Part2 part3 demonstration:

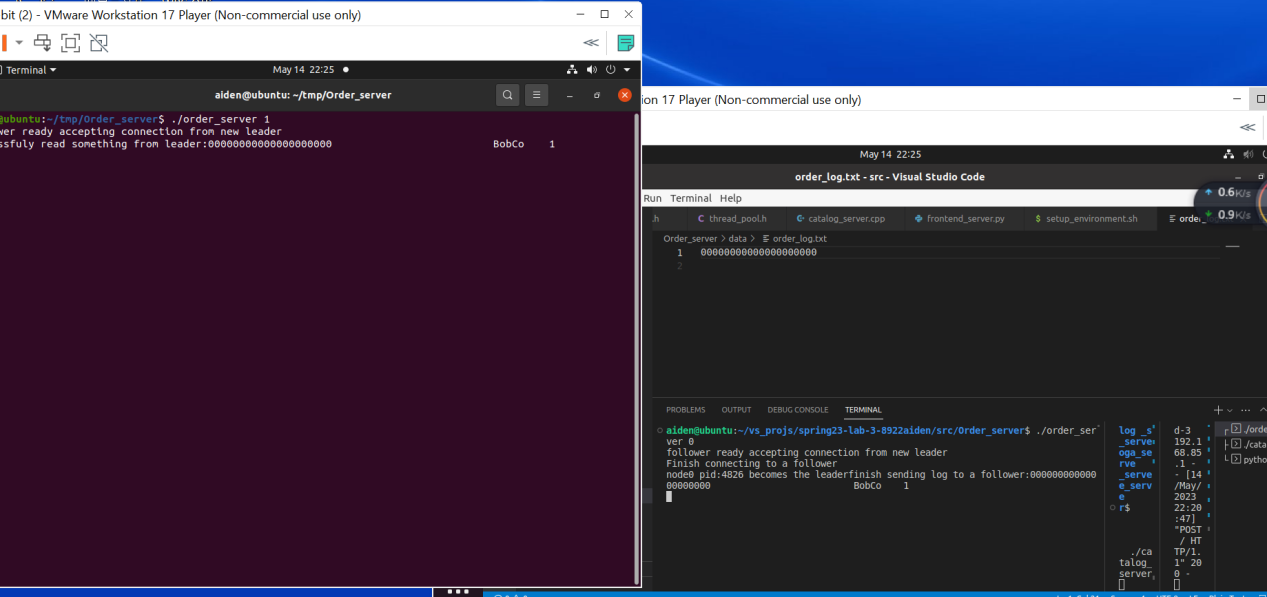
We will have 2 order server replicas running on two different machine:

ORDER\_SERVER\_0:192.168.85.128

ORDER\_SERVER\_1:192.168.85.129

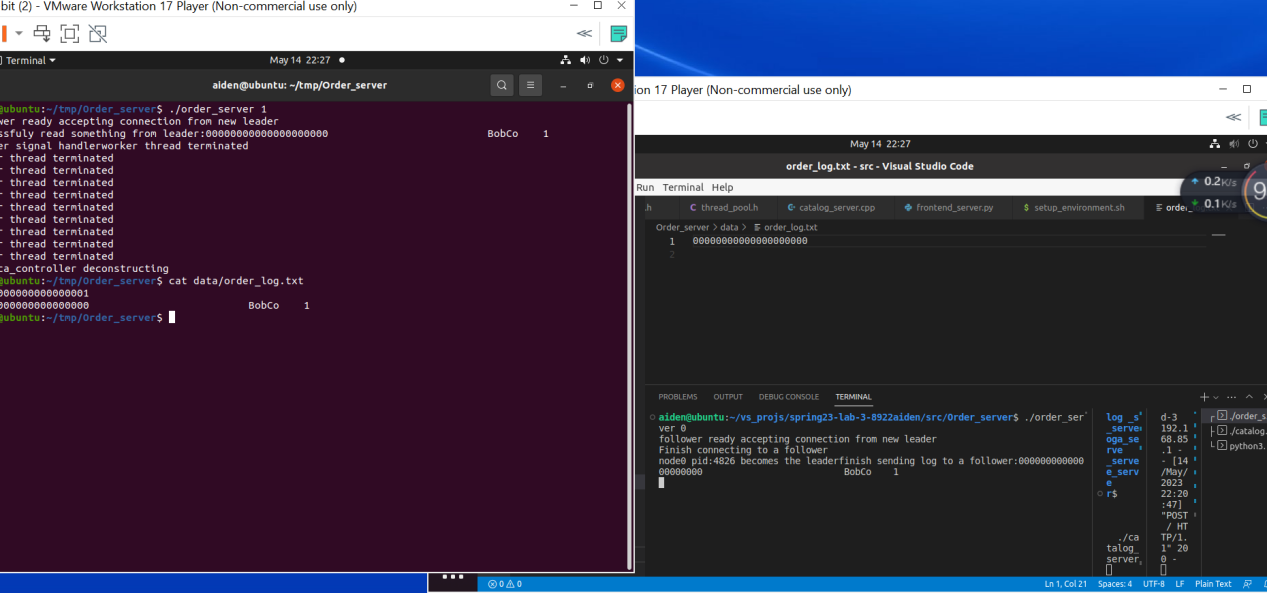
We reset the log file of both order server.

Then we start the two order\_server and issue a request.



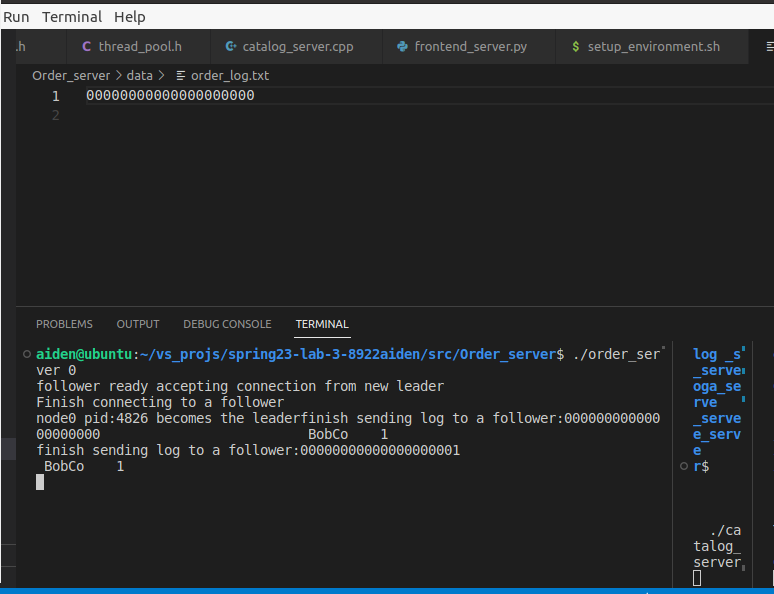
As we can see, order server 0(left)are picked as leader,it receives request from the frontend server and send it to the order\_server 1(right).

Next we shut down the order server 1, which is the follower

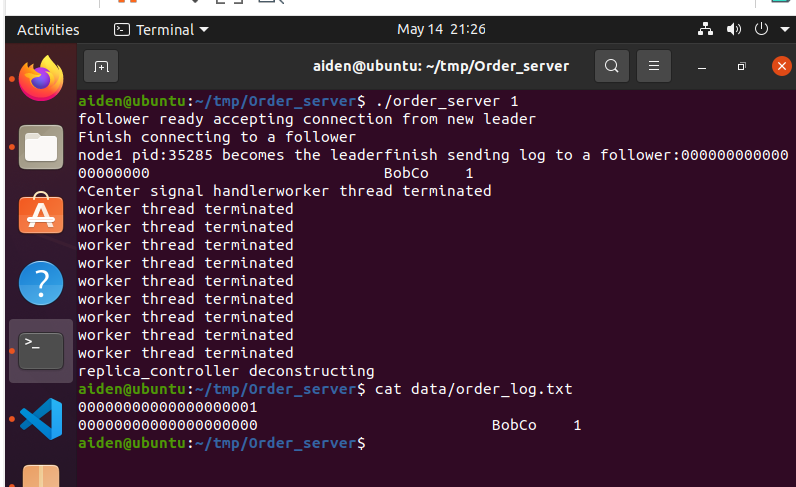


As we can see, the follower’s log file are properly synced. On the other hand, the write to the log file of leader are still cached in the c++ fstream’s buffer, so we won’t be able to see the update inmediately.

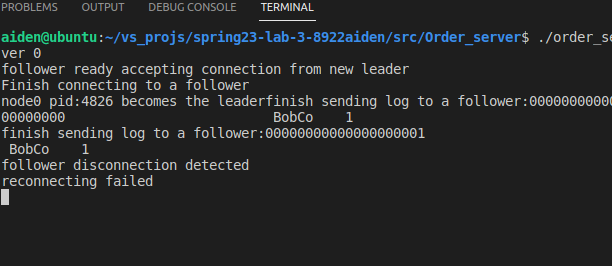
Then we continue to send request to the leader to cause a inconsistency.



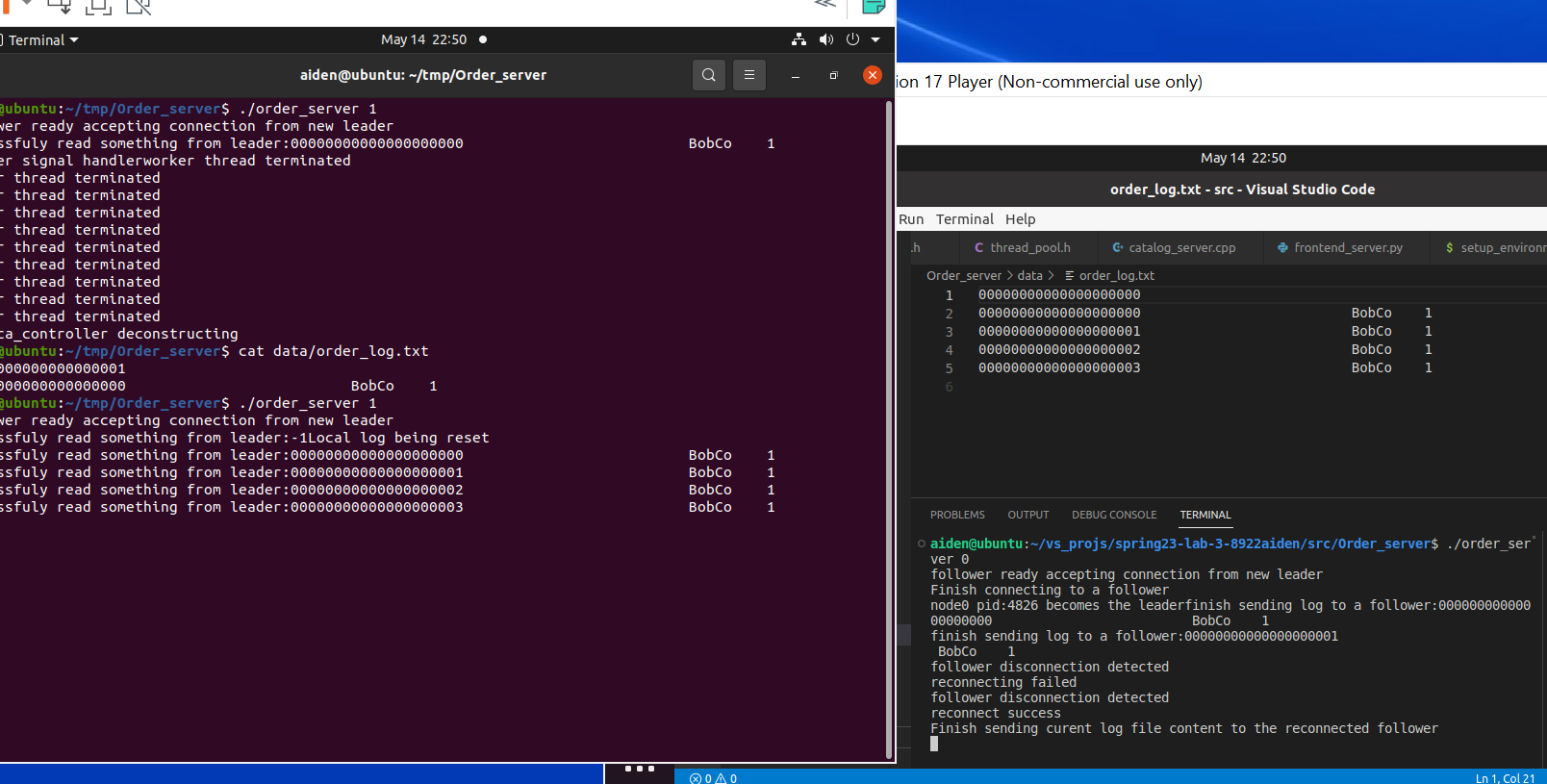
We see although the follower diconnected, the leader’s socket2follower would inmediately perceive the failure of the other side. Even if the leader socket receives fin, it can still send to the other end, but just won’t get any ack, So in this case TCP would only realize the failure of other end either after a while or after a never-got-acked send.



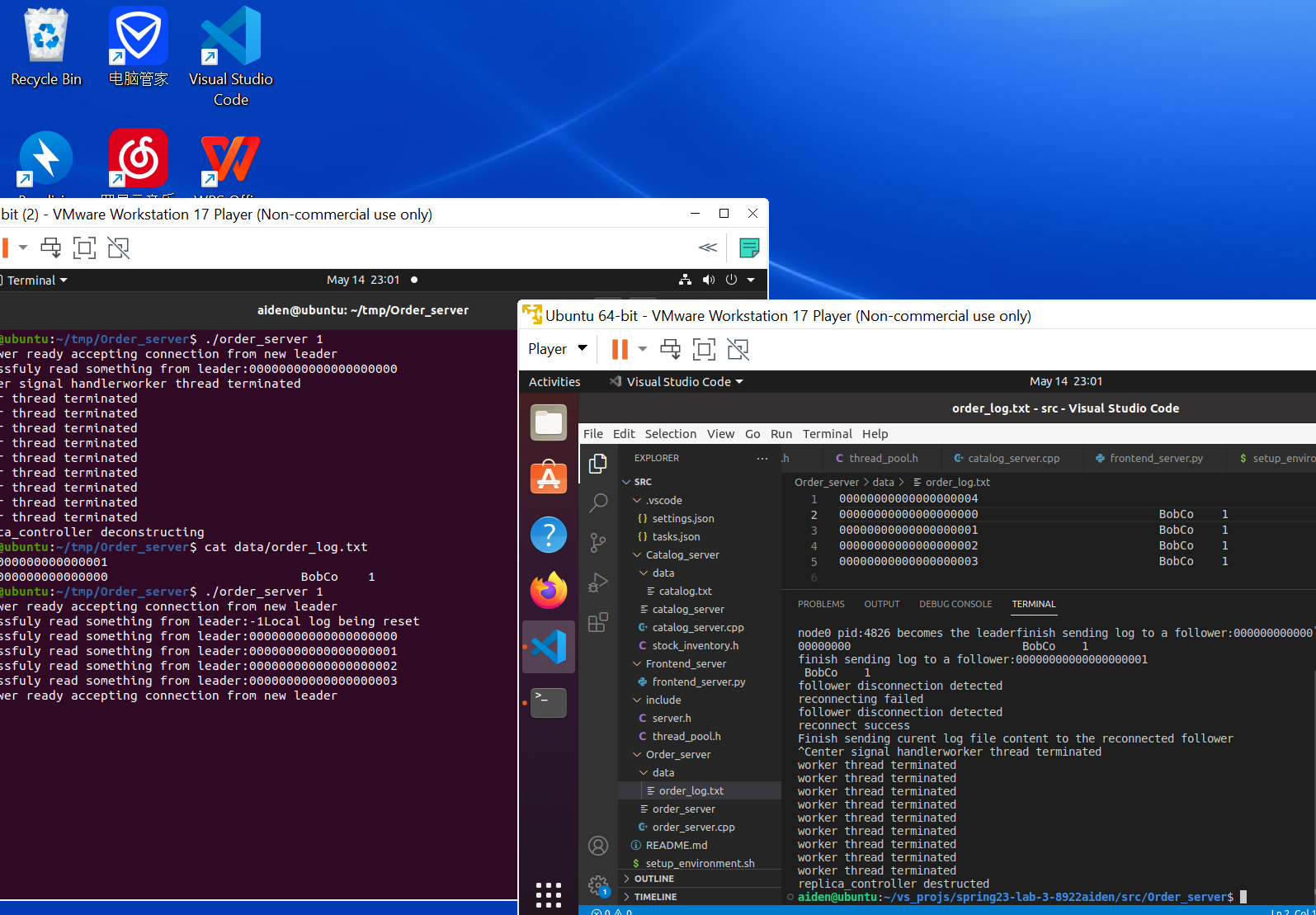
Then we send another request



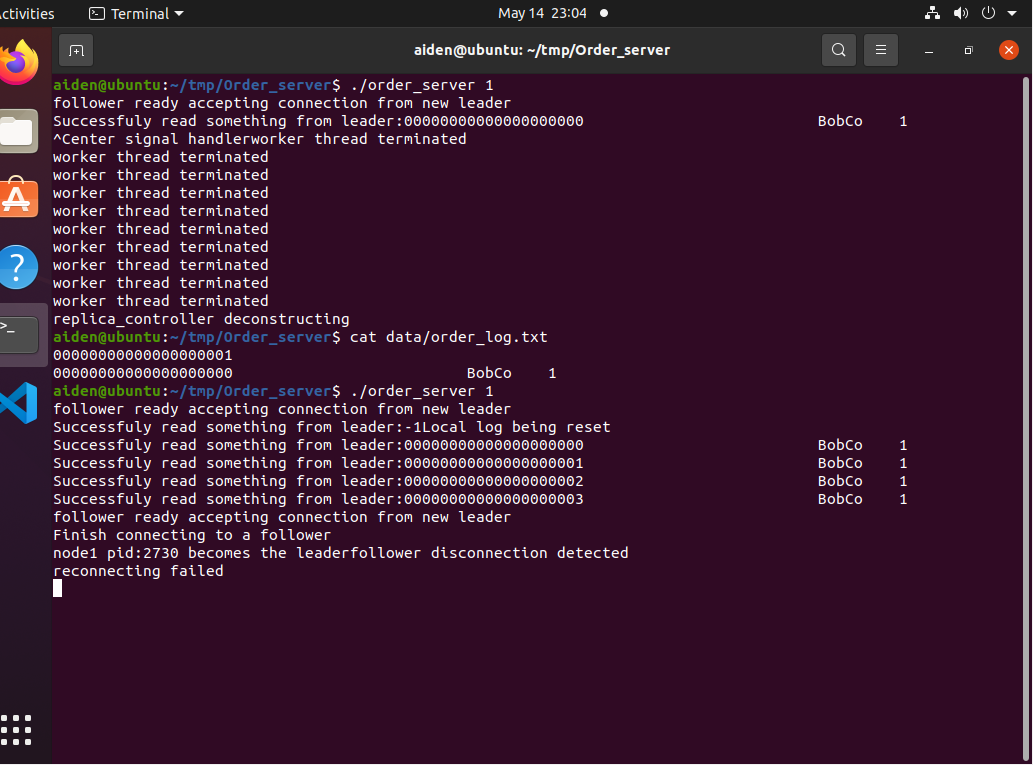
This time the leader detects the disconnection of other end when trying to send updates to follower, it then trys to reconnect. But since we haven’t reboot the server 1 yet, the reconnect will fail with no doubt.

Then we bring up the server 1 again and send another request

This time we leader detects the failure of follower , it reconnects and succeed. then it will tell follower to reset its log file and send its entire copy of the current log file to that follower.

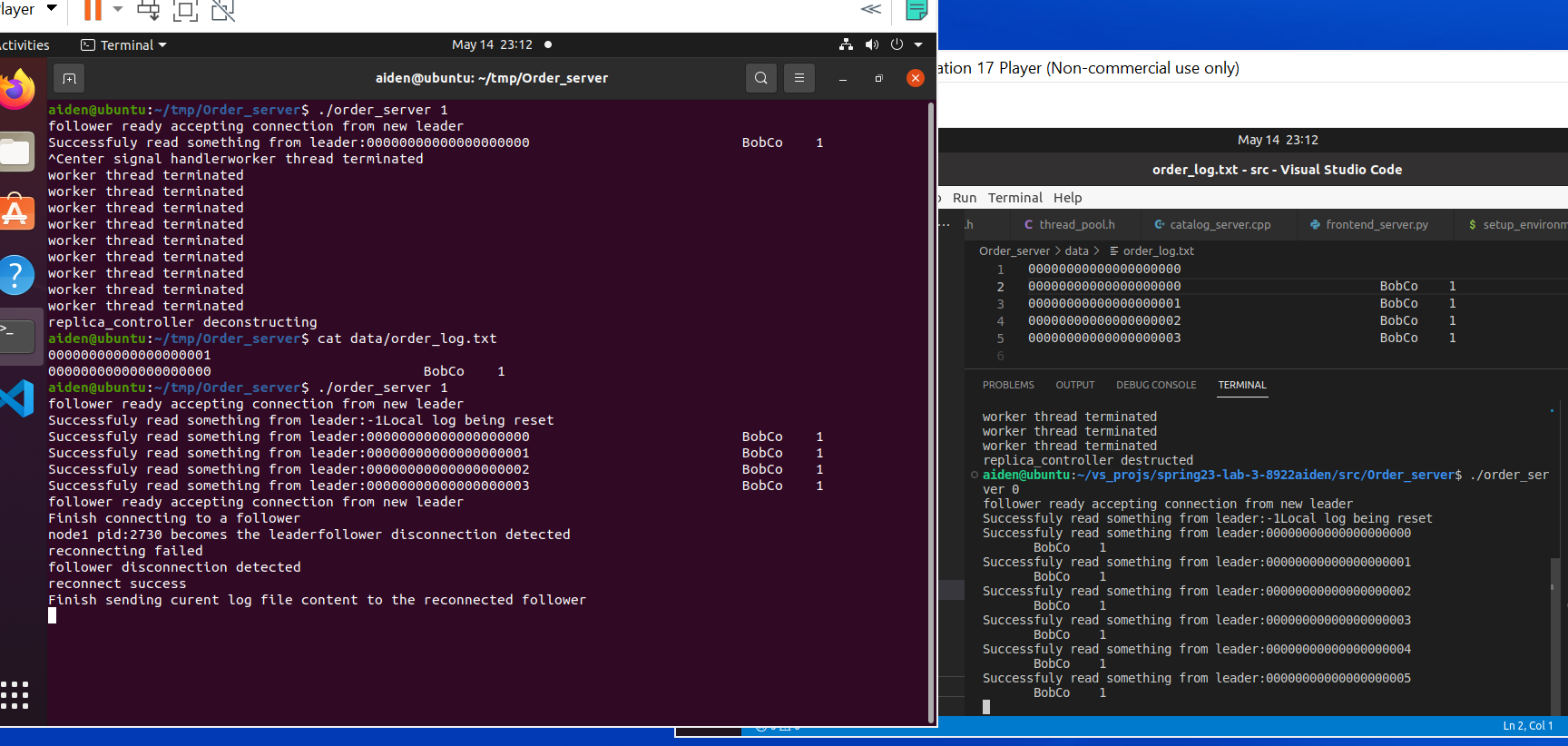
Then we terminate the leader, which is order server 0

As the leader terminates, its buffered file write will finally be conducted and we can check the log correctness. Then we send a request to see if it would be affect or not

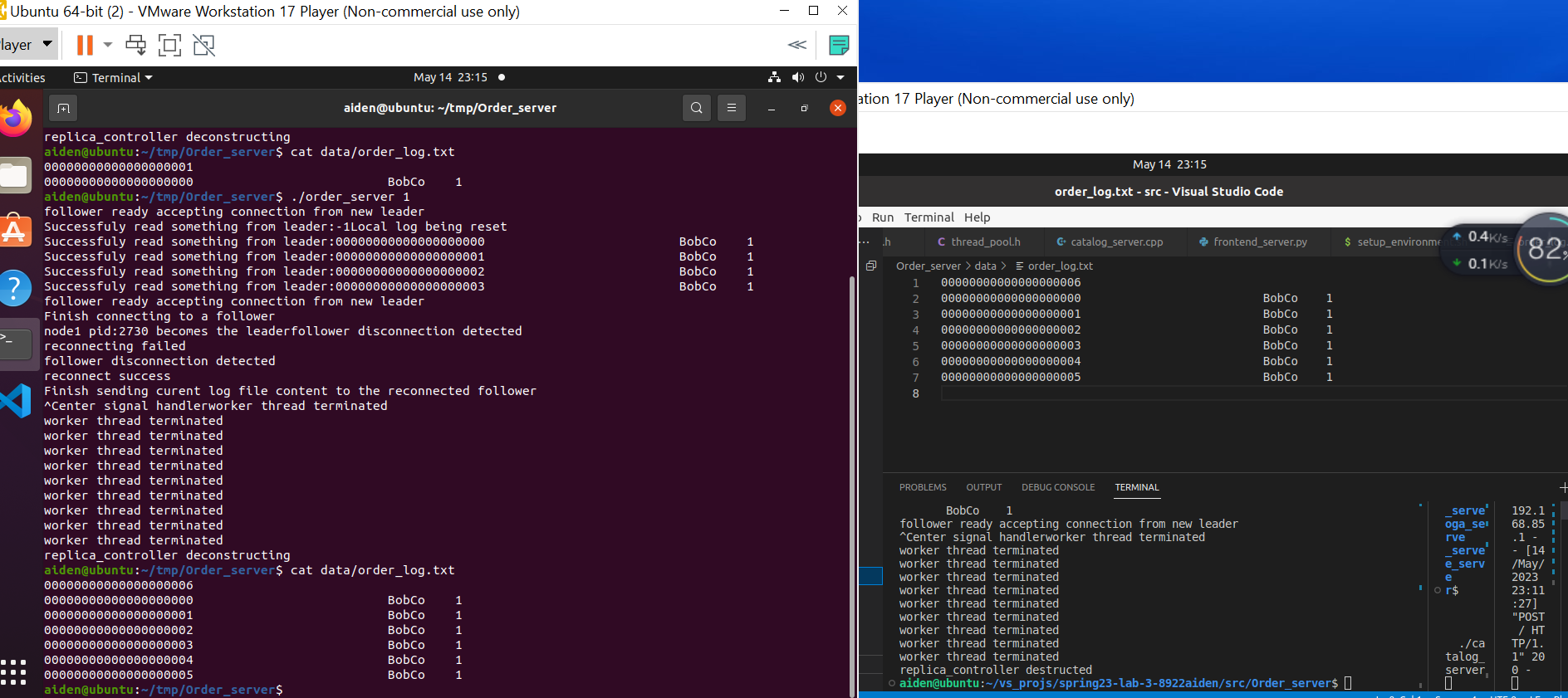


As we can see, the frontend server notice the failure of the current leader, then it select the next replica as the leader, which is order server 1. As server 1 process the request, he trys to send synchronization to order\_server 0, but it detects that order server 1 is down. So he trys to reconnect, but since server 1 hasn’t been rebooted yet. The reconnection fails.

The we bring up the server 0 again. And then send a request.



The leader won’t change simple because server 0 comes back. The frontend server will keep send requests to the current leader as long as it are’t die. And as server 1 receive and process the requests, it detects the disconnection of the follower and try to reconnect, and succeed. So it send its entire log file to server 0.

Then we terminate the two order server and check the log 

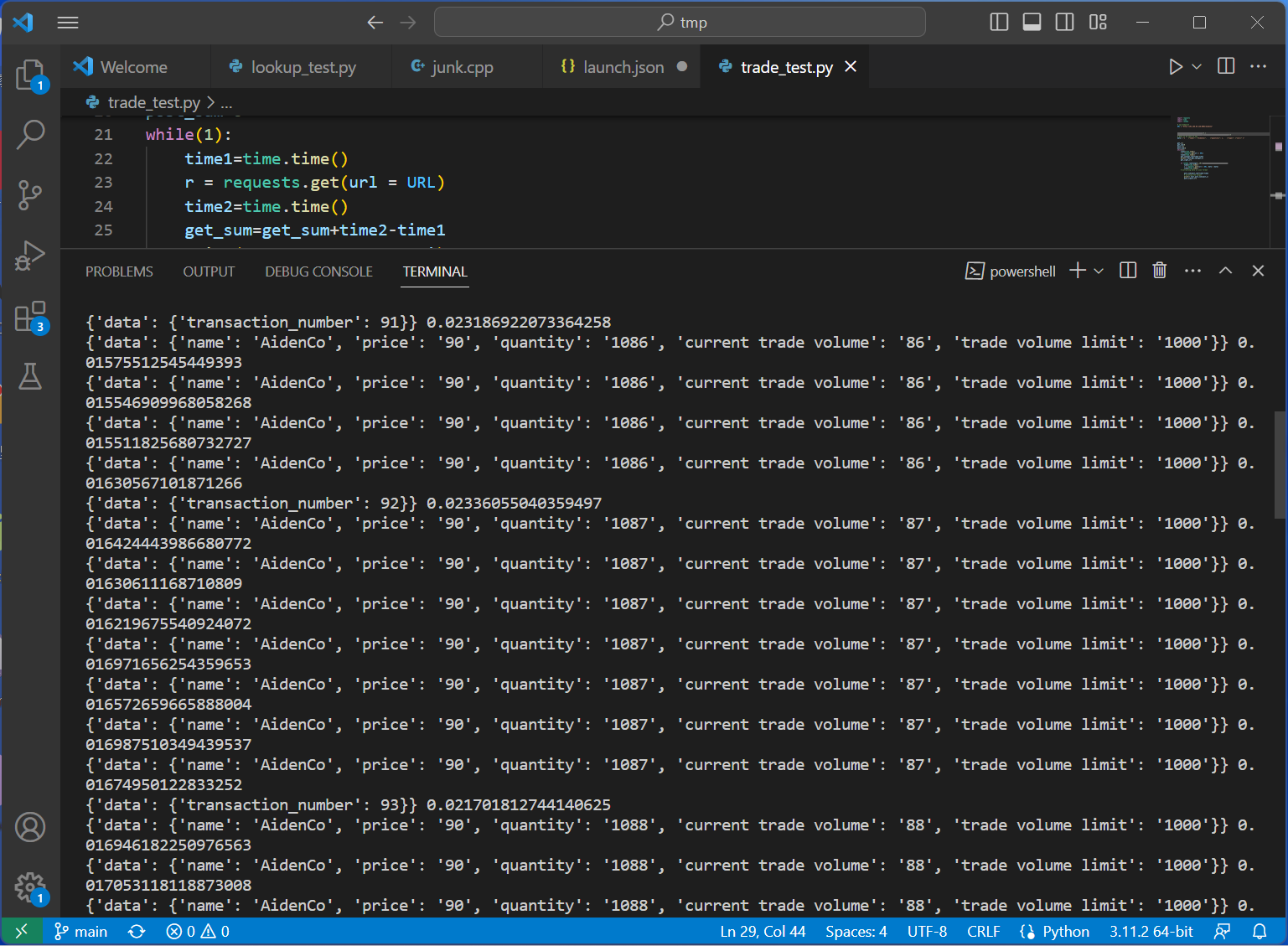
Everything is as expected.

Another thing is as long as at least one order server exists, the request will get dispatched to that server and the user won’t feel any different

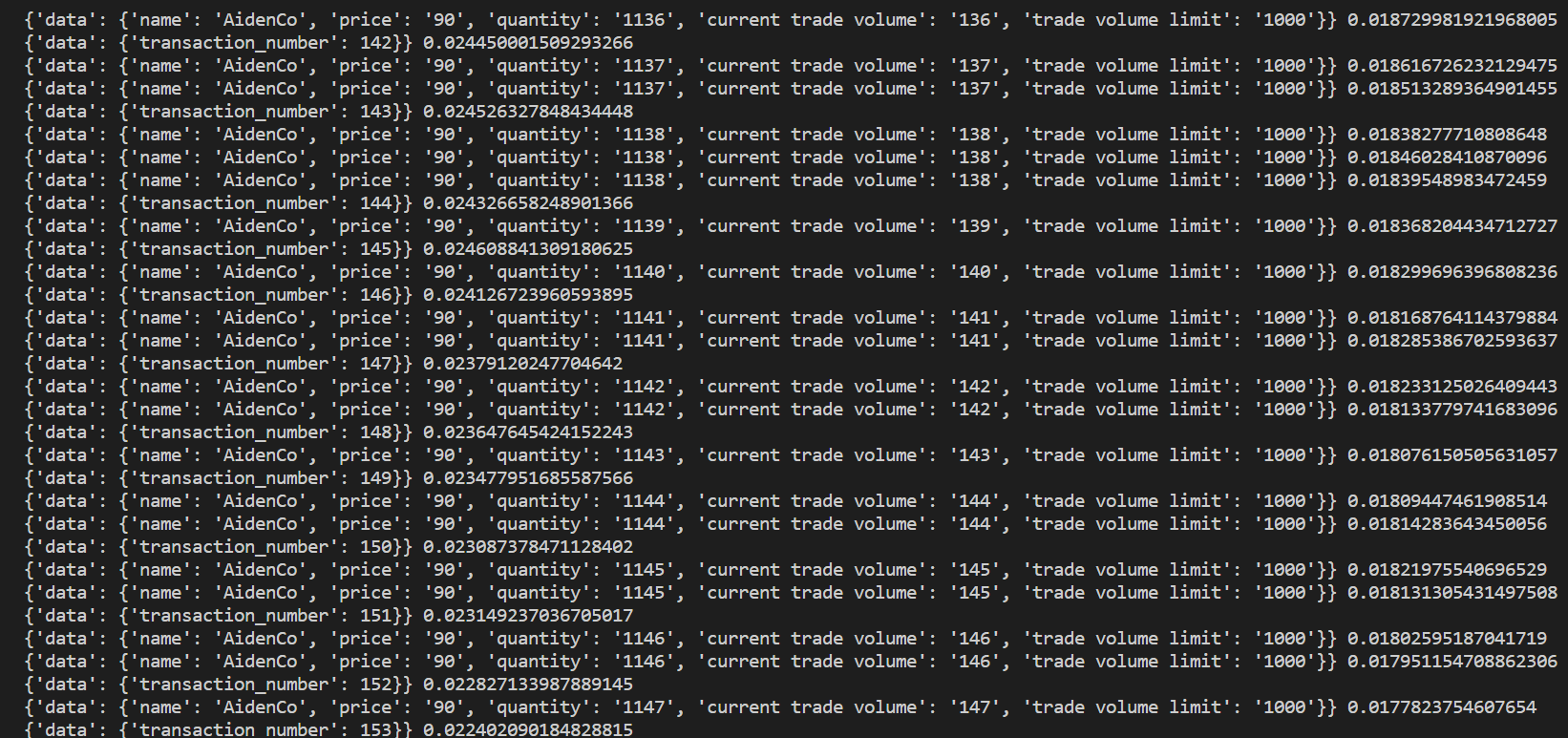
Part1 demonstration:

here we use a client to repeatively issue lookup requests and trade request to the systems, to test the average response time for different requests. Also we chance the possibility of a trade request following a lookup to see the current cache implementation(send invaildation)’s impact to the performance. We reset the cache before every test.

P=0.2

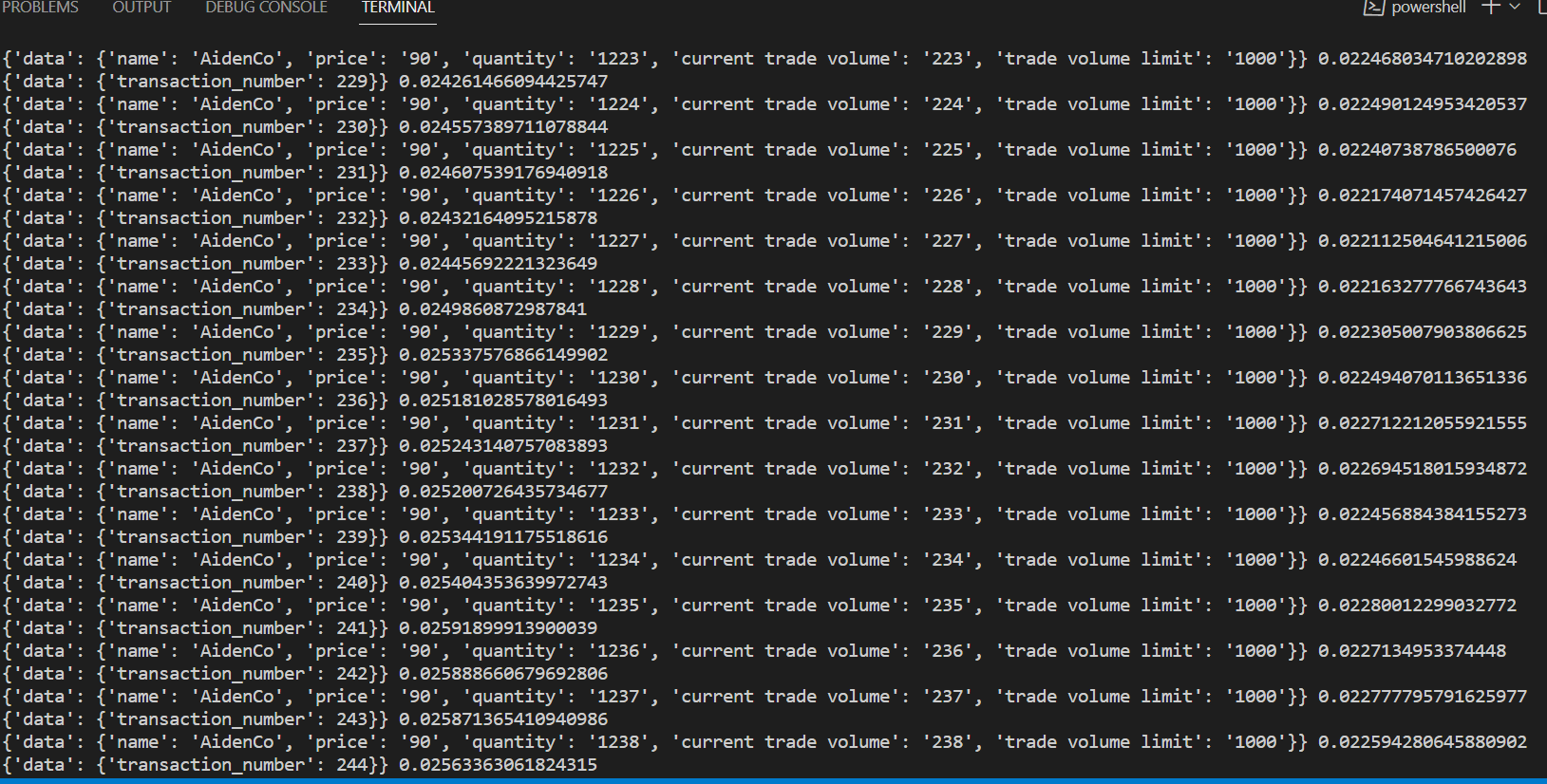


Average lookup:0.016s average trade: 0.021s

P=0.6

Average lookup=0.018s average trade:0.022s

P=1

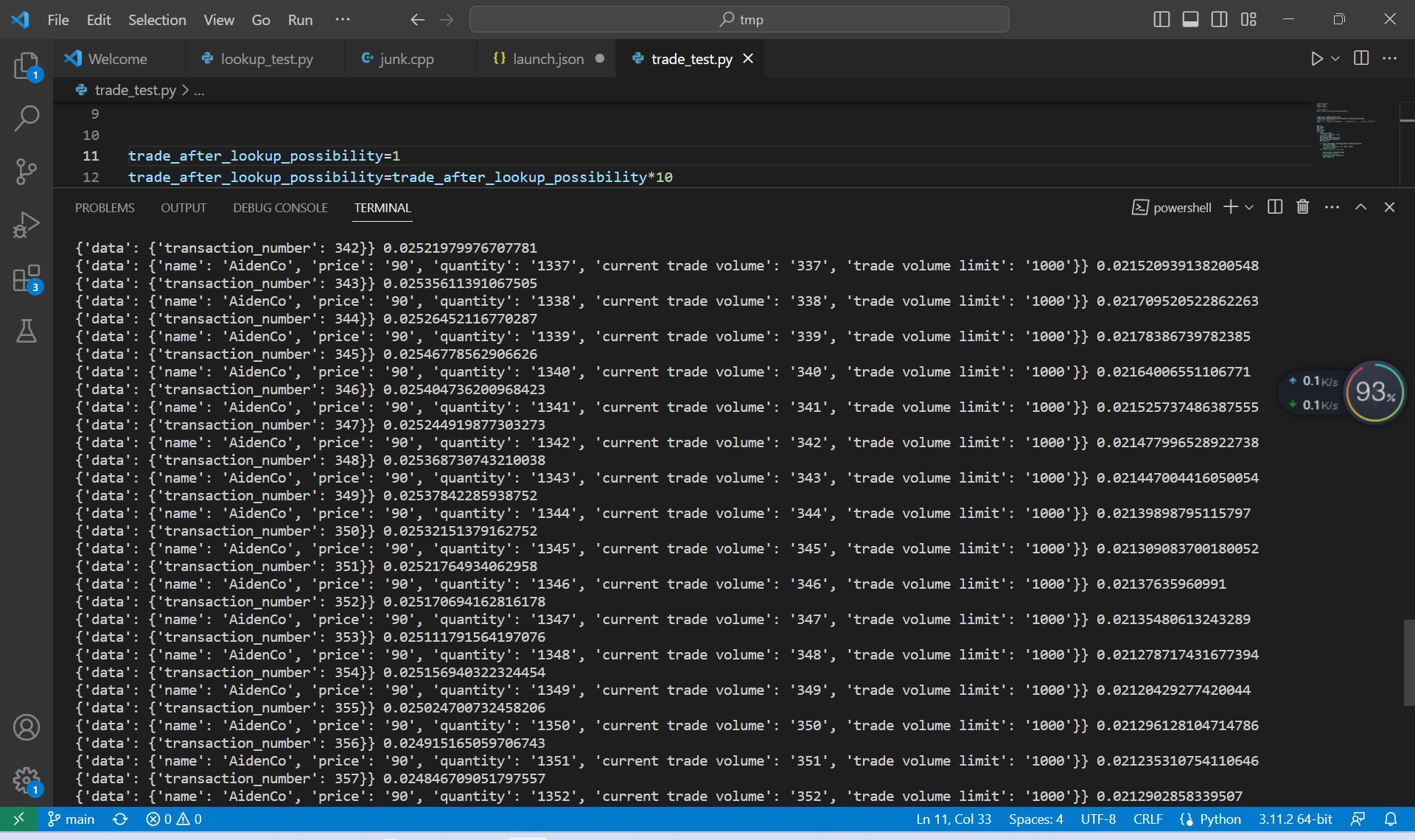


Average lookup:0.022s average trade:0.025s

We can see, as we p goes up the average trade time goes slightly up, but lookup time increase a lot. It is within expection because in the worst case there is a subsequent trade that invaildates what ever is brought to the cache by the former lookup. Which means we wont get benefit from cache at all when look up, despite the overhead of maintaining the cache. The increase respond time for trade can be interpreted as a reason of the increased load(more intense contention in the critical area).

To reinforce our interpretation, we can disable the cache and compare the result

P=1 with cache disabled



We can see there is almost no different, even some performance increase compared to the P=1 enable cache setting

On the other hand, when we let P=0.2 and disable cache , we would find significant performance drawback in lookup

