Lab 3 : MKDIR, UNLINK, and Locking

Introduction

In this lab, you will:

- Add handlers for the MKDIR and UNLINK FUSE operations,
- Add locking to ensure that concurrent operations to the same file/directory from different yfs_clients occur one at a time.

Getting Started

First, merge your solution to Lab 2 with the new code for Lab 3. The only interesting difference are some new test scripts. You can use the following commands to get the Lab 3 sources.

```
% cd lab
% git commit -am 'my solution to lab2'
% git checkout lab3
%git merge lab2
```

As before, if git reports any conflicts, edit the files to merge them manually, then run git commit -a. Since you are building on the previous labs, ensure the code in your Lab 3 directory passes all tests for Labs 1 and 2 before starting this lab.

Part 1 : MKDIR, UNLINK

Part 1: Your Job

Your job in part 1 is to handle the MKDIR and UNLINK FUSE operations. Make sure that when you choose the inum for a new directory created with MKDIR, that inum has its most significant bit set to 0 (as explained in Lab 2). For MKDIR, you do not have to create "." or ".." entries in the new directory since the Linux kernel handles them transparently to YFS. UNLINK should always free the file's extent; you do not need to implement UNIX-style link counts.

When you're done with Part 1, the following should work:

```
% ./start.sh
% mkdir yfs1/newdir
```

```
% echo hi > yfs1/newdir/newfile
% ls yfs1/newdir/newfile
yfs1/newdir/newfile
% rm yfs1/newdir/newfile
% ls yfs1/newdir
% ./stop.sh
```

If your implementation passes the test-lab-3-a.pl script, you are done with part 1. The test script creates a directory, creates and deletes lots of files in the directory, and checks file and directory mtimes and ctimes. Note that this is the first test that explicitly checks the correctness of these time attributes. A create should change both the parent directory's mtime and ctime. Here is a successful run of the tester:

```
% ./start.sh
% ./test-lab-3-a.pl ./yfs1
mkdir ./yfs1/d3319
create x-0
delete x-0
create x-1
checkmtime x-1
...
delete x-33
dircheck
Passed all tests!
% ./stop.sh
```

Part 2 : Locking

Next, you are going to ensure the atomicity of file system operations when there are multiple yfs_client processes sharing a file system. Your current implementation does not handle concurrent operations correctly. For example, your yfs_client's create method probably reads the directory's contents from the extent server, makes some changes, and stores the new contents back to the extent server. Suppose two clients issue simultaneous CREATEs for different file names in the same directory via different yfs_client processes. Both yfs_client processes might fetch the old directory contents at the same time and each might insert the newly created file for its client and write back the new directory contents. Only one of the files would be present in the directory in the end. The correct answer, however, is for both files to exist. This is one of many potential races. Others exist: concurrent CREATE and UNLINK, concurrent MKDIR and UNLINK, concurrent WRITEs, etc.

You should eliminate YFS races by having yfs_client use your lock server's locks. For example, a yfs_client should acquire a lock on the directory before starting a CREATE, and only release the lock after finishing the write of the new information back to the extent server. If there are concurrent operations, the locks force one of the two operations to delay until the other one has completed. All yfs_clients must acquire locks from the same lock server.

Part 2 : Your Job

Your job is to add locking to yfs_client to ensure the correctness of concurrent operations. The testers for this part of the lab are test-lab-3-b and test-lab-3-c, source in test-lab-3-b.c and test-lab-3-c.c. The testers take two directories as arguments, issue concurrent operations in the two directories, and check that the results are consistent with the operations executing in some sequential order. Here's a successful execution of the testers:

```
% ./start.sh
% ./test-lab-3-b ./yfs1 ./yfs2
Create then read: OK
Unlink: OK
Append: OK
Readdir: OK
Many sequential creates: OK
Write 20000 bytes: OK
Concurrent creates: OK
Concurrent creates of the same file: OK
Concurrent create/delete: OK
Concurrent creates, same file, same server: OK
test-lab-3-b: Passed all tests.
% ./stop.sh
%
% ./start.sh
% ./test-lab-3-c ./yfs1 ./yfs2
Create/delete in separate directories: tests completed OK
% ./stop.sh
```

If you try this before you add locking, it should fail at "Concurrent creates" test in test-lab-3-b. If it fails before "Concurrent creates", your code may have bugs despite having passed previous testers; you should fix them before adding locks.

After you are done with part 2, you should also test with test-lab-3-a.pl to make sure you didn't break anything. You might also test test-lab-3-b with the same directory for both arguments, to make sure you handle concurrent operations correctly with only one yfs_client before you go on to test concurrent operations with two yfs_clients.

Part 2 : Detailed Guidance

- What to lock?

At one extreme you could have a single lock for the whole file system, so that operations never proceed in parallel. At the other extreme you could lock each entry in a directory, or each field in the attributes structure. Neither of these is a good idea! A single global lock prevents concurrency that would have been okay, for example CREATEs in different directories. Fine-grained locks have high overhead and make deadlock likely, since you often need to hold more than one fine-grained lock.

You should associate a lock with each inumber. Use the file or directory's inum as the name of the lock (i.e. pass the inum to acquire and release). The convention should be that any yfs_client

operation should acquire the lock on the file or directory it uses, perform the operation, finish updating the extent server (if the operation has side-effects), and then release the lock on the inum. Be careful to release locks even for error returns from yfs_client operations.

You'll use your lock server from Lab 1. yfs_client should create and use a lock_client in the same way that it creates and uses its extent_client.

- Things to watch out for:

This is the first lab that creates files using two different YFS-mounted directories. If you were not careful in earlier labs, you may find that the components that assign inum for newly-created files and directories choose the same identifiers. One possible way to fix this may be to seed the random number generator differently depending on the process's pid. The provided code has already done such seeding for you in the main function of fuse.cc.

This is also the first lab that writes null ('\0') characters to files. The std::string(char*) constructor treats '\0' as the end of the string, so if you use that constructor to hold file content or the written data, you will have trouble with this lab. Use the std::string(buf, size) constructor instead.

Handin procedure

E-mail your code as a gzipped tar file to TA and the teacher by the deadline mentioned on the course web. To do this, execute these commands

```
% cd ~/lab
% ./stop.sh
% make clean
% rm core*
% rm *log
% cd ..
% tar czvf your-student-d-your-name-lab3.tar.gz lab/
```