion to the read buffer, and update read and pointer counters.
- **Return read length**: upon successful completion, return the actual amount of data read.

Writing

Implementing writing was not too difficult as at this point I was more comfortable with the environment as well as the mechanisms of C. Not to say that it was easy by any means, it was still a challenging task. However, with my newfound technique of bulleting the overarching idea

Sethu Senthil
sns5787@psu.edu

Extra Credit for Assigment 5

of the algorithm before actually writing the code helped me a ton. For writing it was the following logic:

- **Validate input parameters:** ensure write permission is granted, the starting address and write length are within valid ranges, and the write buffer is consistent with the write length.
- **Calculate starting and ending disk and block indices:** determine the disk and block indices based on the provided starting address and write length.
- **Initiate disk seeking:** seek the start disk using the jbod_client_operation function.
- **Seek the starting block**: use the seekToStartBlock(0 function to reach the specified starting block.
- **Iterate through blocks and write data:** loop until the entire write length has been processed.
- **Handle specific block positions:** utilize switch-case logic to efficiently handle different scenarios based on the relative positions of the starting, current, and ending blocks.
- **Update write pointer and counter:** increment the write pointer and update the write counter after each write operation.
- **Return written data length:** upon successful completion, return the actual number of bytes written.

Cache

Caching was pretty difficult. Although I had more experience now, I still had extreme difficulty figuring it out. Especially since it was harder to debug now, and the segmentation faults gave me nightmares! I still have not gotten my caching logic to work fully as it only works on one trace file. In fact for my assignment 5, due to this issue, I had to revert to my Assignment 3 MDADM which does not implement caching and works on all trace files. When implementing caching, to make the code more organized and sensible, I followed the DRY principles. DRY stands for “Don’t repeat yourself”, referring to the modularization of code so it can be reused rather than retyped again and again in different portions of the code. So I did exactly that, almost all my JBOD operation calls were wrapped in some higher-level abstracted helper function.

Sethu Senthil
sns5787@psu.edu

Extra Credit for Assigment 5

Reformatting the code like this also made it easier to implement cache as I was able to quickly modify all parts of the code where certain data is being read from disk, to check if the data is included in the cache before actually reading from disk without having to manually go to all my JBOD operation calls. This saved me a lot of time as well as made my code less error-prone.

Networking

Networking was by far the hardest portion of this project. Although I have a lot of experience with REST and HTTP clients, this portion of the assignment used a proprietary communication method which meant I had to code it up and understand it at a byte level. Not to mention, debugging was extra hard on this part of the assignment due to the nature of quitting and reopening the web server after each run as well as the methodology used to diagnose segmentation faults. I ended up writing a simple shell script to automatically run all the trace files for me which did save me some time when debugging. To add further complexity, pointers needed to be used in this part of the program more than they had before. This included memcpy() calls to copy memory into the right parts of the stack. Additionally, this part also involved properly using htons() to convert bytes from network format to “regular” format and vice versa.

Future Enhancements:

One future enhancement I would love to add is encryption. I feel like this is necessary when sending and receiving data in a digital age. I great way to get this started is to use the HTTP protocol, a popular standard for sending and receiving data with web servers. Currently, all data transferred between the server and the client is completely unencrypted which means if a bad actor were to either capture or overwrite the data in transit, it could pose a huge security risk. A great start would be to use the HTTPS protocol specifically the HTTP/3 with QUIC standard published in 2022. This would be the ideal solution as HTTP/3 is the latest standard with faster connection times and faster handshake times. This would inherently add delivery encryption through trusted certificates that follow the symmetric key cryptography concept. Additionally, the QUIC protocol takes care of lost packets efficiently as well as network changes (like from one wifi to another or wifi to data) by retaining the same connection ID. In other words, it would

Sethu Senthil
sns5787@psu.edu

Extra Credit for Assigment 5

make the program more resilient to unstable or subpar networks with large amounts of packet loss. All in all, encryption is a necessary enhancement for this program if it is being deployed in the real world.