File Sharing System Output

Si Liu
A20334820

In this experiment, we start 4 peers and one index server to generate the output file. Table I displays the peer port and id mapping relation.

Table I Peer port and ID map

Peer Port	Peer Id
5700	10001
5701	10002
5702	10003
5703	10004

1. Search and obtain file output

This system supports obtaining small file (<1G), large file(>1G) and binary file.

1) Small file (< 1G):

```
5700
     [java] Sep 15, 2015 9:20:13 AM org.apache.commons.vfs2.impl.StandardFileSystemManager info
     [java] INFO: Using "/tmp/vfs_cache" as temporary files store.
     [java] This peer is running ...
     [java] Please input command:
search
     [java] Please input the filename to search:
file5K.txt
     [java] This file is located at peers: [10003, 10002]
obtain
     [java] Please input the filename to obtain:
file5K.txt
     [java] Please pick up one peerId from its peer list to download the file:
10003
     [java] The address of the peer 10003 is :127.0.0.1:5702
     [java] Ask for file: file5K.txt
     [java] Done receiving file: file5K.txt
```

Figure 1 Obtain small file from remote peer (Client side)

```
5702
     [java] Sep 15, 2015 9:21:16 AM org.apache.commons.vfs2.impl.StandardFileSystemManager info
     [java] INFO: Using "/tmp/vfs_cache" as temporary files store.
     [java] This peer is running ...
     [java] Please input command:
     [java] Sending file: file5K.txt
```

Figure 2 Obtain small file from remote peer (Server side)

2) Large file (>1G):

```
search
     [java] Please input the filename to search:
file1G.txt
     [java] This file is located at peers: [10004]
obtain
     [java] Please input the filename to obtain:
file1G.txt
     [java] Please pick up one peerId from its peer list to download the file:
10004
     [java] The address of the peer 10004 is :127.0.0.1:5703
     [java] Ask for file: file1G.txt
     [java] Done receiving file: file1G.txt
```

Figure 3 Obtain large file from remote peer (Client side)

```
5703
     [java] Sep 15, 2015 9:21:31 AM org.apache.commons.vfs2.impl.StandardFileSystemManager info
     [java] INFO: Using "/tmp/vfs_cache" as temporary files store.
     [java] This peer is running ...
     [java] Please input command:
     [java] Sending file: file1G.txt
```

Figure 4 Obtain large file from remote peer (Server side)

3) Binary file:

```
search
     [java] Please input the filename to search:
Peer.class
     [java] This file is located at peers: [10003]
obtain
     [java] Please input the filename to obtain:
Peer.class
     [java] Please pick up one peerId from its peer list to download the file:
10003
     [java] The address of the peer 10003 is :127.0.0.1:5702
     [java] Ask for file: Peer.class
     [java] Done receiving file: Peer.class
```

Figure 5 Obtain binary file from remote peer (Client side)

```
     [java] Please input the client port for this peer:
5702
     [java] Sep 15, 2015 9:21:16 AM org.apache.commons.vfs2.impl.StandardFileSystemManager info
     [java] INFO: Using "/tmp/vfs_cache" as temporary files store.
     [java] This peer is running ...
     [java] Please input command:
     [java] Sending file: file5K.txt
     [java] Done sending file: file5K.txtSending file: Peer.class
```

Figure 6 Obtain binary file from remote peer (Server side)

2. Loop up peer address through peer ID

```
loopup
     [java] This command is not supported yet!
     [java] Commands supported:  REGISTER, SEARCH, LOOKUP, OBTAIN, DELETE, EXIT
     [java] Please input command:
lookup
     [java] Please input the peerId to look up:
10003
     [java] The address of the peer 10003 is :127.0.0.1:5702
```

Figure 7 Look up peer address through peer ID

3. Delete one file from a peer

```
delete
     [java] Please input the peerId to delete files from :
10001
     [java] Please input the file to delete from the peer 10001 :
file4K.txt
     [java] Delete OK.
```

Figure 8 Delete from index server the file registration information

4. Automatic registration

As long as peer port is input, automatic registration starts. The index server side will display the result (full file-peer registration table). Figure 9 displays the changes in the index server side for automatic registration:

```
run-server:
     [java] Index Server is running...
     [java] Listening on port 5800.
     [java] The current file-peer registration table is:
     [java] file1K.txt: 10001
     [java] file2K.txt: 10001
     [java] file4K.txt: 10001
     [java] file3K.txt: 10001
     [java]
     [java] The current file-peer registration table is:
     [java] file6K.txt: 10002
     [java] file5K.txt: 10002
     [java] file1K.txt: 10001
     [java] file2K.txt: 10001
     [java] file4K.txt: 10001 10002
     [java] file3K.txt: 10001 10002
     [java] file7K.txt: 10002
     [java]
     [java] The current file-peer registration table is:
     [java] file10K.txt: 10003
     [java] file6K.txt: 10002
     [java] file5K.txt: 10003 10002
     [java] file1K.txt: 10001
     [java] file2K.txt: 10001
     [java] file4K.txt: 10001 10002
     [java] file9K.txt: 10003
     [java] file3K.txt: 10001 10002
     [java] file7K.txt: 10002
     [java] file8K.txt: 10003
     [java]
     [java] The current file-peer registration table is:
     [java] file10K.txt: 10003
     [java] file6K.txt: 10002
     [java] file5K.txt: 10003 10002
     [java] file1K.txt: 10001
     [java] file2K.txt: 10001
     [java] file1G.txt: 10004
     [java] file4K.txt: 10001 10002
     [java] file9K.txt: 10003
     [java] file3K.txt: 10001 10002
     [java] file7K.txt: 10002
```

Figure 9 Peer automatically start registration at Index Server