Lab3 报告

518021910531 徐珺涵

part1

- **实现：**
  - **哈希函数：**
    首先将作为 `string` 传过来的 `key` 分解为一个一个的字母并转化为 `unsigned long long` ，为了防止字母之间由于差距比较小而导致加起来一样，还要带上每个字母所在位置的信息，即乘以 `(i + 1)` ，并且由于人们命名习惯为字母加上数字，所以数字和字母要分开对待，最后因为初始化的时候inode的值要限定在2~1023以内，所以 `hash` 函数在返回前要对1024取模。
  - **其他操作 `get/set/del` ：**
    在part1中这三个操作的流程类似，首先对 `key` 进行哈希，然后使用 `ec->getattr` 确定要访问元素的状态是否正确，如果正确则调用 `ec->get/put/remove` 进行操作，最后返回OK。

part2

- **实现的主要思想：**
  - **锁、信号与进程信息的储存：**
    由于lc的局限性，并且还要调用额外的接口，费时费力，所以决定直接在 `ydb_server_2pl` 中使用 `pthread_mutex_t` 为锁，并且使用一个 `pthread_cond_t` 数组作为信号，具体使用方法会在后面详细讲解。
    同时对每个元素还有一个 `map` 记录，其中 `map` 以 `inode` 值为索引， `value` 值为布尔型变量。
    同时使用一个二维 `map` 对进程信息进行存储，第一层 `map` 的索引为 `transaction_id` ，第二层 `map` 索引为 `unsigned long long` 代指在 `ec` 中的 `inode` 号， `value` 值为一个结构体，包含该操作信息与操作目的值，以及是否已经拿到数据锁。
  - **检查死锁函数：**
    使用递归的方法，首先找到本次参数 `current` 对应的需要拿到锁元素，并且在 `map` 表中寻找已经拿到该锁的进程，易知这样的进程只有一个，找到的进程即本层递归所依赖的进程，同时每层递归都会传递一个原始进程(第一次递归的 `current` ，也就是找依赖的起点)，如果发现依赖的进程与原始进程是一个进程，那么出现死锁，返回 `true` ，否则返回 `false` 。
  - **检查id合法性：**
    除了 `begin` 操作外都要在函数一开始就进行检查，如果进程号在 `map` 中不存在即说明不合法。
  - **上锁、放锁与唤醒**
    前面提到的 `pthread_cond_t` 数组正对应着每个可能的inode号，如果一个进程因为拿不到第i个元素的锁而陷入等待，那么将进入 `pthread_cond_wait(&cond[i], &mutex)` 状态，等到拿着该锁

的进程开始放锁时，会对每个拿到的锁调用一个 `pthread_cond_signal(&cond[i])` 函数进行一次唤醒，这样等待的进程就会醒来了。

- **begin**
  先上锁，然后在 `map` 表中以 `current_id` 新建一个空的元素，然后 `current_id++` 后放锁。
- **commit**
  上锁，首先将要 `commit` 对应进程的所有写操作和删除操作写入 `ec` 中，然后对等待的进程进行依次唤醒并放锁，最后将完成的进程从 `map` 中抹去。
- **abort**
  上锁，对等待的进程进行依次唤醒并放元素锁，最后将 `abort` 的进程从 `map` 中抹去。
- **get/put/del**
  首先拿到 `mutex` ，然后检查自己对应进程的map里面是否有本次读/写/删除目的元素的记录，如果有，那么写/删除可以将其覆盖掉(因为此时自己肯定是持有着该元素的锁的)，读的话直接读出自己上次读的值并返回就可以了。
  如果没有则接着检查要拿该元素的话时候会产生死锁，若产生则自身 `abort` ，否则拿到该元素的锁后，读直接读出来并在map中记录，写和删除先不进入到 `ec` 中，等到 `commit` 再放入 `ec`，放掉 mutex`并返回。

part3

- **实现的主要思想：**
  - **储存数据结构**
    进程保存的数据结构还是和2pl一样，使用二维的map。增加了一个 `map` 记录了对应 `inode` 的版本号，版本号从0开始增加，由于每次对比仅对比版本号是否一致，所以不用考虑溢出整数，同时也有 `pthread_mutex_t` 用来维护版本号修改与获取的原子性。
  - **版本号**
    因为乐观的人不想对元素进行上锁，所以每次进程想要使用某个元素时，只需要把自己使用时该元素的版本号保存下来，当然为了获取正确的版本号，需要在获取版本号的过程中、以及改变版本号的过程中加锁，等到进程 `commit` 的时候会逐个元素对比自己保存的版本号和 `server` 保存的，如果自己保存的和 `server` 不一致，那么说明有其他进程的元素和自己冲突并且先于自己 `commit` 了，那么这个进程就会 `abort` 。
  - **begin**
    在map中增加一个空元素。
  - **commit**
    上锁，逐一对比要提交进程的版本号是否冲突了，若冲突则 `abort` ，否则将进程中 `set/del` 的操作执行进 `ec` 里，并增加把 `server` 端的版本号加一，放锁。
  - **abort**
    删除该进程在 `map` 中的位置。
  - **get/put/del**
    首先检查进程本身是否之前操作过该元素，若是，则不需要更新版本号，若是读操作则直接返

回原来的记录，若是写/删除则直接改变自己的记录。若之前没有记录，那么上锁并保存要操作元素的版本号，将此次操作记录下来，放锁。