Report

This homework aim to let us consider about how to avoid confusion when concurrency, and what the reflection of rebooting is.

In our project, we use lock to protect the file when one user process it. It’s that when one user (A) calls to use the file, we lock the associated object in system and no one else can draw it until A finish processing and release the lock. This will definitely avoid that one gets data from file while another one puts the same key data into the file, since if we don’t lock the file and let then happen at the same time, we cannot guarantee the these two processor will not mix up and interrupt each other.

Furthermore, we create two file officialFile and tempFile named cs542.db and tempFile.db respectively, so that we can make sure no error happened when synchronizing hashtable to file, since we use the tempFile to synchronize, even if the synchronic processing goes wrong, i.e. rebooting, we still keep the original final cs542.db, and just lose current process. In addition, tempFile can also be used to measure the length of file in case of oversize. Some more detail will be included in the function explanation.

Put function:

Lock the file when loading the file data into a hashtable, and put the new key- value pair into it. Here is the rule for putting new data that if there is any key in hashtable same as the adding key, previous value will be replaced by the new value, or creating a new line for new data. After putting new key-value pair into hashtable, we synchronize it to tempFile and judge tempFile by length. If the tempfile meets the length requirements, we exchange the name of tempFile and officialFile, else return "The file does not have enough room for data”. Finally release the lock.

Get function:

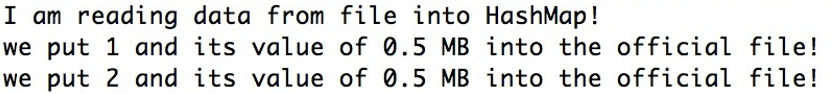
Lock the file when loading the file data into a hashtable, and retrieve the data for the given key. If we can find that key in the hashtable, then return the length of data for that key, else return "We did not find the key”. After that, release the lock.

Remove function:

Lock the file when loading the file data into a hashtable, and remove the data for the given key. If we can find that key in the hashtable, return “Successes”, or return “did not find”. After that, synchronize new hashtable to tempFile and exchange name with officialFile meanwhile update the length of 542.db. Then release the lock.

Test:

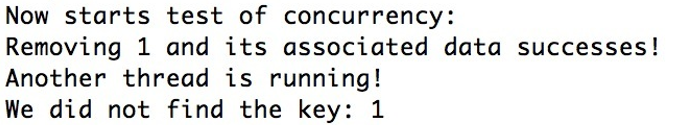
We first put two data pair with key = 1, 2 and length = 0.5M, 0.5M respectively into file as the original officialFile cs542.db and then test concurrency and fragmentation.



Concurrency test:

We do the test: when one caller does a Remove() and another caller does a Get() with the same key a millisecond later.

In the test code, we use thread to control function remove(1) and get(1) happen with a millisecond later. And the result shows as following:



We can find that this two processor do not interrupt each other, it let the remove happen before getting value. We guarantee concurrency successfully.

Durability test:

Fragmentation test:

We do the test: Put() 4 values, byte arrays of 1 MB each, with keys A, B, C and D. Remove key B.Put() ½ MB in size for key E. Validate that a Put() 1 MB in size for key F fails. Remove C and now validate that a Put() 1 MB in size for key G succeeds. Remove E and try Put() 1 MB in size for key H. With a naive implementation, it will fail even though there is room in store.db. An extra bonus point if you can modify your code such that Put("H", …) succeeds.

The resulting is showing as following, we do every process successfully.

