Synchronization Design

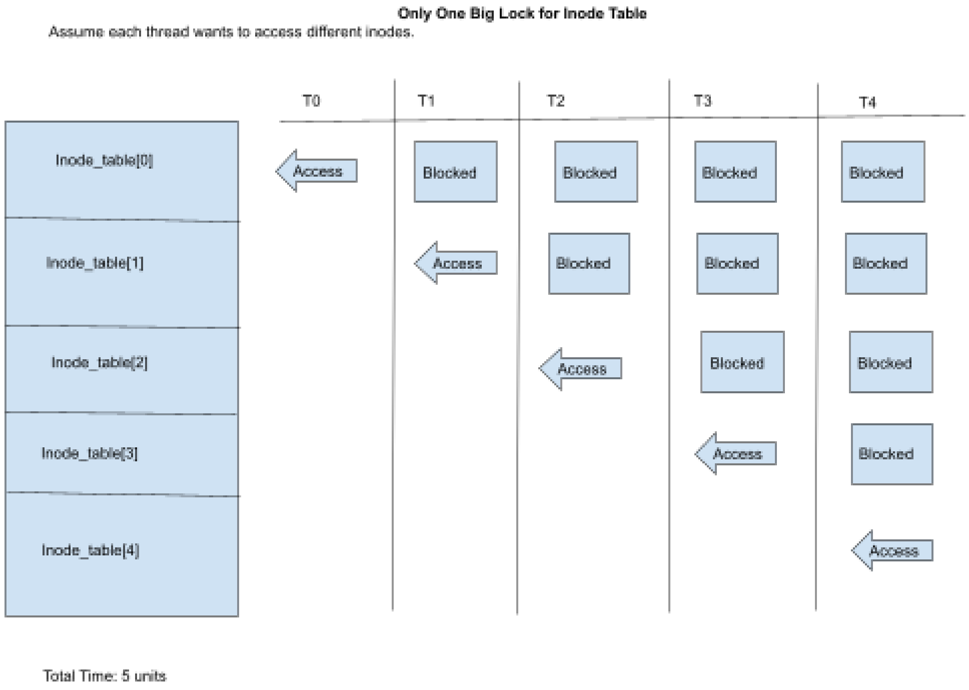
Choice of Lock: Mutex

Initially, I was deciding between mutex locks and read/write locks. The reason that I used mutex lock over read/write lock was that after going over my overall designs, I noticed that I had more writing operation than reading operations. Therefore, I decided to use mutex lock to avoid possible writer starvation

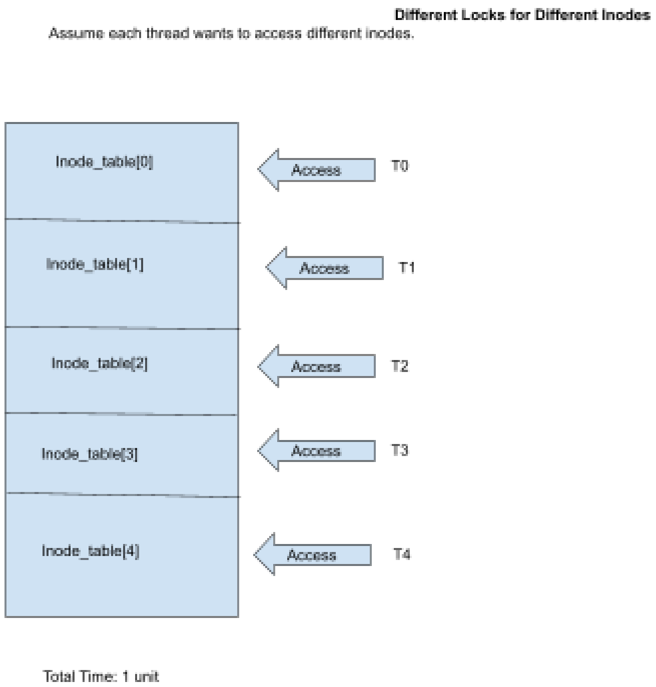
Locks:

1. block_bitmap_lock
2. inode_bitmap_lock
3. sb_lock (super block lock)
4. gd_lock (group descriptor lock)
5. inode_table_lock[]

Above locks protect all major metadata and shared data about the file system among the threads. The first four locks are pretty self-explanatory. The inode_table_lock[] is an array of mutex locks for every inode in the inode table to avoid big locks. If we only have one mutex lock for the entire inode table, then threads access to different inodes will be blocked inefficiently, which will cause worse parallelism. Below are diagrams showing the difference.



(Figure 1, one big lock for inode table)



Different Locks for Different Inodes

Assume each thread wants to access different inodes.

Total Time: 1 unit

(Figure 2. Different Locks for Different Inodes)

Prevention for Deadlock

To avoid deadlock, I ruled out the lock acquire order for all functions in the following order:

block bitmap
inode bitmap
group descriptor
superblock

Inodetable_lock[i] should be added depending on situation