Chad Dugie

David Trinh

CSS 430

11-29-15

Program 4 Report

# SPECIFICATION AND DESIGN

**Cache.java**

The cache class is designed to be used as a cache as part of ThreadOS. It uses the enhanced second chance algorithm using the reference bit and dirty bit that is stored with the private class Entry. Entry also has a block ID stored inside. The page table is created as an array of Entry classes.

When reading in Cache, the read method searches the page table for the block id to be read. When found, reference bit is set to true. Otherwise the method will continue searching for a free page to store the block id. If there is no free page then the method will look for a page to replace.

The same process occurs for writing, except the dirty bit is set to true when the block id has been written over.

Cache also has two methods called sync and flush. Sync maintains clean block copies by writing block id to disk and setting the dirty bit to false. Flush invalidates all cached blocks by going through each page in the page table and setting block id to -1 and reference to false.

**Test4.java**

Test4 has two byte arrays used as the write and read buffer. The constructor accepts only two arguments and the first argument must be either enabled or disabled, the second argument must be an integer from 1-4. If the arguments are not satisfied, then a runtime exception will occur. The arguments are stored in private variables to be used throughout the class

Test4 has a method called run(). This method contains a switch statement that depends on the argument given in the constructor. The four cases call on different methods to perform different tests. Random is case 1, localized is case 2, mixed is case 3, and adversary is case 4.

RandomAccess method uses the java library random to generate a random number to store into an array. The array and the write buffer is then written to the disk cache or disk depending on the argument. The read buffer and array is also read from the disk or disk cache depending on the argument.

Local access fills a write buffer and reads and writes to cache or disk more often than other methods.

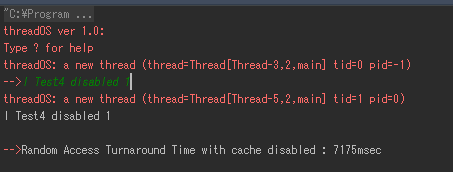
Mixed Access used random from the java library to generate 10% random accesses while having 90% localized accesses.

Adversary Access writes and reads to disk or disk cache, but does not read and write in sequential order. Instead, the method reads and writes several spaces ahead from the previous place that was written and read. This is to generate disk accesses that do not make good use of the disk cache.

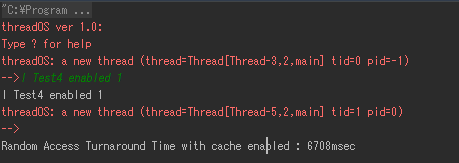
# pERFORMANCE

**Random Access**

With cache disabled: 7175msec



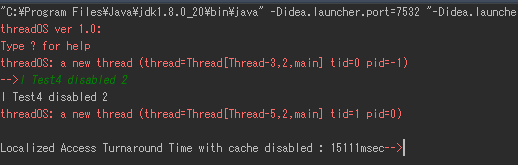
With cache enabled: 6708msec



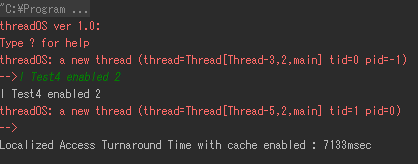
Consideration: With cache enabled, random access time is improved slightly. The time for both can vary by a few hundred msec due to the random nature of the method. For example, my second test run of random access yielded 6373 msec. That is a 467msec difference. Going with all random access does not seem to yield the best performance.

**Localized Access**

With cache disabled: 15111msec



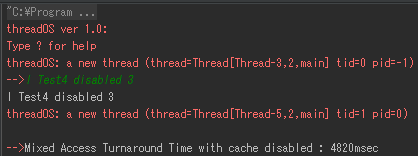
With cache enabled: 7133msec



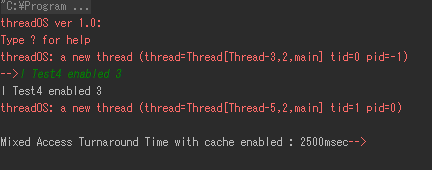
Consideration: With cache use, localized access is greatly improve, cutting turnaround time by roughly half. The turnaround time does not vary greatly either and if only off by a few msec. Without cache use, localized access can be extremely slow due to read and writing from disk instead of cache.

**Mixed Access**

With cache disabled: 4820msec



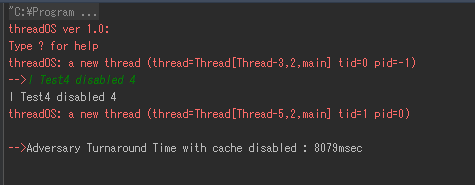
With cache enabled: 2500msec



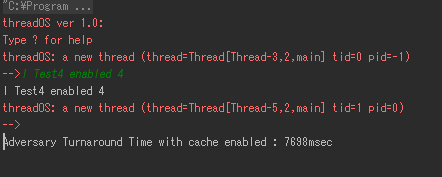
Consideration: With cache, mixed access is greatly improve. Turnaround time can vary due a few hundred msec due to the randomness implemented, but the use of cache still improves the time more greatly than random access.

**Adversary Access**

With cache disabled: 8079msec



With cache enabled: 7698msec



Consideration: Adversary access turnaround time is only slightly improved with cache enabled due to the poor use of cache space. The performance on adversary is similar to randomAccess in the amount of improvement done to turnaround time compared to without the cache enabled.

# Conclusion

From the test results, we have concluded that random access and adversary are not an efficient way of using a cache. Using mixed and all localized yielded the best turnaround time. Mixed access tended to vary more than localized when it came to turnaround time. Localized only varied by a few msec, while mixed varied by a few hundred per result.