Read Me

(You may also refer to the test and result evaluation section in the pdf file for more demonstration)

1. **Mount the filesystem**

The local directories can be mounted by following the steps below:

1. First we need to start the server using the following command:

*./server.py*

2. Then in two other terminals we mount the directories (t1, t2):

*python client1.py t1 "http://localhost:51233"*

*python client2.py t2 "http://localhost:51233"*

Note that we use "local host" as the IP numbers, "51233" as the port number of the server, and "51235" and "51236" of the two clients.

1. **Test the multithreading functions**

**1. Create a file called "testfile" in the FUSE Filesystem:**

*cd t1*

*touch testfile*

and schedule two concurrent shell commands by using the command:

*crontab -e*

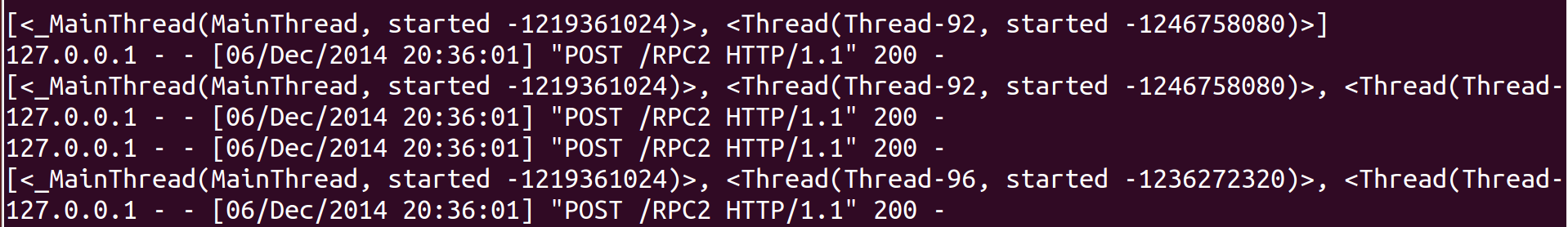
Schedule two concurrent echo operations by two clients to the same file in crontab, fill in the time and path information of your own:

m h dom mon dow command

*56 14 \* \* \* echo "testfile write 1 by client1">/home/ud/pocsd/fusepy/t2/testfile*

*56 14 \* \* \* echo "testfile write 1 by client2">/home/ud/pocsd/fusepy/t1/testfile*

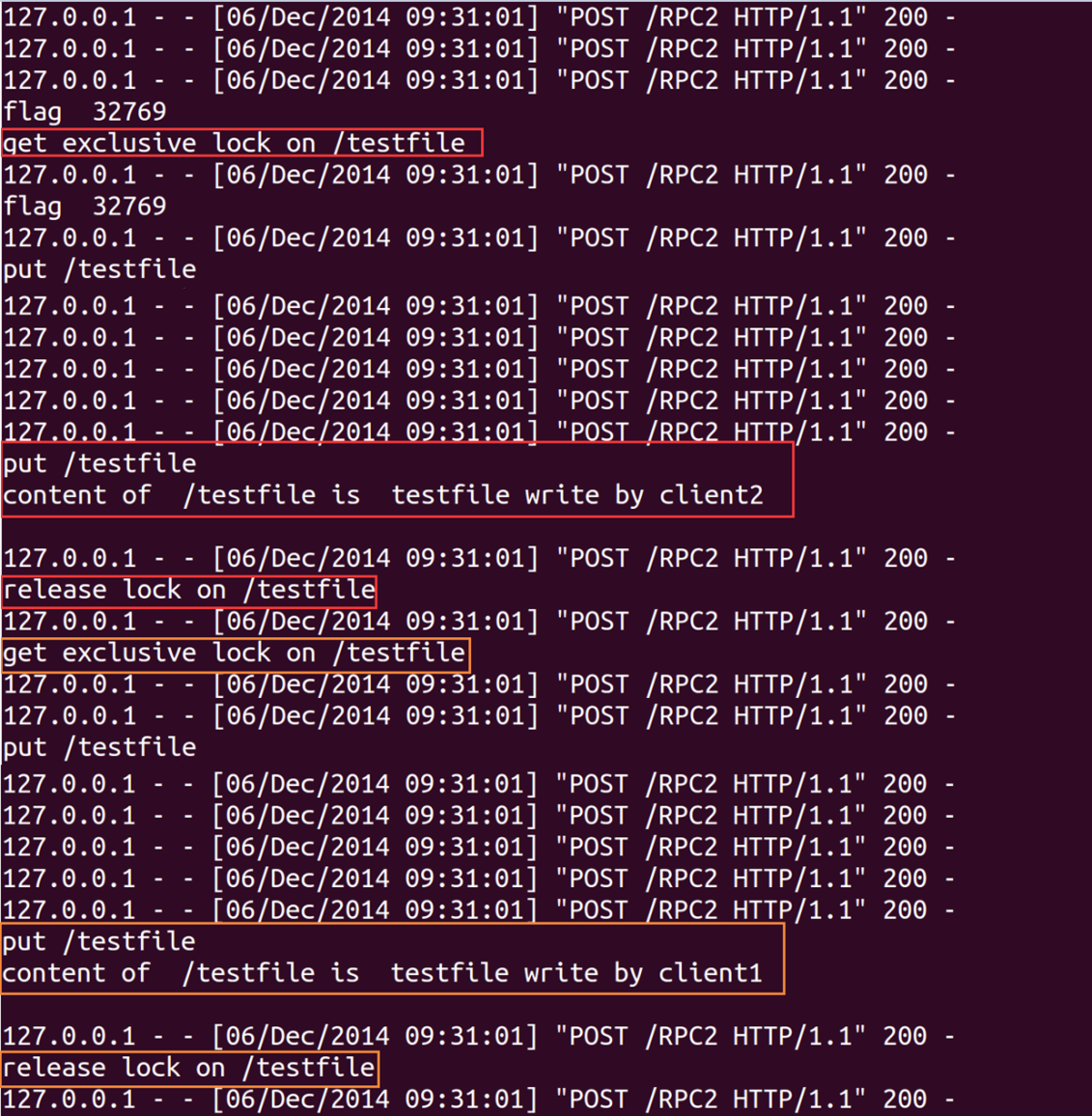
The output in the server's terminal should contain the threads information as follows:



As shown in the screenshot above, One thread is created for each request.

2. Observe the lock behavior

The lock release behavior should also be observed in the server's output as below:



The above screenshot shows that two write operations on the same file cannot get the exlusive lock at the same time

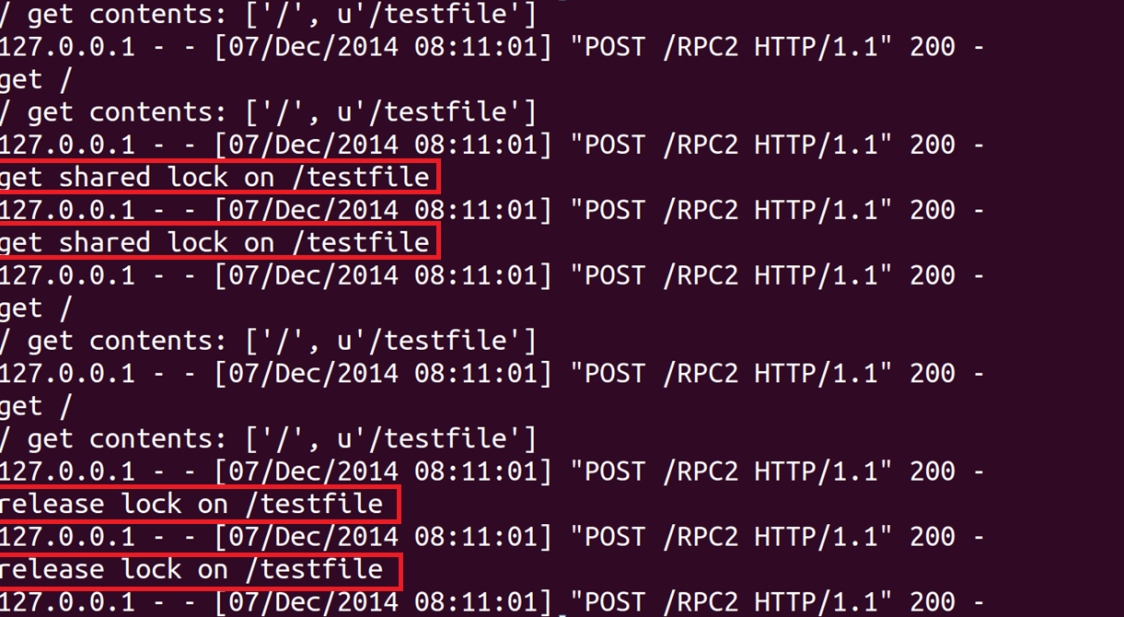
When we schedule two concurrent "cat" command in crontab:

m h dom mon dow command

*20 08 \* \* \* cat /home/ud/pocsd/fusepy/t2/testfile*

*20 08 \* \* \* cat /home/ud/pocsd/fusepy/t1/testfile*

The output will be:



Above screenshot shows that two read operations on the same file can get the shared lock at the same time

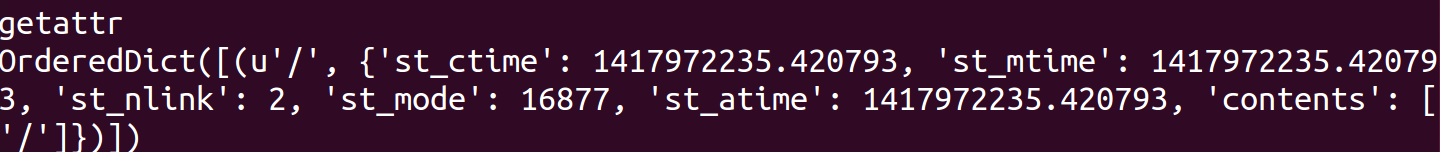
**3. Cache coherence**

To test the cache coherence of our filesystem, the following tests are conducted:

First we enter the FUSE Filesystem:

*cd t2*

At this stage, the client requests data for "/", and the data of "/" is stored in cache, which can be observed in the client's output:

****

Then we create a file called "testfile" in the system:

*touch testfile*

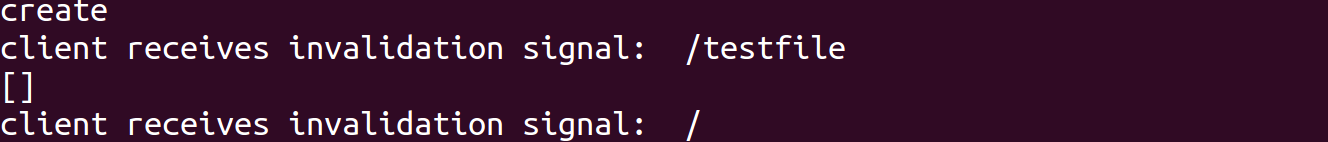
The touch command modifies the "/" data and the "testfile" data. So invalidation signals containing their pathnames are sent by the server, and the following output will be observed:

Server's output:

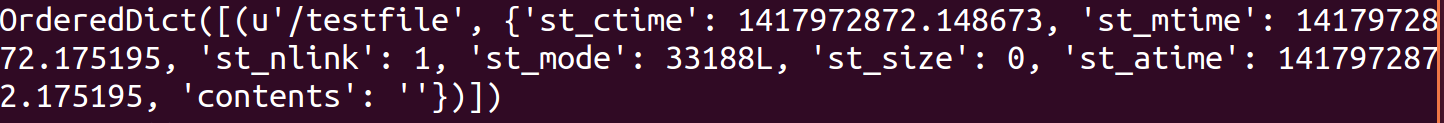




We can observe that the client receives the signals and invalidate the files:



Since "testfile" data is obtained again at the end of the request, now the cache contains:



Then we "echo" and "cat" the file:

*echo 'a' > testfile*

*cat testfile*

Observe the output of the cat operation. The cache now contains: 