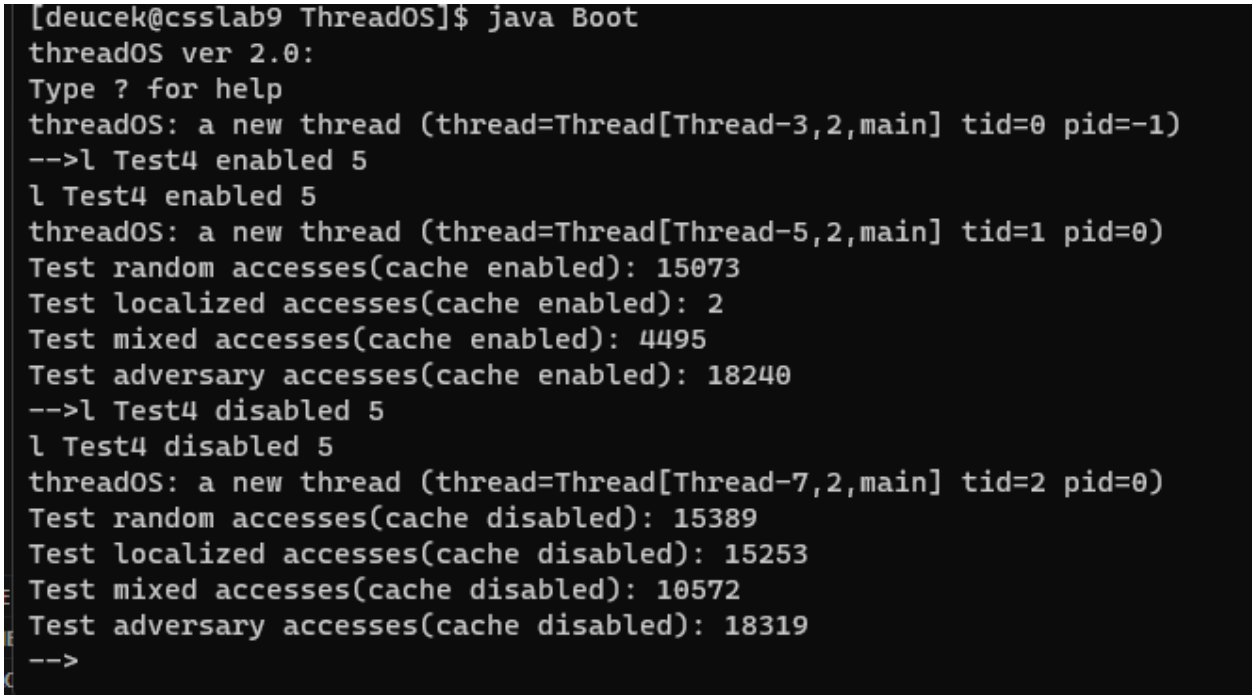
Caelen Cloud-Rouzan
CSS 430
05/26/2023

PROGRAM 4J REPORT

SPECIFICATION AND DESIGN ANALYSIS :

- Starting out with the given template the only things that needed to be implemented were the 5 functions that were left mostly blank and an additional private function that I felt was necessary.
    - **Cache** : this is our constructor that initiates our variables and assigns each Block a blockSize byte of data so that they can be used later in the code.
    - **Read** : First we assure the blockId is valid, then we try to find a valid page to use. If we find a valid one then we read the contents into buffer, and if we don't then we add it to the page table and load into main memory
    - **Write** : The checks and if statements for this and read are going to be the same, but once we reach the end when we find the page we are looking for we are going to be writing our data into the cache block instead of just reading what is already stored.
    - **Sync and flush** : These were already implemented with the given template file, with the code only taking a few lines and being very straightforward.
        - For the **sync** function we are iterating over the pageTable's contents and writing back all the dirty blocks.
        - For the **flush** function we are also iterating over the pageTable's contents and writing back all the dirty blocks. However we are also going to be wiping all the cached blocks while we are going through the contents.
    - **findInvalid** : this was the only function implemented that I felt was necessary outside of the required group. This privatized function iterates over the pageTable's contents and returns any invalid position.
- Overall, the purpose of this code is to demonstrate the performance between the different accessing methods and how implementing things like a buffer cache could change the effectiveness of disk caching.

PERFORMANCE AND ANALYSIS :



- **LOCALIZED ACCESSES** : This memory accessing method is the fastest due to it being the simplest and most reliable method out of the group. This approach was able to turn out a high ratio of cache hits for its designed test because of its sequential and quick turnaround of a new cache to test. Other methods have to do a lot more thinking/checks before they can deliver a cache to be used unlike this one.
- **MIXED ACCESSES** : This accessing method lands second, or right in between localized and random, because it is a mix of the two. Although it is mainly localized, it is a mix between the two which in turn makes it in between the two in performance as well.
- **RANDOM ACCESSES** : This approach is on the lower end of the performance testing results because of the verifying and random block grabbing processes that are involved. While the localized access is able to just give you the next block in the disk to test, this accessing method has to randomize a block to grab first and then verify its availability. This adds more time than some of the other accessing methods because they don't have to worry about things like this.
- **ADVERSARY ACCESSES** : This accessing performance was the slowest out of the group because of its inefficient approach of generating cache misses/disk accesses that do not make good use of the disk. Which in turn causes the performance to be much lower than the other methods.