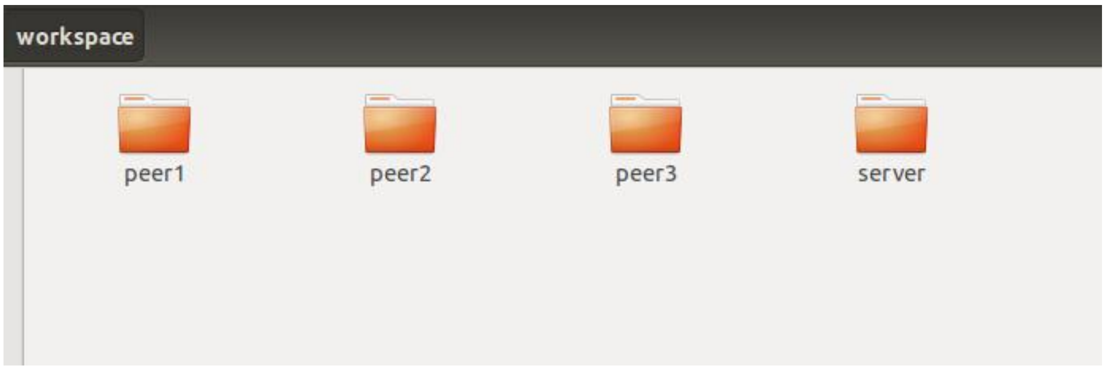
Verification

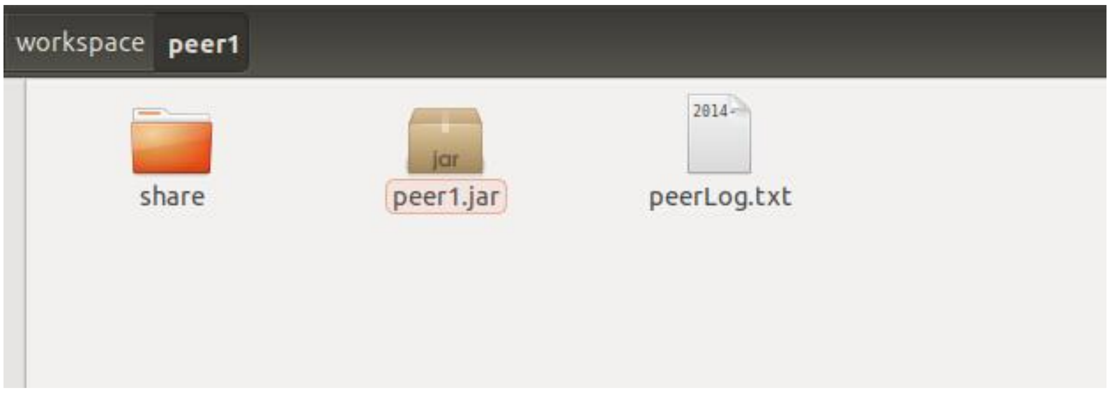
This file shows the verification of the functions. All the tests are conducted under Linux system.

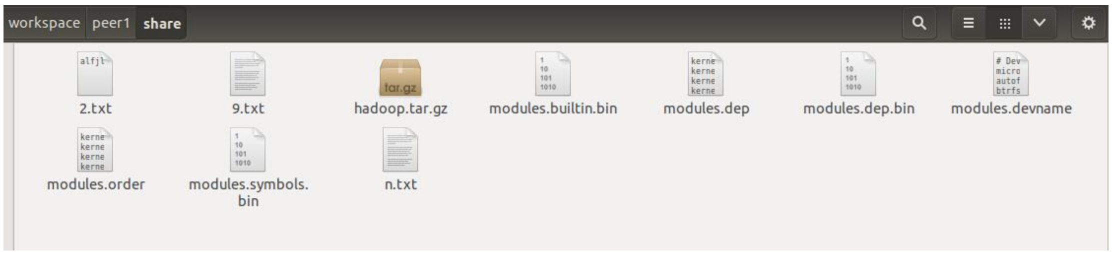
1. Test Environment and Nodes Deployment

Under this test, one index server and 3 peers were used for testing.



Under the peer folder, there is a share folder, where stores files registered on the index server.

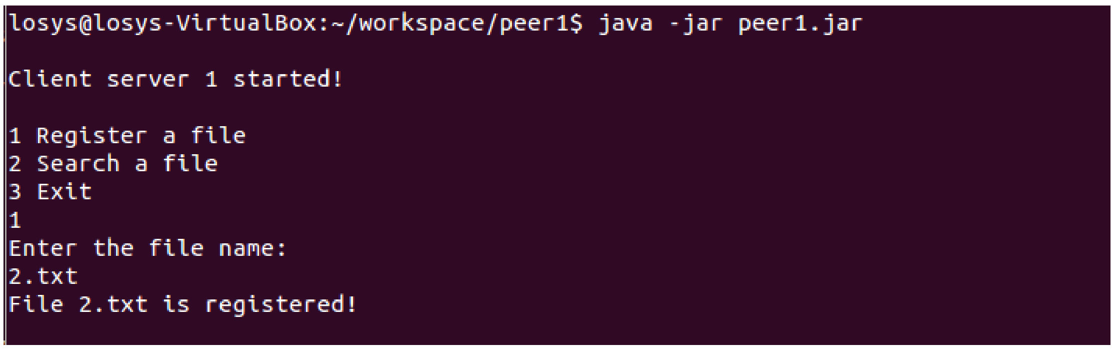




And there is a peerLog.txt that records the operations on the local peer.

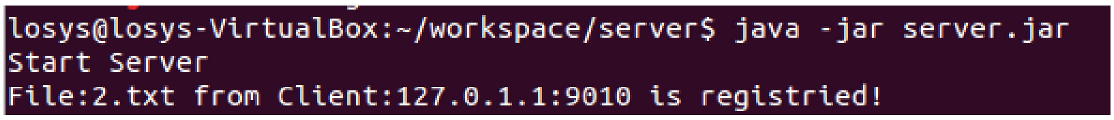
2. Registry.

When register a file, enter the file's name, the file information will be recorded in server's data structure. The below pictures show that the registery function is correct.



```
losys@losys-VirtualBox:~/workspace/peer1$ java -jar peer1.jar

Client server 1 started!

1 Register a file
2 Search a file
3 Exit
1
Enter the file name:
2.txt
File 2.txt is registered!
```

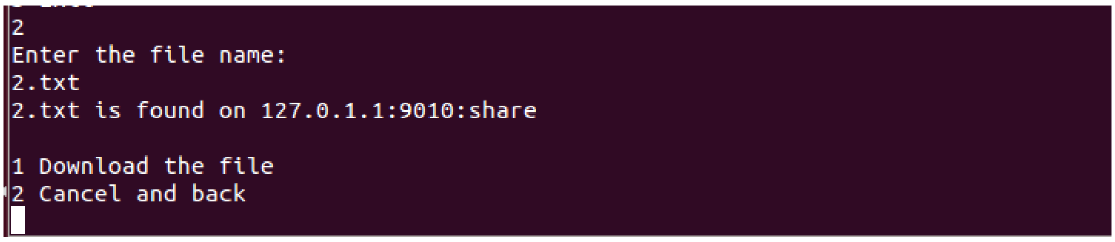
The server screen displays:



```
losys@losys-VirtualBox:~/workspace/server$ java -jar server.jar
Start Server
File:2.txt from Client:127.0.1.1:9010 is registried!
```

3. Search File

The registry has been verified successful. Now we will verify the search function. Use Peer3 to search the file 2.txt:

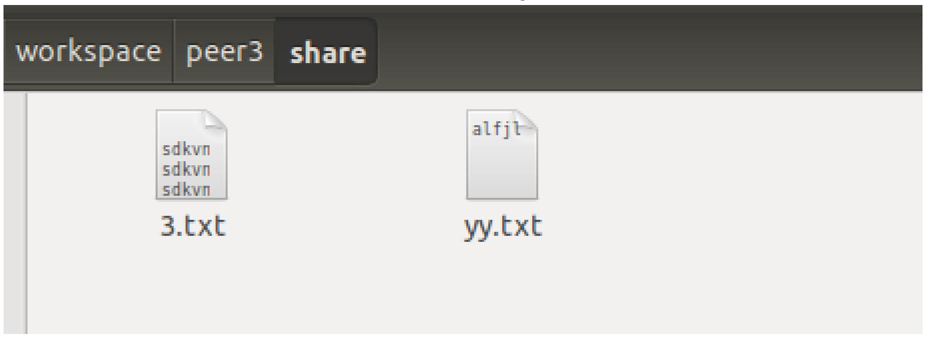


```
2
Enter the file name:
2.txt
2.txt is found on 127.0.1.1:9010:share

1 Download the file
2 Cancel and back
```

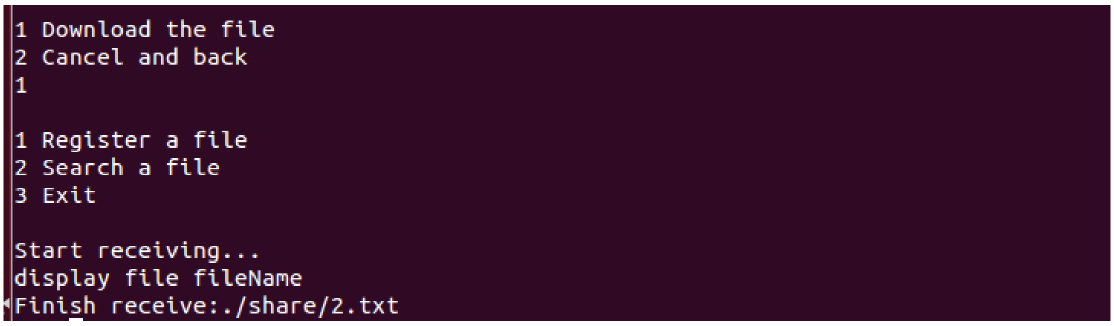
Obviously it succeeds. Then you can choose to download this file.

4. Download

Before download, the share directory contains 2 files:

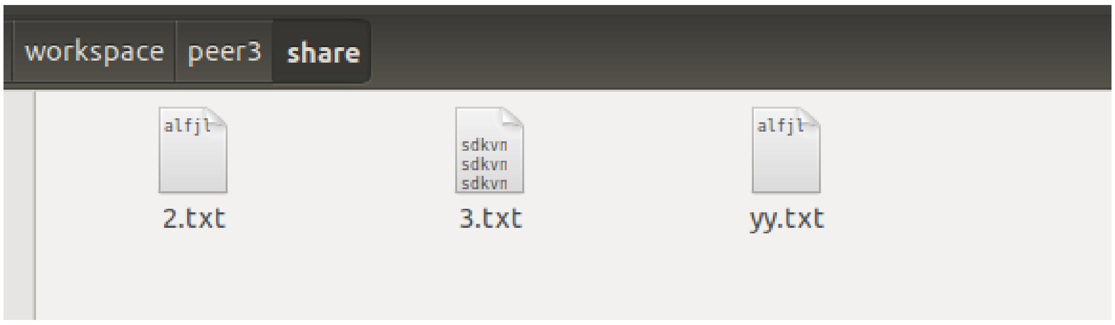


When you find a file, you can download it:



```
1 Download the file
2 Cancel and back
1

1 Register a file
2 Search a file
3 Exit

Start receiving...
display file fileName
Finish receive:./share/2.txt
```

Now the file 2.txt has been downloaded, open the share directory of peer3 to see whether the file 2.txt is in.

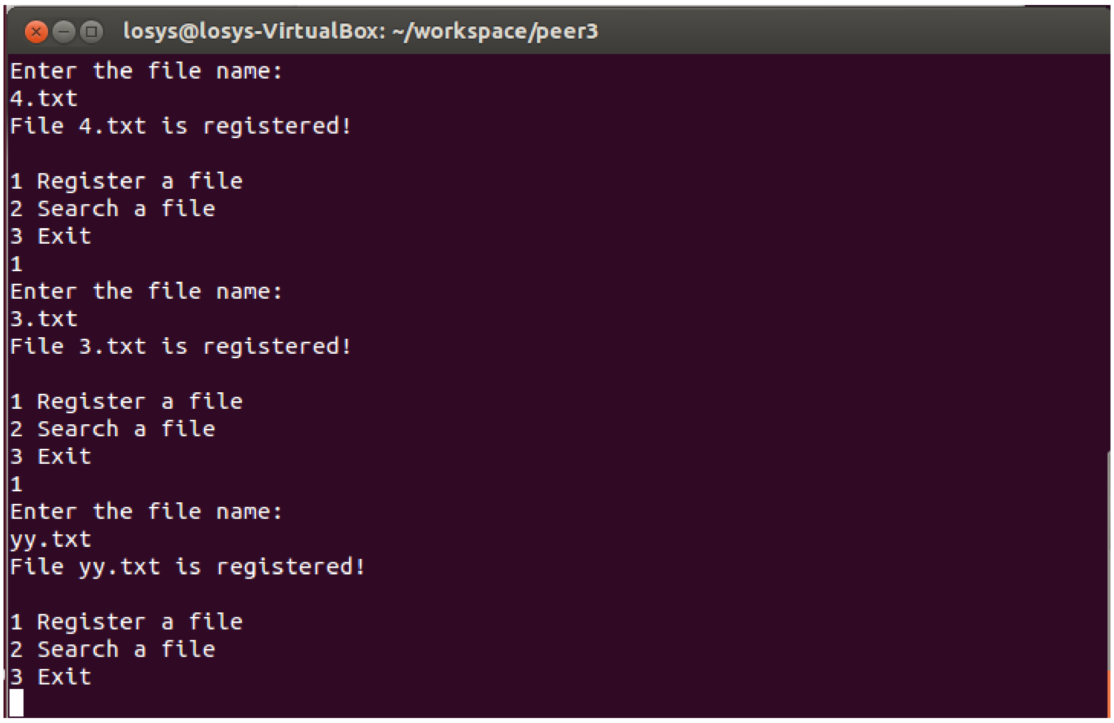


Surely 2.txt is downloaded!

5. Automatic Update

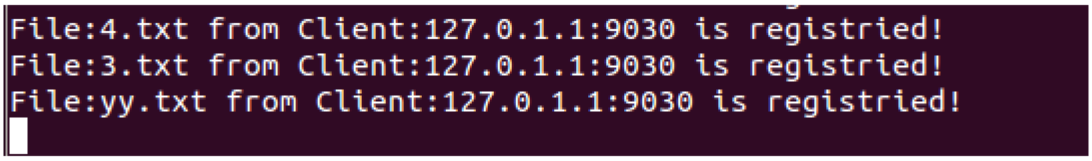
The peers have a file moniter, if the registered file is removed, they can nofity the server to delete the register information.

First, register three files (3.txt, 4.txt, yy.txt) to server



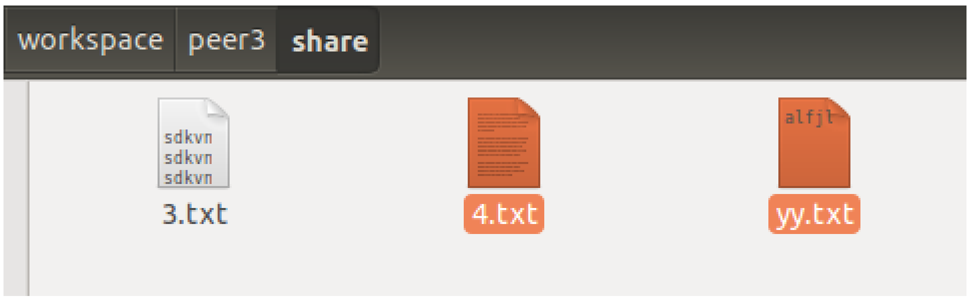
```
Enter the file name:
4.txt
File 4.txt is registered!

1 Register a file
2 Search a file
3 Exit
1
Enter the file name:
3.txt
File 3.txt is registered!

1 Register a file
2 Search a file
3 Exit
1
Enter the file name:
yy.txt
File yy.txt is registered!

1 Register a file
2 Search a file
3 Exit
```

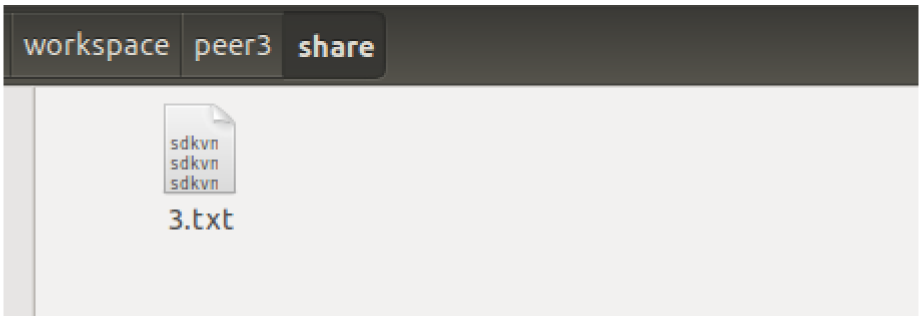
Server records the registries.



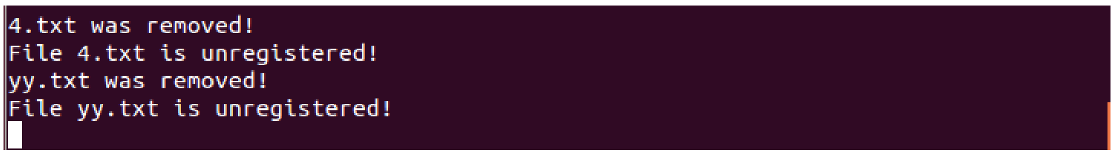
```
File:4.txt from Client:127.0.1.1:9030 is registried!
File:3.txt from Client:127.0.1.1:9030 is registried!
File:yy.txt from Client:127.0.1.1:9030 is registried!
```

Now remove 4.txt and yy.txt from the share directory of peer3.



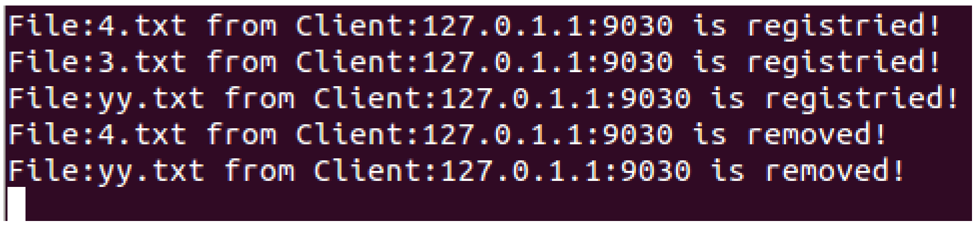


The screen of peer3 will display:



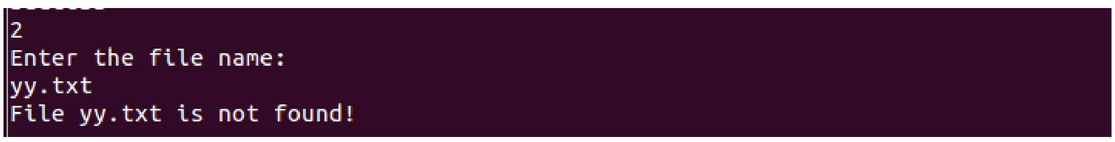
```
4.txt was removed!
File 4.txt is unregistered!
yy.txt was removed!
File yy.txt is unregistered!
```

And the server will display:



```
File:4.txt from Client:127.0.1.1:9030 is registried!
File:3.txt from Client:127.0.1.1:9030 is registried!
File:yy.txt from Client:127.0.1.1:9030 is registried!
File:4.txt from Client:127.0.1.1:9030 is removed!
File:yy.txt from Client:127.0.1.1:9030 is removed!
```

When you search the yy.txt using another peer, the file will not be found.

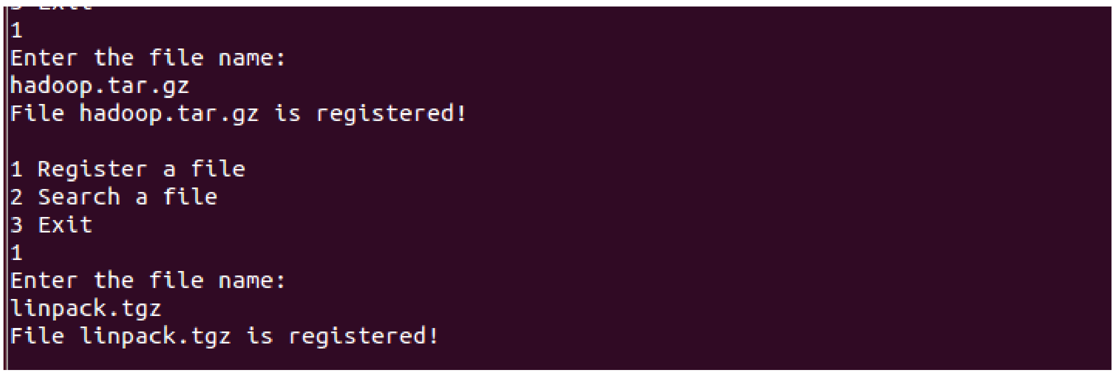


```
2
Enter the file name:
yy.txt
File yy.txt is not found!
```

6. Peer Server Concurrency

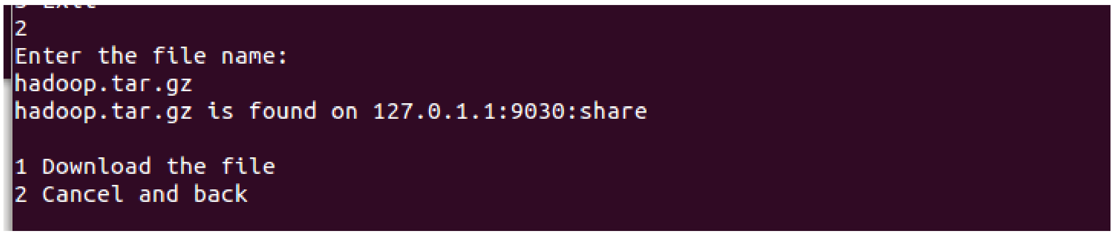
This phase will test the concurrency of peer server. We want to test whether two peers (peer1 and peer2) can download files from peer3 at the same time.

First, peer3 registers two files: hadoop.tar.gz and linpack.tgz



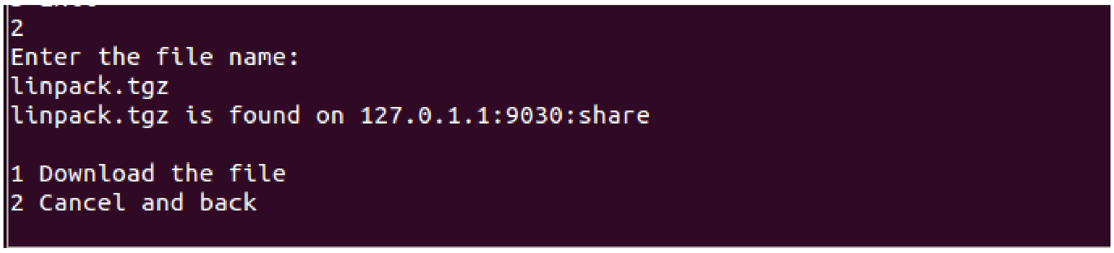
```
1
Enter the file name:
hadoop.tar.gz
File hadoop.tar.gz is registered!

1 Register a file
2 Search a file
3 Exit
1
Enter the file name:
linpack.tgz
File linpack.tgz is registered!
```

Then use peer1 to search for the file hadoop.tar.gz



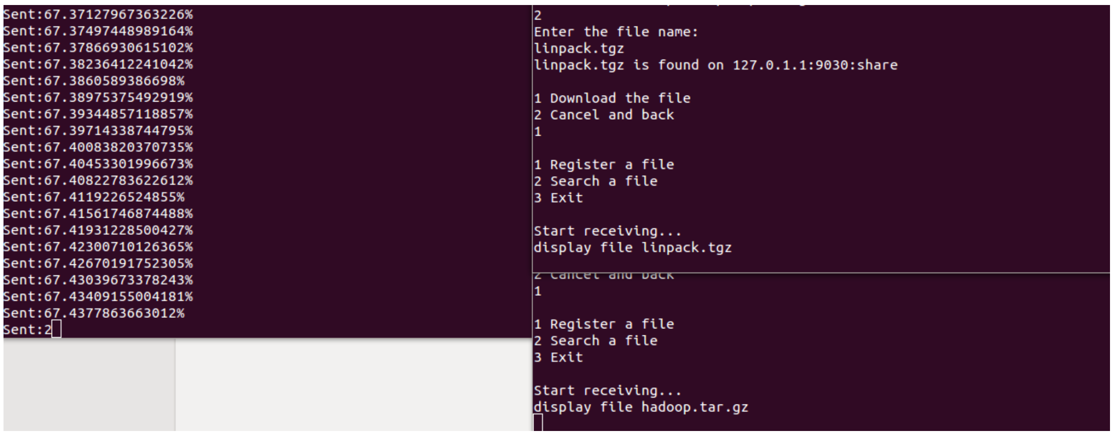
```
2
Enter the file name:
hadoop.tar.gz
hadoop.tar.gz is found on 127.0.1.1:9030:share

1 Download the file
2 Cancel and back
```

Use peer2 to search for the file linpack.tgz



```
2
Enter the file name:
linpack.tgz
linpack.tgz is found on 127.0.1.1:9030:share

1 Download the file
2 Cancel and back
```

Then two peers start to download the files:

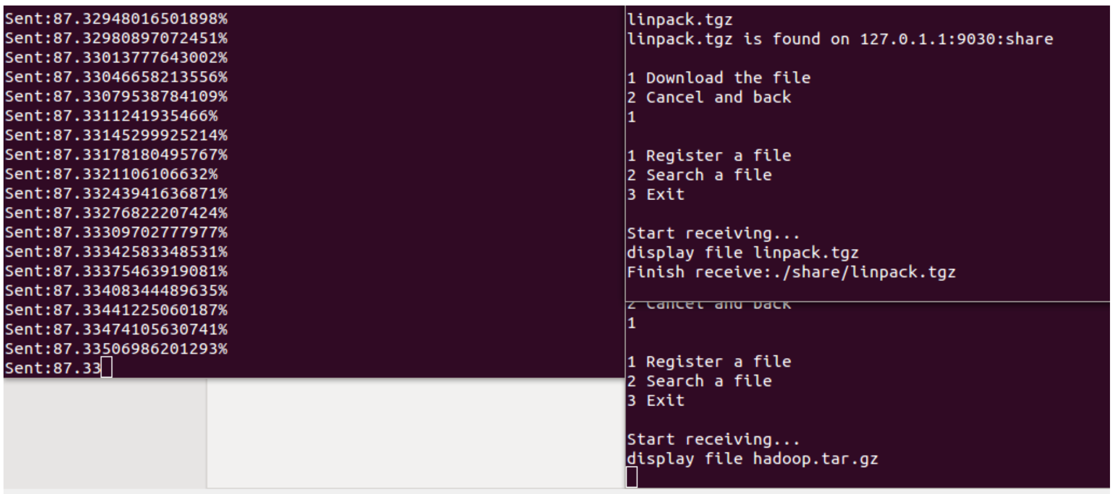


```
Sent:67.37127967363226%
Sent:67.37497448989164%
Sent:67.37866930615102%
Sent:67.38236412241042%
Sent:67.3860589386698%
Sent:67.38975375492919%
Sent:67.39344857118857%
Sent:67.39714338744795%
Sent:67.40083820370735%
Sent:67.40453301996673%
Sent:67.40822783622612%
Sent:67.4119226524855%
Sent:67.41561746874488%
Sent:67.41931228500427%
Sent:67.42300710126365%
Sent:67.42670191752305%
Sent:67.43039673378243%
Sent:67.43409155004181%
Sent:67.4377863663012%
Sent:2
```

```
2
Enter the file name:
linpack.tgz
linpack.tgz is found on 127.0.1.1:9030:share

1 Download the file
2 Cancel and back
1

1 Register a file
2 Search a file
3 Exit

Start receiving...
display file linpack.tgz
```

```
1

1 Register a file
2 Search a file
3 Exit

Start receiving...
display file hadoop.tar.gz
```

The above picture shows that peer1 and peer2 download files at the same time.

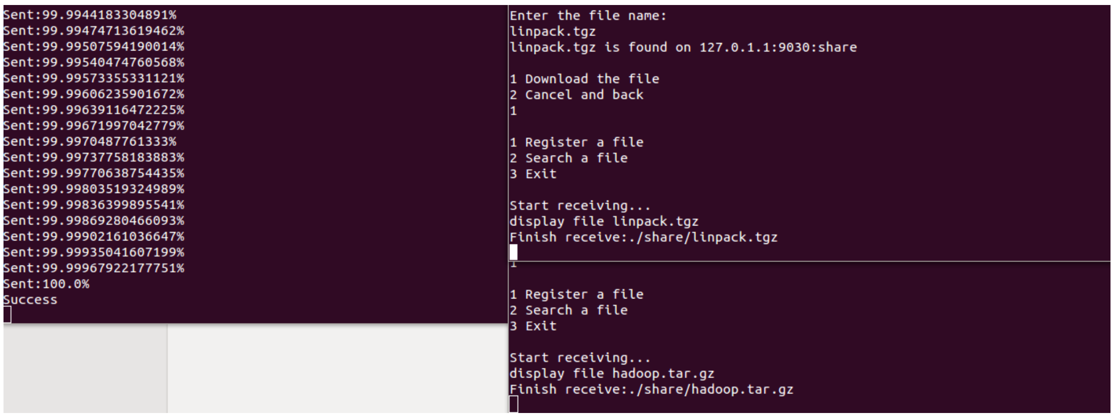


```
Sent:87.32948016501898%
Sent:87.32980897072451%
Sent:87.33013777643002%
Sent:87.33046658213556%
Sent:87.33079538784109%
Sent:87.3311241935466%
Sent:87.33145299925214%
Sent:87.33178180495767%
Sent:87.3321106106632%
Sent:87.33243941636871%
Sent:87.33276822207424%
Sent:87.33309702777977%
Sent:87.33342583348531%
Sent:87.33375463919081%
Sent:87.33408344489635%
Sent:87.33441225060187%
Sent:87.33474105630741%
Sent:87.33506986201293%
Sent:87.33
```

```
linpack.tgz
linpack.tgz is found on 127.0.1.1:9030:share

1 Download the file
2 Cancel and back
1

1 Register a file
2 Search a file
3 Exit

Start receiving...
display file linpack.tgz
Finish receive:./share/linpack.tgz
```

```
2 Cancel and back
1

1 Register a file
2 Search a file
3 Exit

Start receiving...
display file hadoop.tar.gz
```

The above picture shows that peer2 finished receiving, but peer1 had not finished.



```
Sent:99.9944183304891%
Sent:99.99474713619462%
Sent:99.99507594190014%
Sent:99.99540474760568%
Sent:99.99573355331121%
Sent:99.99606235901672%
Sent:99.99639116472225%
Sent:99.99671997042779%
Sent:99.9970487761333%
Sent:99.99737758183883%
Sent:99.99770638754435%
Sent:99.99803519324989%
Sent:99.99836399895541%
Sent:99.99869280466093%
Sent:99.99902161036647%
Sent:99.99935041607199%
Sent:99.99967922177751%
Sent:100.0%
Success
```

```
Enter the file name:
linpack.tgz
linpack.tgz is found on 127.0.1.1:9030:share

1 Download the file
2 Cancel and back
1

1 Register a file
2 Search a file
3 Exit

Start receiving...
display file linpack.tgz
Finish receive:./share/linpack.tgz
```

```
1

1 Register a file
2 Search a file
3 Exit

Start receiving...
display file hadoop.tar.gz
Finish receive:./share/hadoop.tar.gz
```

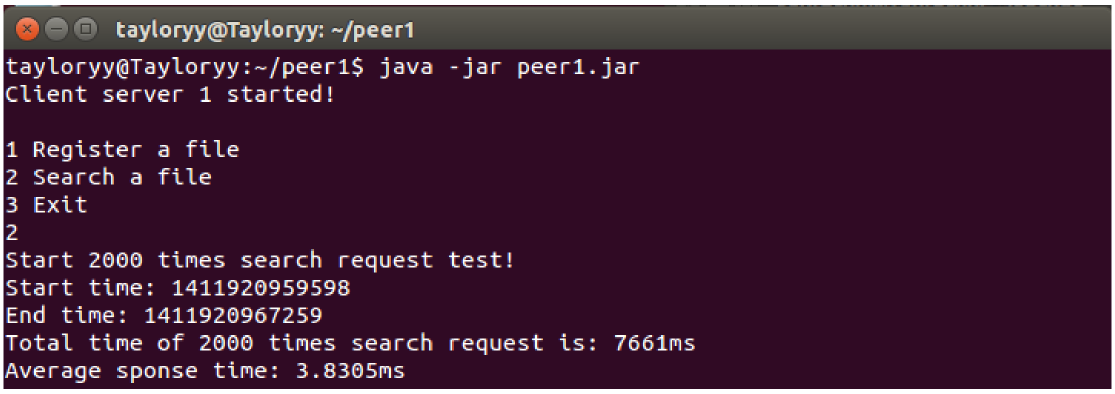
This picture shows that peer1 and peer2 both finished downloading from peer3.

This test proved that the servers of peers can support concurrencies.

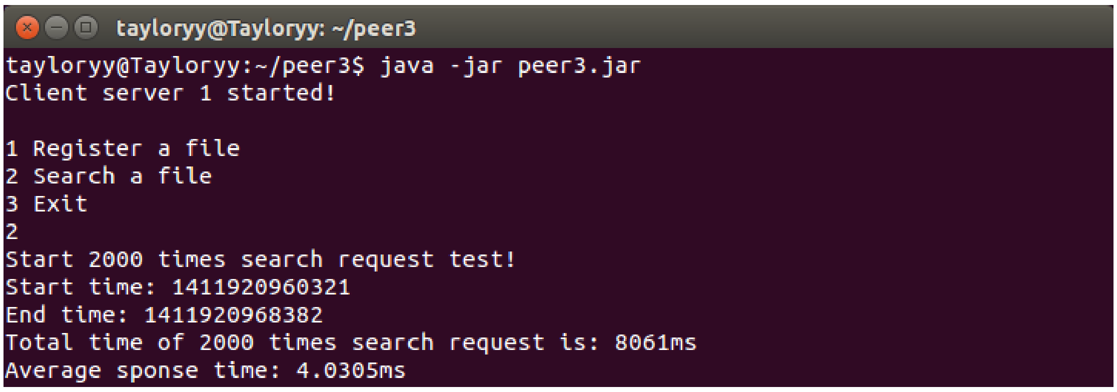
7. Index Server Concurrency

In this part, we run our test for several situations, such as 2 peers concurrent, 3 peers concurrent and 4 peers concurrent. In the following, concurrent tests data of all the situations listed above will be showed through tables and the results will be analyzed according to figures. However, briefly, just the screenshot of the running of 4 peers situation is showed through figures.

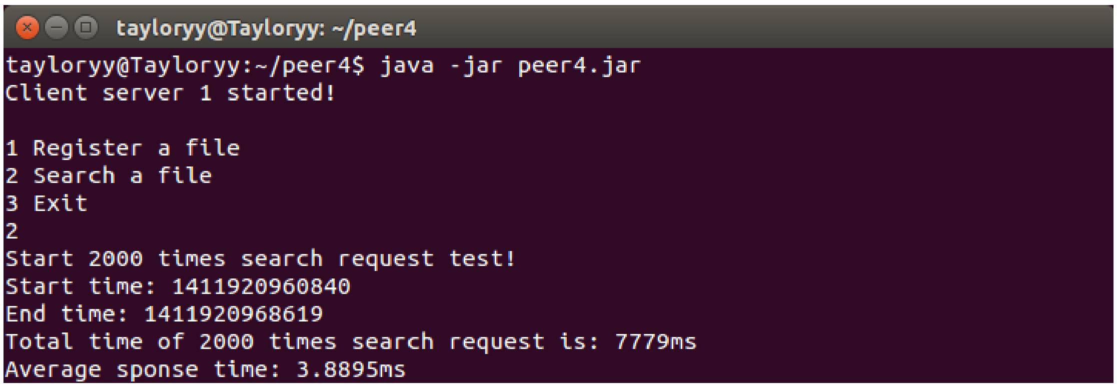
Four Peers Concurrent:



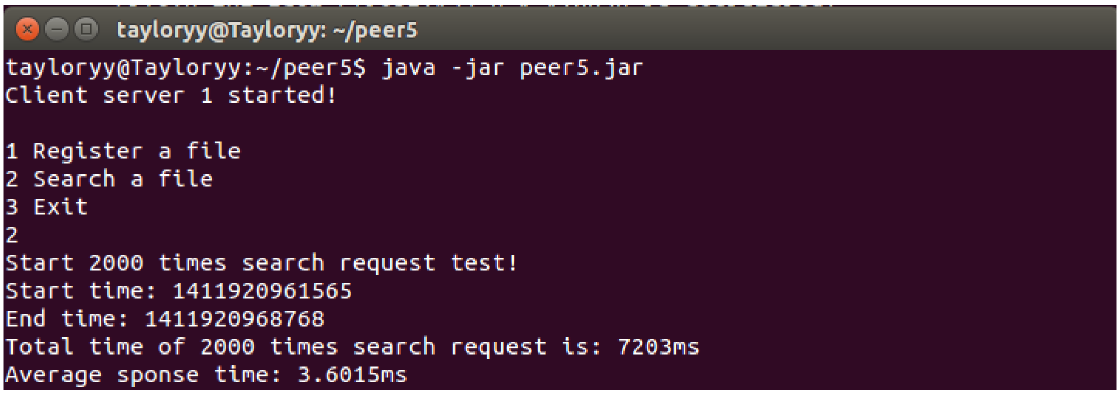
```
tayloryy@Tayloryy:~/peer1$ java -jar peer1.jar
Client server 1 started!

1 Register a file
2 Search a file
3 Exit
2
Start 2000 times search request test!
Start time: 1411920959598
End time: 1411920967259
Total time of 2000 times search request is: 7661ms
Average sponse time: 3.8305ms
```



```
tayloryy@Tayloryy:~/peer3$ java -jar peer3.jar
Client server 1 started!

1 Register a file
2 Search a file
3 Exit
2
Start 2000 times search request test!
Start time: 1411920960321
End time: 1411920968382
Total time of 2000 times search request is: 8061ms
Average sponse time: 4.0305ms
```



```
tayloryy@Tayloryy:~/peer4$ java -jar peer4.jar
Client server 1 started!

1 Register a file
2 Search a file
3 Exit
2
Start 2000 times search request test!
Start time: 1411920960840
End time: 1411920968619
Total time of 2000 times search request is: 7779ms
Average sponse time: 3.8895ms
```



```
tayloryy@Tayloryy:~/peer5$ java -jar peer5.jar
Client server 1 started!

1 Register a file
2 Search a file
3 Exit
2
Start 2000 times search request test!
Start time: 1411920961565
End time: 1411920968768
Total time of 2000 times search request is: 7203ms
Average sponse time: 3.6015ms
```

We can see that peers can request server at the same time, so the index server can support high concurrencies.

In conclusion, all the required functions can be work correctly.