Lab 6：Cache+Log

本次实验将实现缓存和日志机制。

参考资料：

1. 实验课 slides
2. The xv6 book，第 8 章，“File system”

你需要实现 `src/fs/cache.c` 中的所有 “`TODO`”。我们建议你在动手实现之前阅读 `src/fs` 目录下的 `cache.h` 及其它头文件中的注释说明。

测试

1. `make qemu` 能够正常运行，并且 lab 5 的 `sd_test` 能够通过
2. 通过 `cache_test`：

```
$ cd src/fs/test
$ mkdir -p build; cd build
$ cmake ..
$ make && ./cache_test
```

如果遇到编译错误，请尝试运行 `make clean && make`。如果依然不能解决问题，请尝试删除 `build` 文件夹后重试以上步骤。

通过标准：最后一行出现 “`(info) OK: 23 tests passed.`”。此外助教会通过测试的输出判断测试是否在正常运行。

以下是助教的实现的输出：

```
(info) "init" passed.
(info) "read_write" passed.
(info) "loop_read" passed.
(info) "reuse" passed.
(debug) #cached = 20, #read = 154
(info) "lru" passed.
(info) "atomic_op" passed.
(fatal) assertion failed: "i < OP_MAX_NUM_BLOCKS"
(info) "overflow" passed.
(info) "resident" passed.
(info) "local_absorption" passed.
(info) "global_absorption" passed.
(info) "replay" passed.
(fatal) cache_alloc: no free block
(info) "alloc" passed.
(info) "alloc_free" passed.
(info) "concurrent_acquire" passed.
(info) "concurrent_sync" passed.
(info) "concurrent_alloc" passed.
(info) "simple_crash" passed.
(trace) running: 1000/1000 (844 replayed)
(info) "single" passed.
(trace) running: 1000/1000 (224 replayed)
(info) "parallel_1" passed.
(trace) running: 1000/1000 (168 replayed)
(info) "parallel_2" passed.
(trace) running: 500/500 (221 replayed)
(info) "parallel_3" passed.
(trace) running: 500/500 (229 replayed)
(info) "parallel_4" passed.
(trace) throughput = 20751.62 txn/s
(trace) throughput = 21601.20 txn/s
(trace) throughput = 18314.84 txn/s
(trace) throughput = 21231.88 txn/s
(trace) throughput = 18097.45 txn/s
(trace) throughput = 21225.89 txn/s
(trace) throughput = 18049.48 txn/s
(trace) throughput = 21035.98 txn/s
(trace) throughput = 17995.50 txn/s
(trace) throughput = 21220.89 txn/s
(trace) throughput = 17972.01 txn/s
(trace) throughput = 20875.56 txn/s
(trace) throughput = 18040.48 txn/s
(trace) throughput = 21422.29 txn/s
(trace) throughput = 18128.94 txn/s
(trace) throughput = 21169.92 txn/s
(trace) throughput = 18037.98 txn/s
(trace) throughput = 21279.36 txn/s
(trace) throughput = 18271.36 txn/s
(trace) throughput = 21325.34 txn/s
```

```
(trace) throughput = 17884.06 txn/s
(trace) throughput = 21182.41 txn/s
(trace) throughput = 18100.45 txn/s
(trace) throughput = 21194.90 txn/s
(trace) throughput = 18414.29 txn/s
(trace) throughput = 20993.50 txn/s
(trace) throughput = 18106.95 txn/s
(trace) throughput = 21266.87 txn/s
(trace) throughput = 17812.59 txn/s
(trace) throughput = 21317.34 txn/s
(trace) running: 30/30 (7 replayed)
(info) "banker" passed.
(info) OK: 23 tests passed.
```

以上输出无需完全一样。注意中间出现了两个“`(fatal)`”，是正常现象。

作为参考，助教的助教的实现在服务器上测试耗时 329 秒。

提示

推荐完成顺序及相关提示（不搞猜谜，没说清楚来问助教）

- 先行阅读cache.h,以及sblock的结构定义。
- slides也是非常重要的参考内容
- LRU Cache部分
    - cache_acquire
        - 需要的block如果在cache中我们需要做什么?
        - cache中block的数目有上限吗?
            - 其实是有“软约束”，类似“非必要不超过”
        - 如何减少block数目?
            - 将某些block uncache，那么是哪些?
        - cache所用的空间从哪来?
    - cache_release
        - 是否意味着这个block可以不用放在cache?
    - get_num_cached_blocks

- Log部分
  - cache_begin_op
    - 看ppt 30，31，begin_op需要等待什么?
  - cache_sync
    - ctx用于标记op，ctx==null代表直接写磁盘
    - 否则处于事务中，应该在commit时写到磁盘上
    - 事务能“持有”的block数目有上限，怎么体现?
    - 经过sync的block内容与磁盘不一致，意味着什么?
  - cache_end_op
    - ppt 31提到，最后一个op负责commit，如何标识?
    - 其他的end_op需要在commit->checkpoin完成后才能返回，如何实现?
    - commit->checkpoint的细节过程见ppt 32
  - init_bcache
    - 系统崩溃后，再启动时，log区的数据是需要重新写入磁盘的。
- Bitmap管理部分
  - cache_alloc
  - cache_free

提交

提交：将实验报告提交到 elearning 上，格式为`学号-lab6.pdf`。

从lab2开始，用于评分的代码以实验报告提交时为准。如果需要使用新的代码版本，请重新提交实验报告。

截止时间：2022年11月9日 19:30。逾期提交将扣除部分分数。

报告中可以包括下面内容

- 代码运行效果展示（测试通过截图）

- 实现思路和创新点
- 对后续实验的建议
- 其他任何你想写的内容

报告中不应有大段代码的复制。如有使用本地环境进行实验的同学，请联系助教提交代码（最好可以给个git仓库）。使用服务器进行实验的同学，助教会在服务器上检查，不需要另外提交代码。