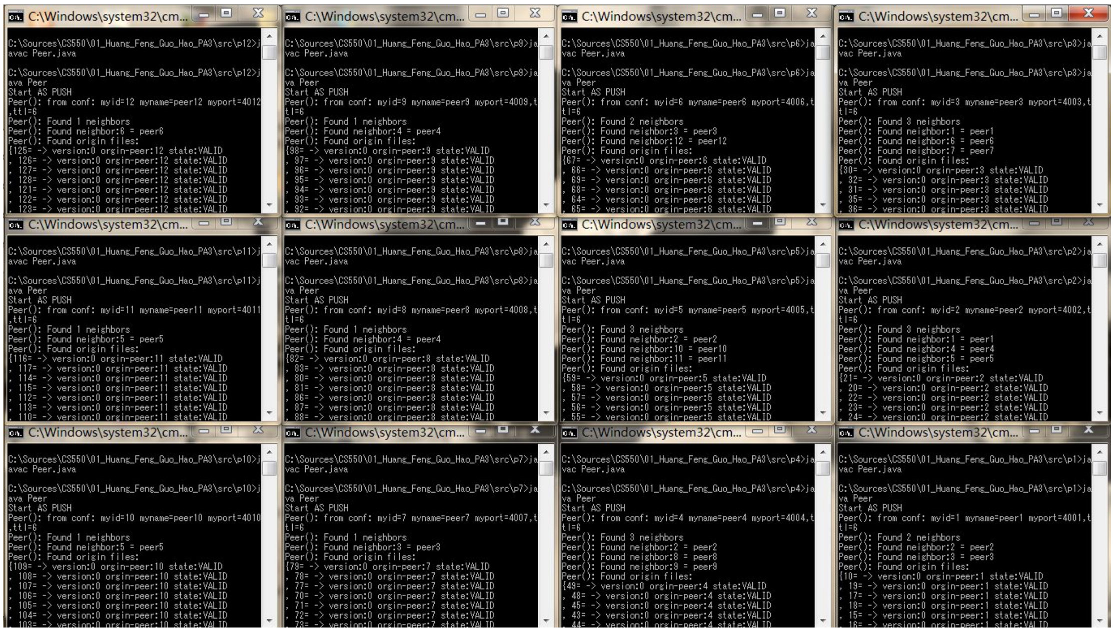
Verification Test

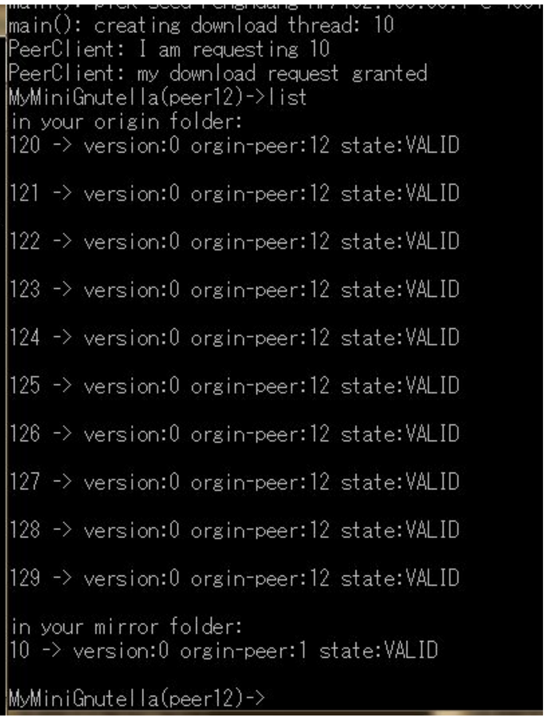
In this file, we will demonstrate why our program works correctly, in other words, how does a file get invalidated. Details about topology and file transferring are talked in previous projects, in this one, we focus on file consistency. How the testing files are located is stated in performance.section1.

PULL:

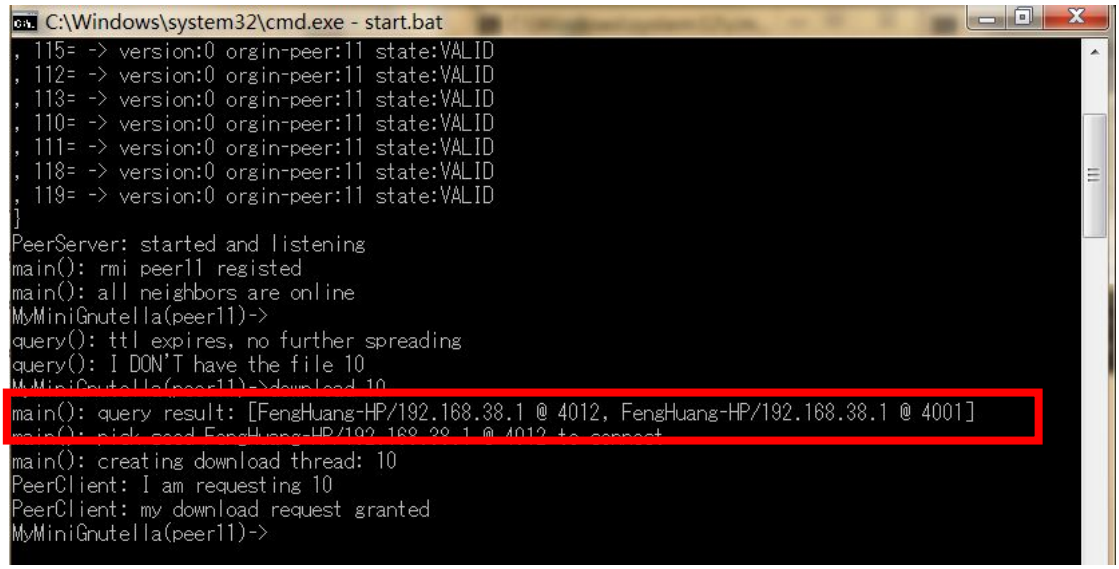
After startup:



Zoom in to peer 12: after issued : download 10

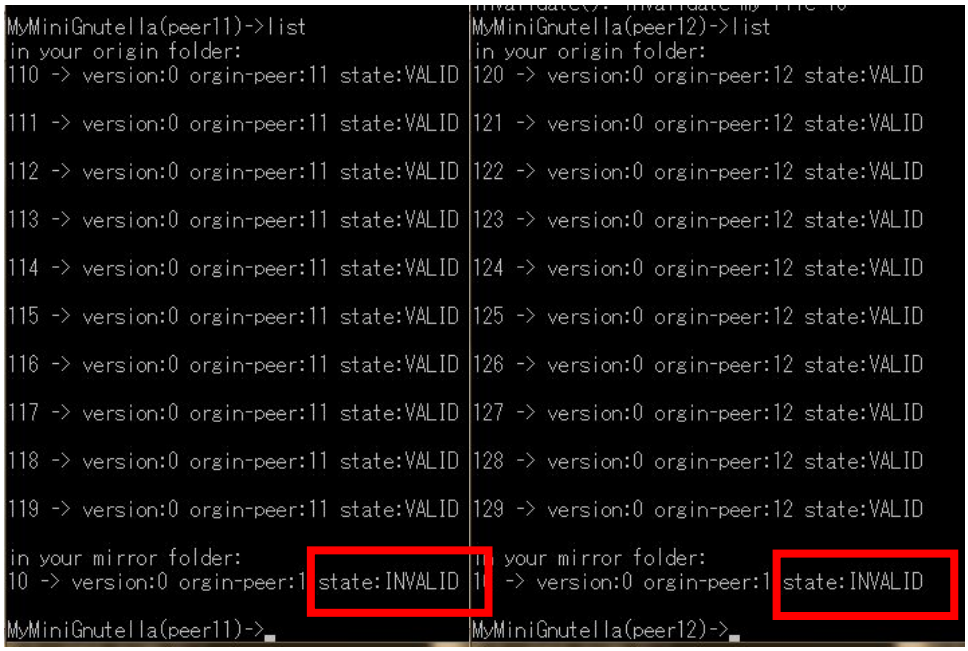


At this time we can see peer12 has file 10, in mirror folder, with version and owner. Now we try to download this 10 from peer 11 :



From the red box we see peer11 got to know both peer 1 and peer 12 has the copy. Now we type "modify 10" in peer 1, we will see a lot of invalidation message transferring, and finally in peer 1:

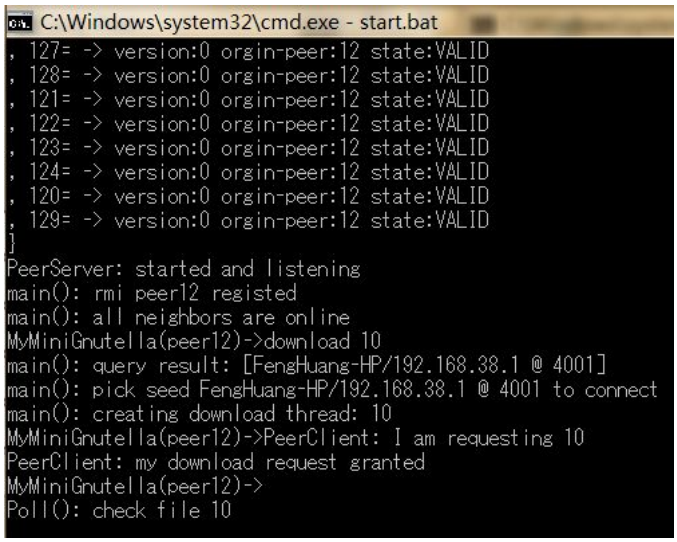
We see that 10's version became 1, not 0. And since invalidation message spread to peer 12 and 11, they are expected to invalid the copy:



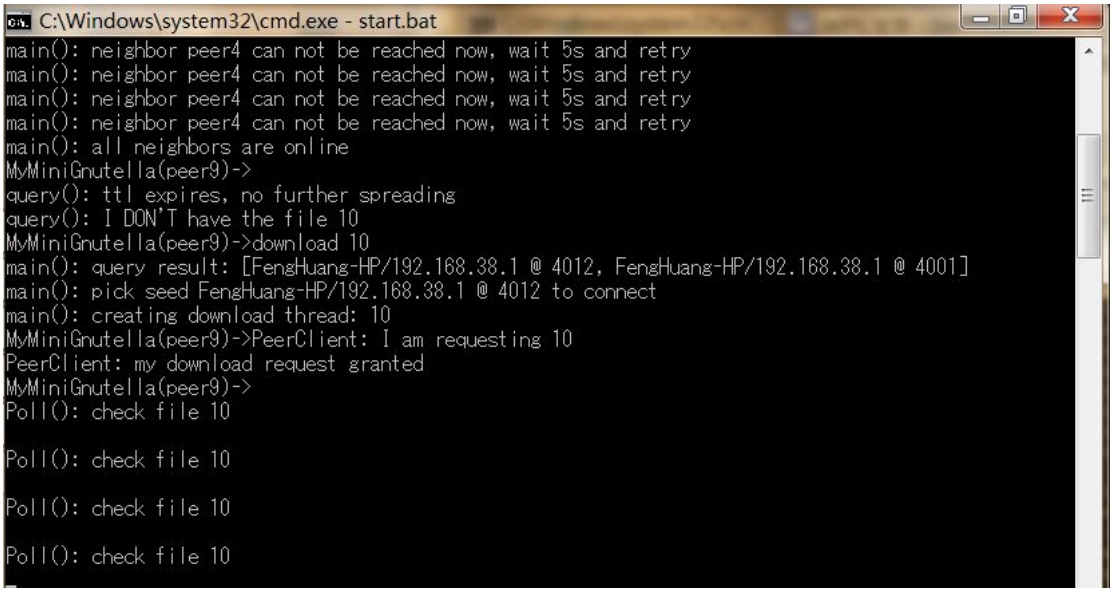
At last, if we issue a “download 10” in peer 10, it should know peer 11 and peer 12’s invalidated copies:

PUSH:

In the push mode, each file has a thread to poll the owner about version of specific file, see the following:

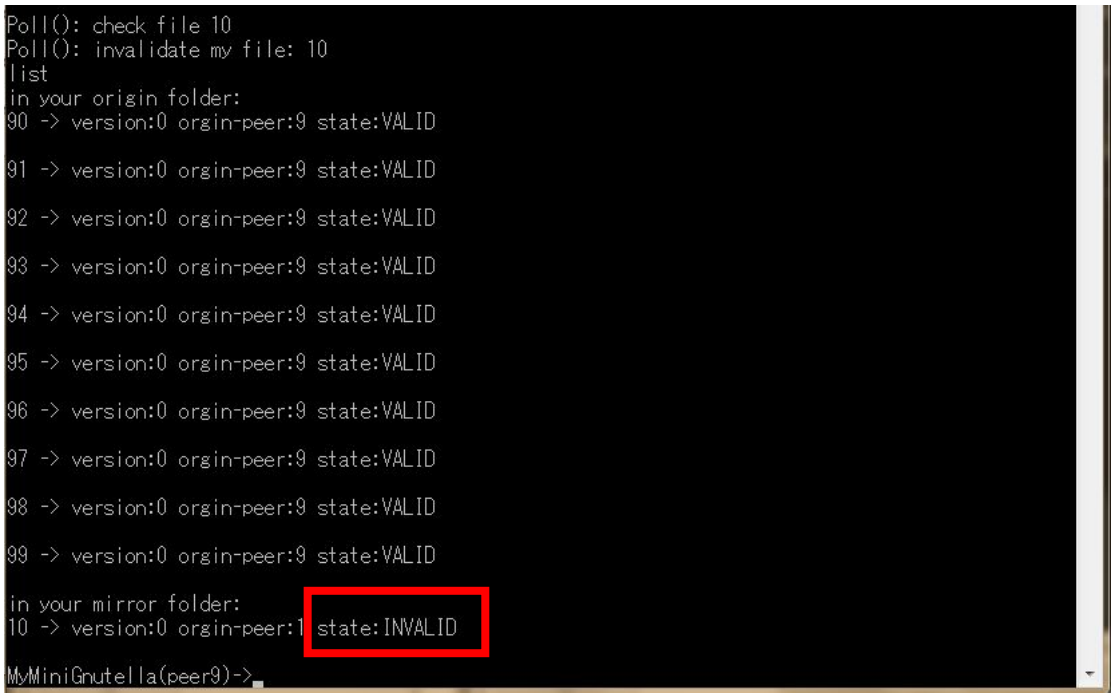


After peer 12 download 10, peer 9 also does it:



After peer 1 modified its file 10, it version changed:

And after the polling interval, peer 9 polled and found its 10 is out-dated:


Poll(): check file 10
Poll(): invalidate my file: 10
list
in your origin folder:
90 -> version:0 orgin-peer:9 state:VALID
91 -> version:0 orgin-peer:9 state:VALID
92 -> version:0 orgin-peer:9 state:VALID
93 -> version:0 orgin-peer:9 state:VALID
94 -> version:0 orgin-peer:9 state:VALID
95 -> version:0 orgin-peer:9 state:VALID
96 -> version:0 orgin-peer:9 state:VALID
97 -> version:0 orgin-peer:9 state:VALID
98 -> version:0 orgin-peer:9 state:VALID
99 -> version:0 orgin-peer:9 state:VALID
in your mirror folder:
10 -> version:0 orgin-peer:1 state:INVALID
MyMiniGnutella(peer9)->