Operating System: Project 4

Instructed by *Wei Xu*

Due on Jan 8, 2021

Task 2

To make sure that only one thread can access the segment buffer and segment bitmap, we use a mutex lock named `segment_lock`. Threads intend to enter `get_block()`, `new_data_block()`, `new_inode_block()` and `remove_inode()` have to acquire `segment_lock` first and then release it before returning.

To maintain the consistency of the whole file system, we use locks for each independent inode. In each file operation, the thread needs to first locate the inodes it need to access, and then acquire all these locks corresponding to them (in the order from small inumbers to large inumbers).

To make the cache operations atomic, we use a mutex lock for all the cache operations.

Task 3

The write-back cache overrides our original `read_block()` and `write_segment()` functions with their `_through_cache()` versions. The cache occupies 4MB space with 512 cachelines, and each cacheline contains contiguous 8 blocks. Upon any cache miss, LRU policy is applied to decide to-be-deleted cachelines. When an eviction finishes, all evicted dirty cachelines are `fsync()`ed to disk.

The cache also provides `init_cache()` and `flush_cache()` functions to support file system initialization and garbage collection.

Task 4

The garbage collection mechanism may be automatically triggered whenever LFS asks for a free new segment. For simplicity, we consider three levels of "fullness" that may trigger such a mechanism:

- **Largely full**: when 80% of the segments are full. In this case, the *normal garbage collection* procedure will be performed, where *the least utilized blocks are cleaned*. The normal mode guarantees that at least 30% of the segments are cleaned, and those with utilization smaller than 1% will also be cleaned.
- **Almost full**: when 96% of the segments are full. In this case, the *thorough garbage collection* procedure will be performed, where *all segments will be scanned, cleaned, and reordered in disk file*.
- **Completely full**: when 100% of the segments are full. In this case, the same *thorough garbage collection* procedure will be performed. When it fails to release much space, LFS will report itself as full, and stop creating new segments (read and deletion are still applicable, in most cases).

To prevent performing garbage collection too frequently, we set a lower bound `GARBCOL_INTERVAL` for the same mode of *failed collections*; i.e., once some collection fails, the same procedure will not be repeated until `GARBCOL_INTERVAL` seconds later.

For the concurrent version, when we start the garbage collection procedure, we will redirect all new blocks to a temporary log queue in memory, which will be flushed to disk as soon as the garbage collection procedure completes. The semantics of a logged system naturally guarantees the correctness of writing concurrency. For the reading part, we will read the disk file as usual, since the contents will not be replaced until the end of garbage collection.

We provide two heavily-loaded tests (`testGC1.cpp` and `testGC2.cpp`) for garbage collection.