Contents

[1.0 Introduction 2](#_Toc494478637)

[2.0 User Manual 3](#_Toc494478638)

[2.1 Environment setup: 3](#_Toc494478639)

[2.2 Compiling the java code: 3](#_Toc494478640)

[2.3 Executing code: 4](#_Toc494478641)

[3.0 Limitations and further improvements: 6](#_Toc494478642)

# 1.0 Introduction

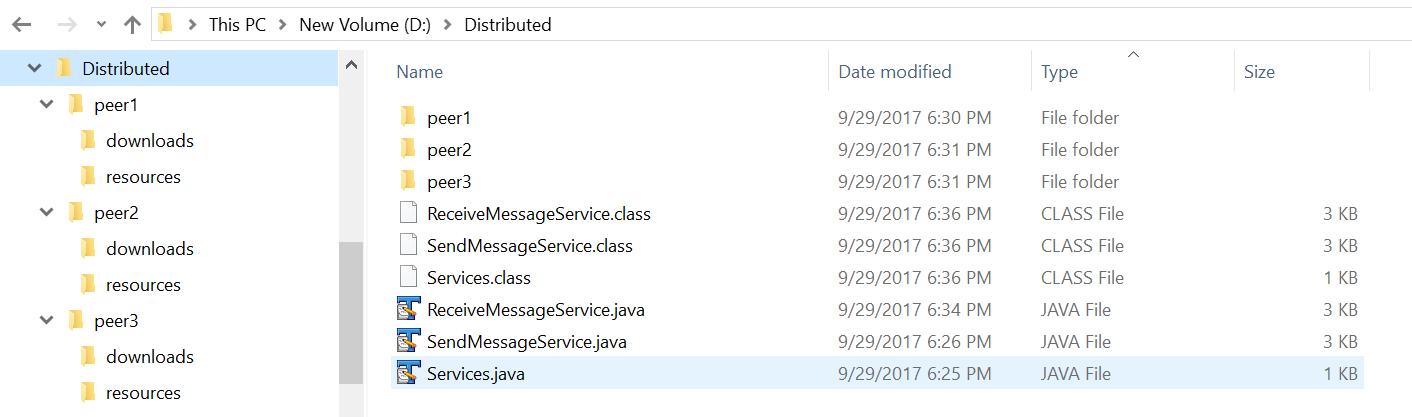
The application is developed to demonstrate the distributed file/message sharing. Multicast socket is heavily used to distribute same message among the peers in group. Similarly, client server communication using TCP socket is used to share the resource directly via socket. The application uses the concept of both client server communication and multicast message broadcasting service. Each peer joining multicast group will have unique id and send the message to each member of the group. Each group member except the requesting peer will check if they hold the requested resource and share file via socket. The requesting peer will check the input stream in the TCP socket and download the resource if available.

First section of this report will demonstrate the configuration setup. In second part, it will demonstrate the compilation of java code. In third section, it will present a short system walkthrough. Finally, the report with list down some of the limitations and enhancement suggestion on the application. Some of the input parameters are hardcoded for testing purpose.

# 2.0 User Manual

## 2.1 Environment setup:

The multicast group IP, Multicast port and TCP port are hard coded to “225.5.6.7”, 9998, and 9999 respectively. 3 peers (peer1, peer2, peer3) are joined the multicast group for demonstration. Each pair hold separate resource / download folder. If the peer 3 requests file (say file1), each peer looks for file availability if their resource folder and share through server socket (port 9999). In the meantime, the requesting pair will for input stream in the TCP socket.



Before running the application in the java installation must be checked. It can be checked by executing “java -version” command in command prompt.

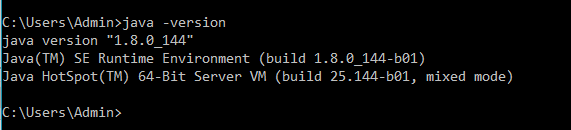


Figure 1: Checking java version

## 2.2 Compiling the java code:

1. Navigate to the folder where the codes are stored (For example in this case “D:\Distributed”).
2. Execute the command “javac \*.java”.

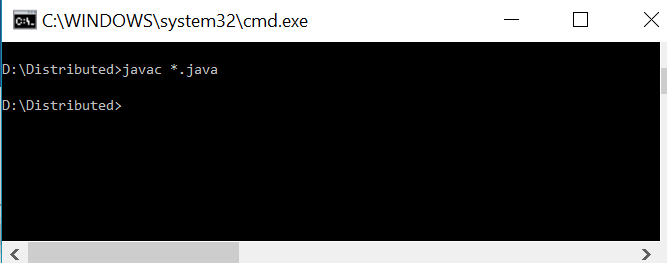


Figure 2: compiling the java code

1. If the compilation is successful, system will create class files of each java class.

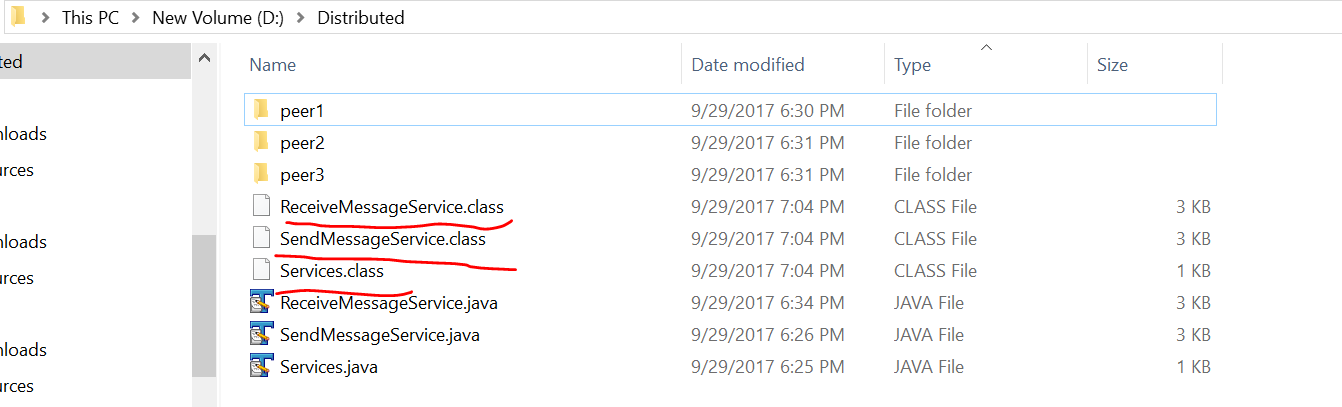


Figure 3: Showing the list of class files

1. Now the java class is compiled successfully and ready to execute. The main class name is Services.class and it expects one input argument as a peer unique id. Also, the main class executes 2 other class ReceiveMessageService and SendMessageService in the thread to enable the two-way communication. Let’s execute the same code with 3 different pair Id (peer1, peer2,peer3).

## 2.3 Executing code:

1. Let’s execute the java code with 3 different peer Id (peer1, peer2, peer3):

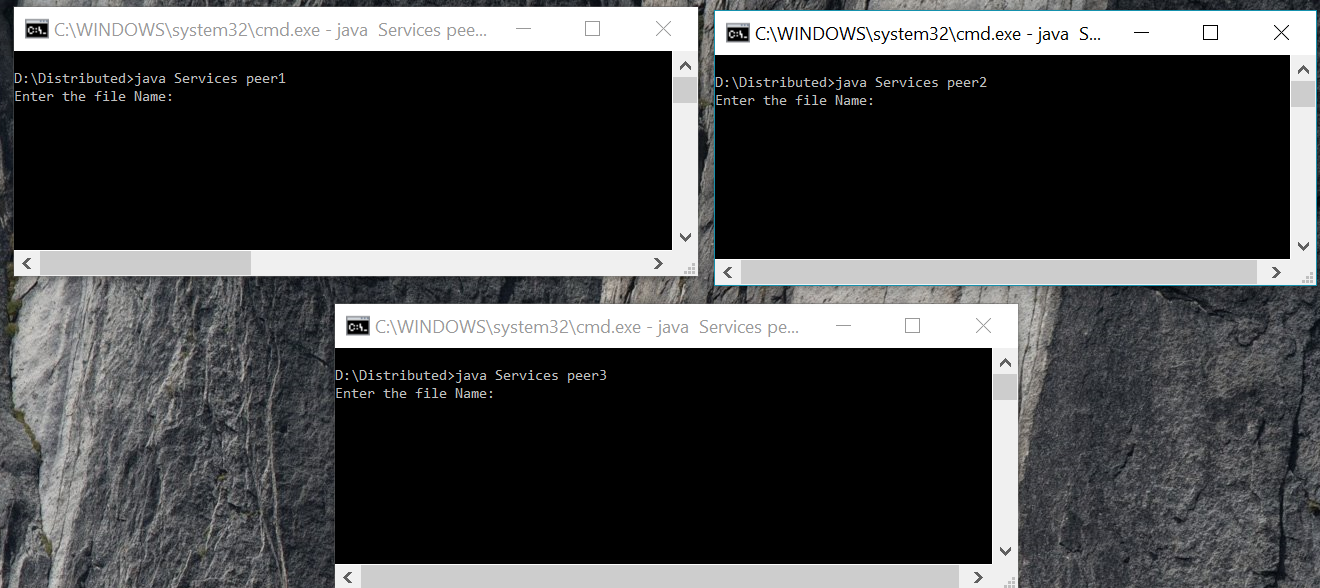


Figure 4: Execution of java code with different peer id

1. Let’s check the resource folder of each peer. Peer1 and peer2 have files file1 and file2 respectively. Whereas peer 3 has no resource.

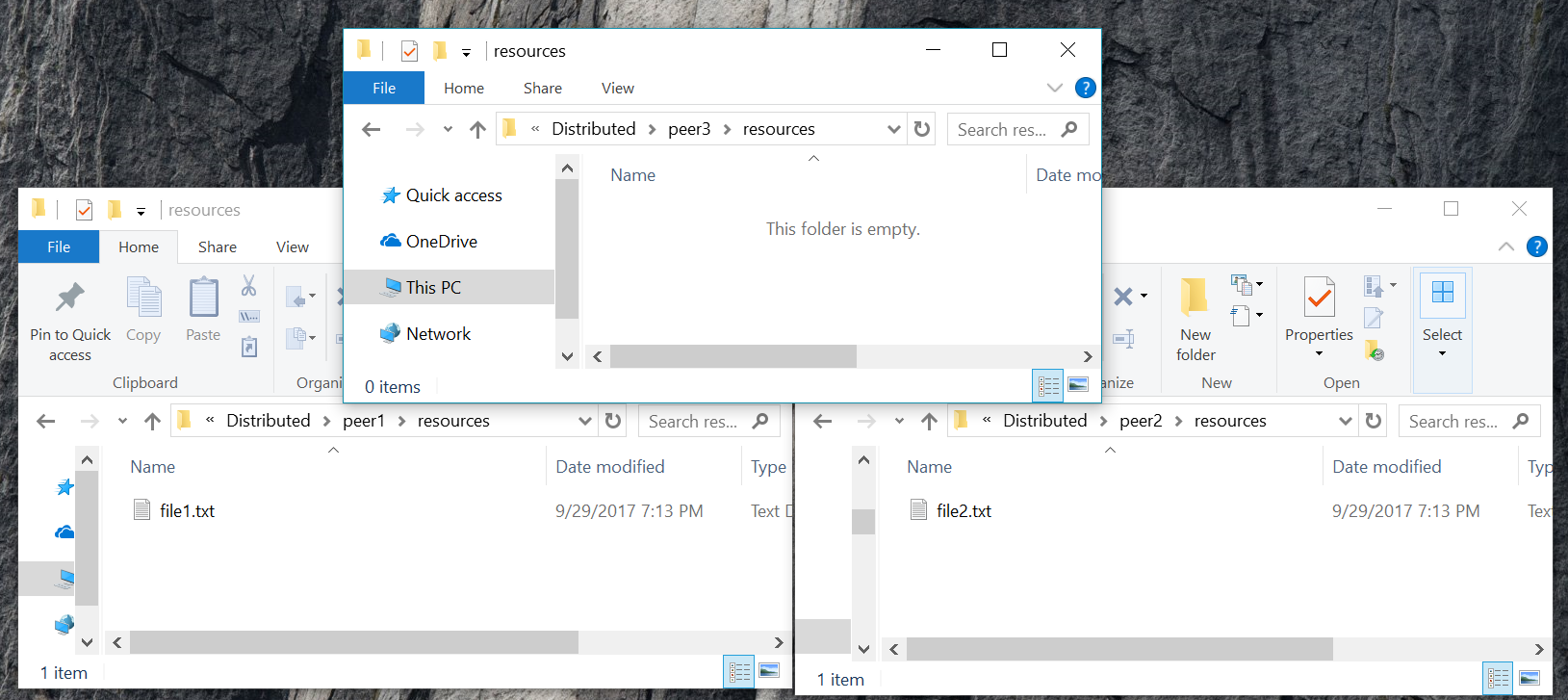


Figure 5: Exploring resource folder of each peer

1. Let’s request file1 from peer3 and see the result:

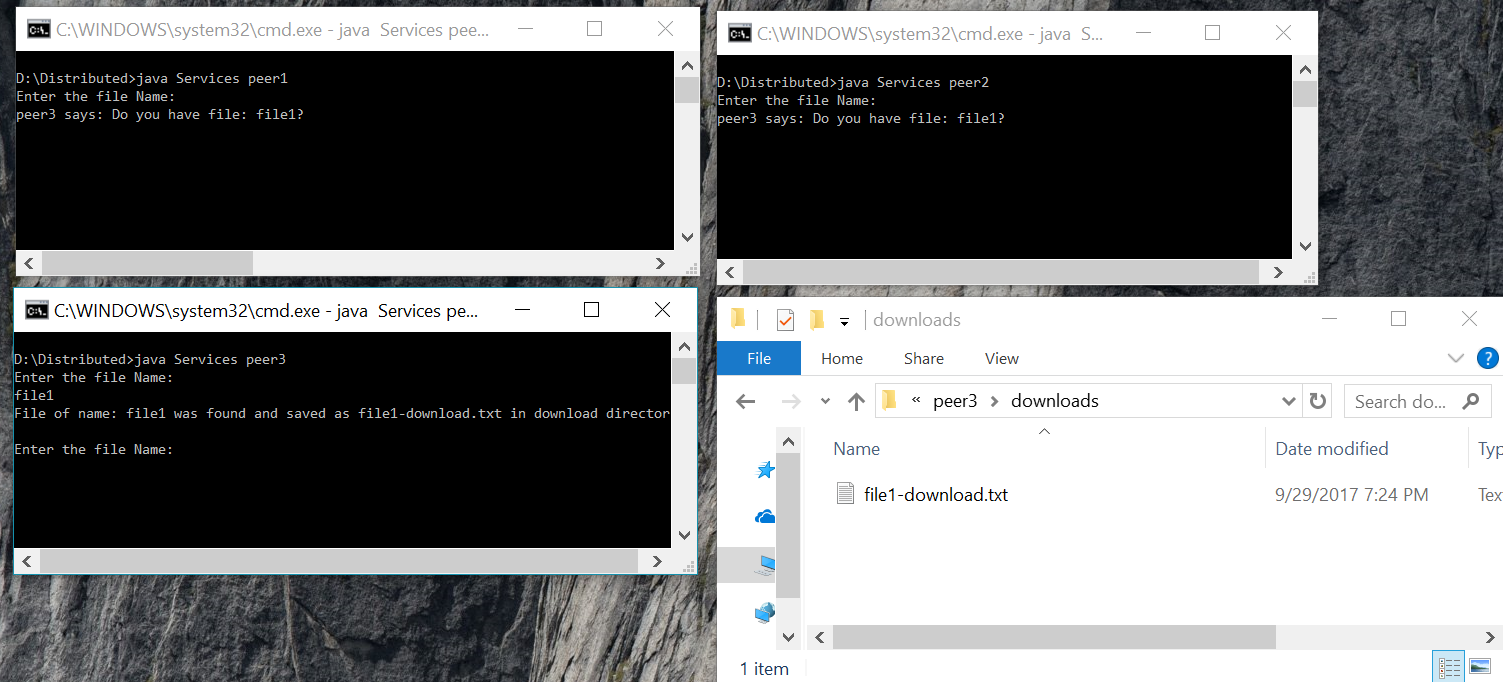


Figure 6: Showing the message broadcast and file download

Peer 3 requested the file “file1”. Each peer except peer3 see the message requested file name. Now, peer1 and peer 2 check the file in their corresponding resource folder. Since peer1 has the requested file it sends the copy file into TCP socket as an output stream. Peer3 checks the input stream and download the file in its download folder if the file available. The fourth snapshot in above screen shows the downloaded file.

Once the file transfer is successful, the peer1 closes the socket so that next request can be accepted.

Now let’s try to check different file name from peer3.

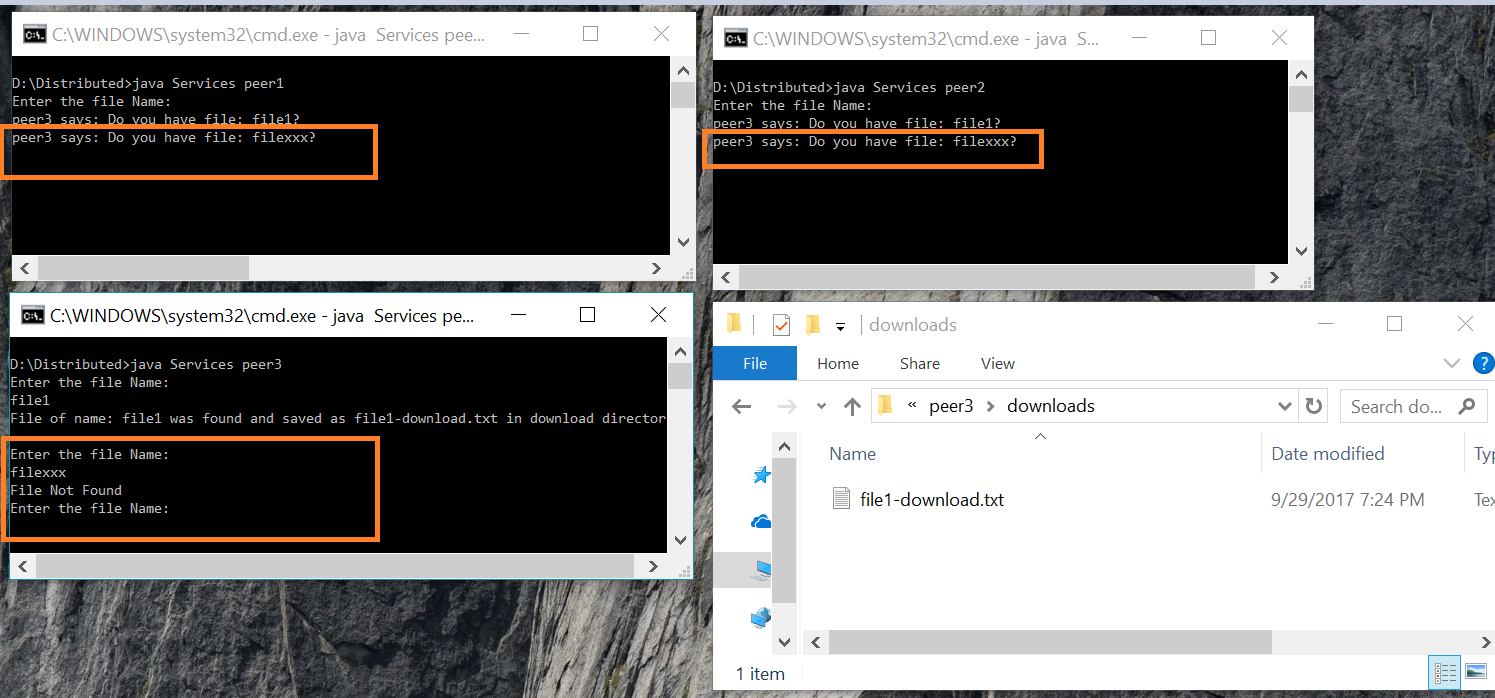


Figure 7: Response while search unavailable file

Since the file is not available in any of the peer’s resource folder, socket doesn’t open. Peer three check the socket and it notices that the TCP socket is not open. Then it generates the message file not found.

Similarly, peer3 can download file from peer 2 as well in the similar manner.

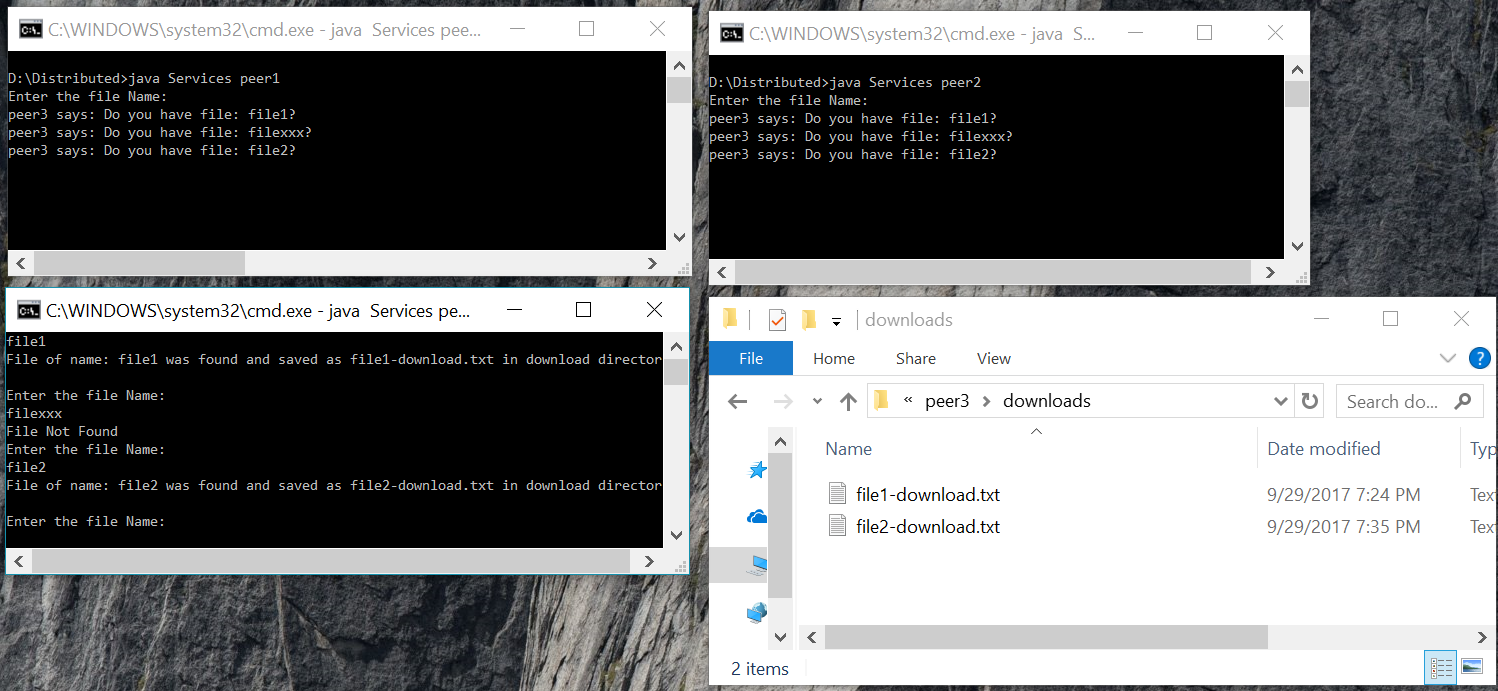


Figure 8: Showing file download from other peer

If peer 1 request the file contained in its own resource, it won’t be able to find the file because, other members only perform the file checking. Also, the broadcast message will be populated to other group members only.

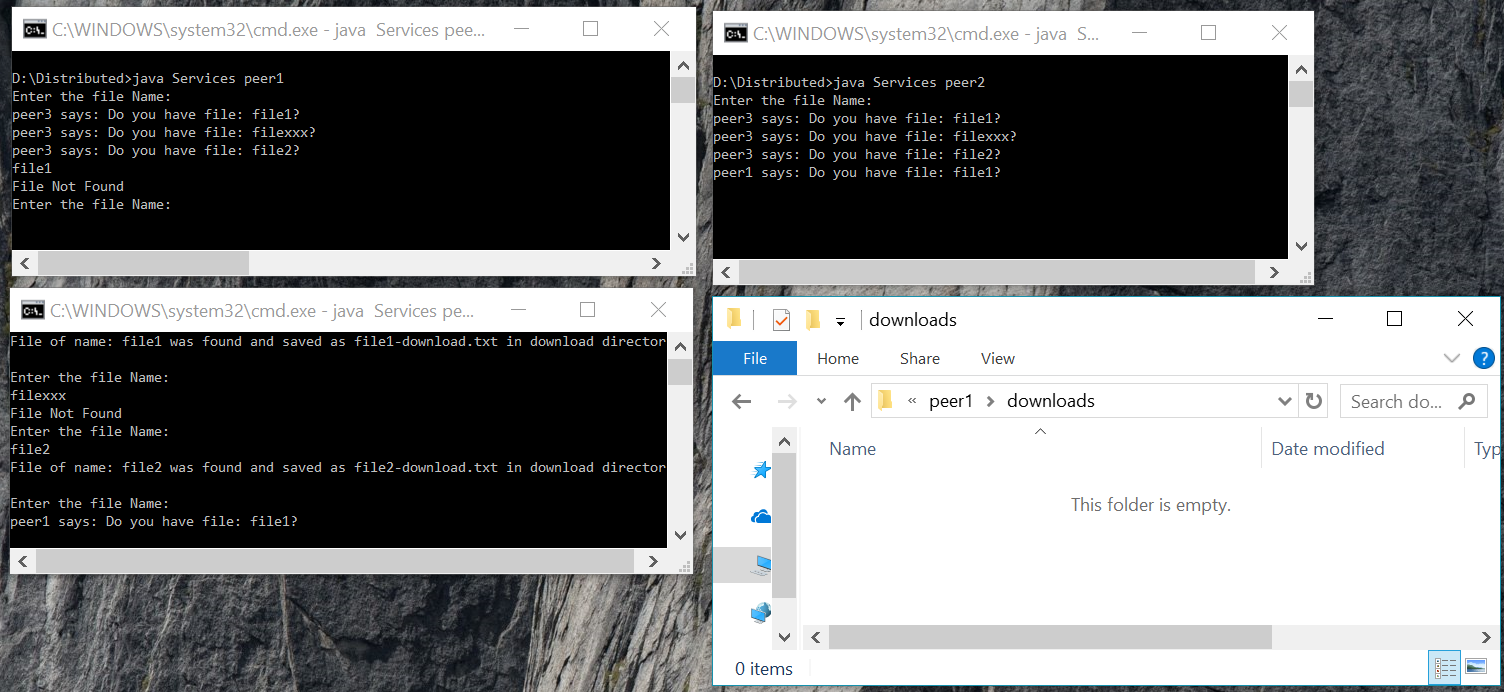


Figure 9:Checking own file download

# 3.0 Limitations and further improvements:

Currently, the application supports the transfer of text file only. System bandwidth can be the bottleneck if we transfer the large files at once. Also, there is no any security mechanism applied to control the sharing of information.

Below are the further changes suggested in this application:

1. We can change the logic to transfer other types of data such as images/videos/documents.
2. Handshaking can be performed to confirm the file transfer among the peers.
3. Security certificate can be adopted to allow members to join the multicast group.
4. The scope of this application is in local area network only. We can enhance to enable the communication over internet.
5. The application can be very useful for intracompany message/file sharing if we adopt some GUI (Graphical User Interface) and user authentication.