Lab 1~5 @ MIT 6.824

By 章明星

参考

- 参考了部分 GitHub 上的实现
  - ahaven / 6.824-Project
  - ichaos / mit-6.824-2012 (有错)
  - …
- 我的也在 GitHub 上可以查看
  - james0zan / mit6.824

Step 1 @ Lab 1: Lock Server

- 按照提示给 lock_server 添加一个 map 记录锁与持有者的对应关系

- 具体实现中有两种策略
  - 为每一个锁都配一个条件变量
  - 只用一个条件变量

Step 1 @ Lab 1: Lock Server

- 注意一：编译不通过

  如果提示 rpc.cc 中 getpid 未定义的话可以通过添加 unistd.h 解决

  # 从 gcc 4.7 开始不再自动 include 这个头文件了

- 注意二：测试不完全

  lock_tester 没有测试“尝试释放自己并没有持有的锁”这种情况 （ ichaos 就错在这）

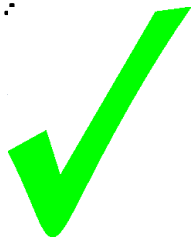
Step 1 @ Lab 1: Lock Server

- 注意三：条件变量的使用

  pthread_cond_broadcast 这个函数的作用是唤醒当前所有等待在这个条件变量上的线程

  所以：

  Wai



  Boardcast

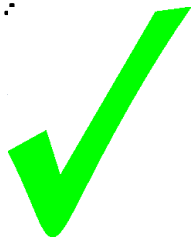
Step 1 @ Lab 1: Lock Server

- 注意三：条件变量的使用

  pthread_cond_broadcast 这个函数的作用是唤醒当前所有等待在这个条件变量上的线程

  所以：

Wai



Boardcast

Boardcast

Wait

Step 2 @ Lab 1: Sliding Window

- 对于 add_reply 函数

  在记录了之前回答的 reply_window_[clt_nonce] 中查找相应的 xid，如果找到了就记录起来

  # 正常情况下是不会找不到的

Step 2 @ Lab 1: Sliding Window

- 对于 add_reply 函数
- 对于 checkduplicate_and_update 函数
  - 将 reply_window_[clt_nonce] 中 xid 比 xid_rep 小的都删掉
  - 在 reply_window_[clt_nonce] 中寻找 xid
    - 找到的话返回 cb_present ? DONE : INPROGRESS
    - 没找到的话就添加这个 xid

Step 2 @ Lab 1: Sliding Window

- Tip

  仔细看函数前的注释

Step 1 @ Lab 2: 文件创建

- Extend Server

  方便起见用的和 ahaven 一样的内存模拟

  文件直接用如下的结构表示

```
struct file {
    std::string str;
    extent_protocol::attr attr;
};
```

Step 1 @ Lab 2: 文件创建

- Extend Server
- 目录的格式

  还是和 ahaven 一样用

  文件名 \0 eid \0

  文件名 \0 eid \0

  …

  虽然限制了文件名里不能用 \0 不过按道理来说本来就不可以，加上用 sstream 处理起来很方便

Step 1 @ Lab 2: 文件创建

- Extend Server
- 目录的格式
- create/lookup/readdir

  纯码农活，按照注释实现功能就好

  注意点：

  - 初始化的时候就要先建立一个 eid 为 1 的根目录
  - create 前先查一下目录里有没有同名的
  - eid 格式

Step 2 @ Lab 2: 文件读写

没什么特别需要说的，只需注意其实完全不用知道 fuse 是怎么工作的，在 fuse.cc 里搜“// You fill this in for Lab 2”然后把空填完就行。

Step 1 @ Lab 3: mkdir/unlink

- mkdir

  可以重用 create 中大部分代码

- unlink

  也和 create 差不多其实

Step 2 @ Lab 3: 锁

- 虽然根据原有代码中的提示，这里应该用和 ahaven 一样用 goto 语句实现释放锁，不怎么优美
- 实现一个简单的 ScopeLock

```
struct myScopeLock {
  lock_client *lc;
  yfs_client::inum ino;
  myScopeLock(lock_client *lc_, yfs_client::inum ino_) {
    lc = lc_; ino = ino_;     lc->acquire(ino);
  }
  ~ myScopeLock() { lc->release(ino);  }
};
```

Step 2 @ Lab 3: 锁

- 注意：锁要加齐

  由于 Extend Server 那边也用了锁所以 yfs_client 这边很多地方可以不用加。

  但是：

  1）后面做内容 cache 的时候的一致性

  2）潜在的 data race

  还是都加上的好

Lab 4: Lock Cache

- Server 端数据结构

```
std::map<std::string, rpcc*> clients;
struct lock {
  bool used;
  std::string x;
  std::set<std::string> waited;
};
std::map<lock_protocol::lockid_t, lock> possessed;
```

Lab 4: Lock Cache

- Server 端实现
  - Release
    - 合法性判断
    - 如果没有人等待这个锁的话返回
    - 不然调用等待者其中之一的 retry 函数

Lab 4: Lock Cache

- Server 端实现
  - Release
  - Acquire
    - 如果被占用
      - 插入等待列表
      - 调用占用者的 revoke 函数

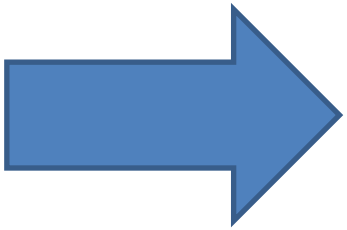
Lab 4: Lock Cache

- Server 端实现
  - Release
  - Acquire
    - 如果被占用
      - 插入等待列表
      - 调用占用者的 revoke 函数
    - 如果没有被占用
      - 返回 OK 将锁交给申请者
      - 如果此时等待列表中还有其它客户端，立即调用申请者的 revoke 函数

Lab 4: Lock Cache

- Client 端状态设计

NONE
FREE
LOCKED
ACQUIRING
RELEASING



NONE
FREE
LOCKED
bool revocked

Lab 4: Lock Cache

- Client 端数据结构

```
enum CS {NONE, FREE, LOCKED};
struct lock {
  int stat;
  bool revoked;
  pthread_t id;
  sem_t cond;
};
pthread_mutex_t mutex;
std::map<lock_protocol::lockid_t, lock> cache;
```

Lab 4: Lock Cache

- Client 端实现
  - Acquire
    - FREE： 直接付给它
    - LOCKED： wait 在相应的信号量上
    - NONE：
      - 调用 server 端的 acquire
      - 返回 OK 则切换到 LOCKED 状态
      - 返回 RETRY 就 wait

Lab 4: Lock Cache

- Client 端实现
  - Acquire
  - Release
    - 检查合法性
    - 如果 revocked == false 切换到 FREE 状态
    - 不然就调用 server 端的 release
    - sem_post （唤醒本地那些因为原来是LOCKED 状态所以等待的线程

Lab 4: Lock Cache

- Client 端实现
  - Acquire
  - Release
  - Revoke
    - 如果是 FREE 就调用 server 端的 release
    - 不然置 revocked 为 true

Lab 4: Lock Cache

- Client 端实现
  - Acquire
  - Release
  - Revoke
  - Retry
    - sem_post

Lab 4: Lock Cache

- 为什么不用条件变量

  client 在调用 server 的 rpc 或 wait 时是不持有全局锁的。所以可能出现下面这种情况：

CThread 1	CThread 2	Server
调用 acquire		
		收到 acquire 返回 retry
	调用 release	
		收到 release调用 Thread 1 的 retry
收到 retry 调用		
收到 acquire 返回的 retry		

Lab 5: Extend Cache

- Step 1
  - 添加一个 map 然后所有操作只对这个 cahce map 进行就好
  - 要有一个 bool 变量记录是否进行了写操作
  - 如果 remove 也 cache 的话还要有一个 bool 记录是否删掉了
- Step 2
  - 在 lock_client_cache.cc 中所有调用 server 的 release 的地方之前加一个 lu->dorelease(lid)
  - dorelease 的具体实现为如果脏了就回写

Q & A