HO CHI MINH CITY, VIETNAMESE-GERMAN UNIVERSITY

DISTRIBUTED SYSTEMS REPORT

P2P FILE SHARING SYSTEM

TEAM: 6

Supervisor: Dr. Phan Trong Nhan

Student:	Nguyen Truc Linh
ID:	10421088
Student:	Pham Phi Long
ID:	10421034
Student:	Thai Minh Kien
ID:	10421087
Student:	Dang Hoang Anh Khoi
ID:	10421031
Student:	Dao The Hien
ID:	10421074

June 12, 2024

Content

1 Introduction

A peer-to-peer(P2P) file-sharing system is a network technology where peers are able to share files with each other without a need of a server. There are several structures of P2P file-sharing system. One of those is hybrid P2P system, which we decided to implement for our project.

2 Problem Description

2.1 Goal of the problem

A peer can stably connect to the host server, uploading and downloading the files. The system has almost full function as a normal P2P file-sharing system, the GUI is easy to use for client and server can handle any connections error without raising any. error on the client-side.

2.2 How it works

First, each peer needs to enter the Folder Directory they want to share, the Client Port, the hosting server address will be given by the server device, and the Server Port in our project will be 3434. After a successful connection, both the server and the client will see shared files from other clients. Finally, peers just need to look up the file name and download it.

3 Implementation Details

3.1 Used libraries

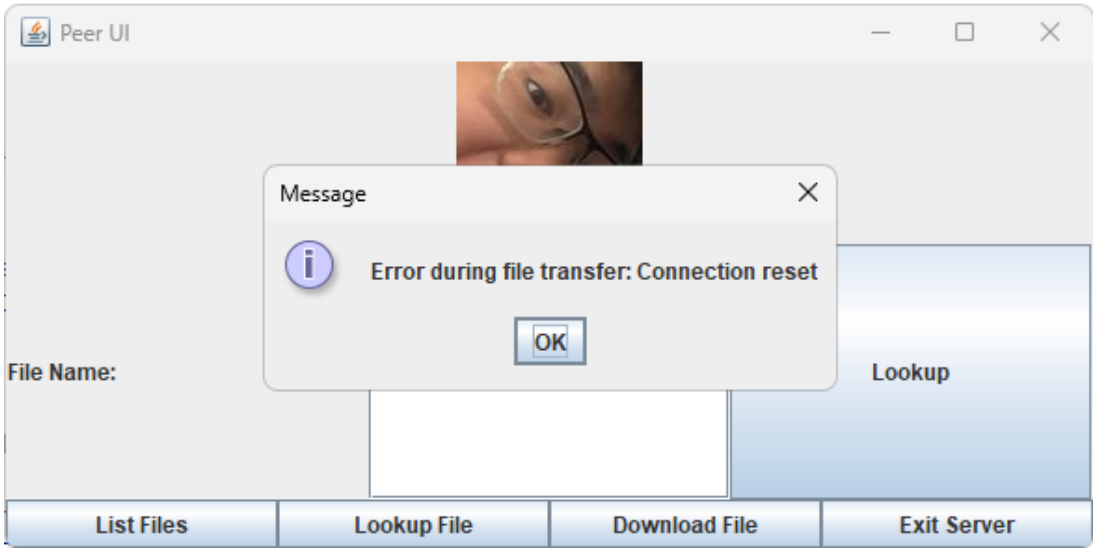
Thanks to Java's diverse libraries, it has helped us a lot in development. We utilized the following libraries:

- java.io.BufferedReader;
- java.io.DataInputStream;
- java.io.DataOutputStream;
- java.io.EOFException;
- java.io.File;
- java.io.FileOutputStream;
- java.io.FileInputStream;
- java.io.IOException;
- java.io.InputStream;
- java.io.InputStreamReader;

- java.io.OutputStream;
- java.io.PrintWriter;
- java.net.InetAddress;
- java.net.Socket;
- java.net.ServerSocket;
- java.net.URL;
- java.nio.file.FileSystems;
- java.nio.file.Path;
- java.nio.file.Paths;
- java.nio.file.StandardWatchEventKinds;
- java.nio.file.WatchEvent;
- java.nio.file.WatchKey;
- java.nio.file.WatchService;
- java.util.ArrayList;
- java.util.Arrays;
- java.util.Hashtable;
- java.util.HashSet;
- java.util.HashMap;
- java.util.LinkedList;
- java.util.List;
- java.util.Set;
- java.util.concurrent.ExecutorService;
- java.util.concurrent.Executors;
- javax.swing.*;
- java.awt.*;
- java.awt.event.*;

3.2 Fault tolerance

In case a peer downloads a file but the peer that owns that file leaves the server. The system will display a message to notify you of that. The file being downloaded will only download until the connection is disconnected.



3.3 Scalability

We have simulated the situation where multiple clients connect at the same time. We simulated by creating a code called "Fake Client" so we could adjust the number of clients connected. After testing many times with different numbers, we found that the system would work best for about 500 clients.

3.4 Architecture

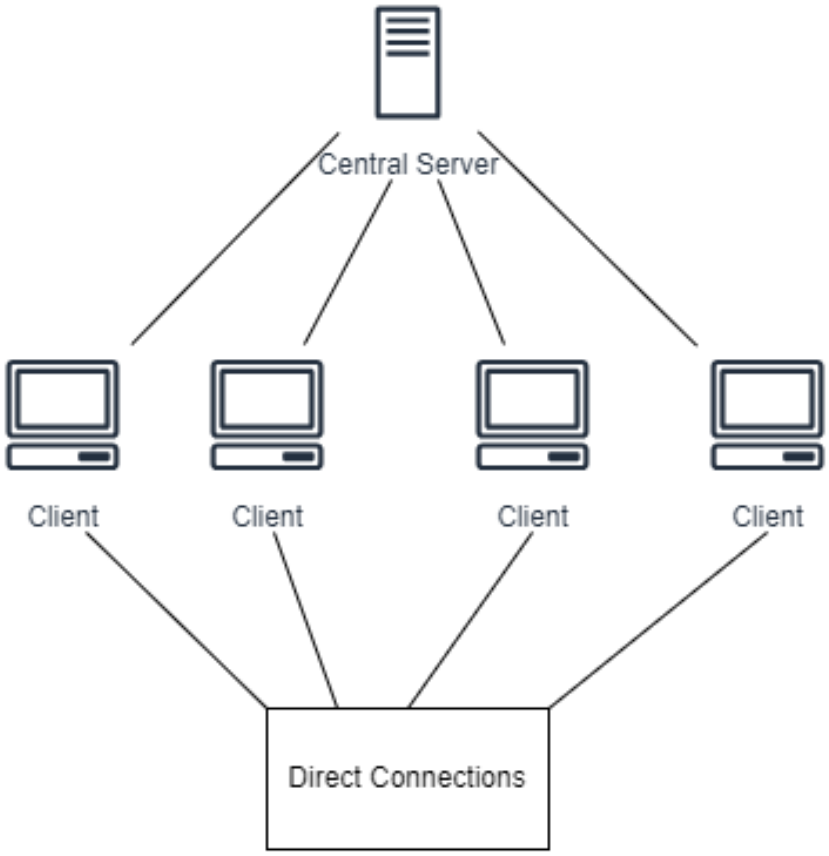


Figure 1: Hybrid P2P file-sharing architecture

We have the central peer with peers in the network acting as a client. The role of the central peer is to show the information about connected peers, and the uploaded file names. In particular, the central peer only has an intermediary role to display information, not a place to store files.

3.5 Methodology

In this report, we only mention the code of important function of the p2p file sharing system, so there will be some missing code compare to the file code we submit.

1. Main features:

   - List files: List files Code
     After connecting, you can click the "List Files" button. A list of uploaded files will be displayed with name and type.

   - Search files: Search files Code
     First, you can click the "Lookup File" button. Next, you can enter the desired file name. Then, after searching, if any peer owns the file you need, it will display "File Found!!!".

- Download files: Download files Code
  Once you have completed the two sections above and found the file you need, you can download it with the "Download" button.

2. Peer registration: Register Code
   First, each peer needs to enter the Folder Directory they want to share, the Client Port, the hosting server address will be given by the server device, and the Server Port in our project will be 3434. After a successful connection, both the server and the client will see shared files from other clients.

3. Peer Disconnect: Disconnect Code
   This will announce the server that which peer has disconect from the server.

3.6 Code

- Register code:
  - Server

```
1  public void run() {
2  try {
3      DataInputStream dIn = new DataInputStream(socket.↩
           getInputStream());
4      DataOutputStream dOut = new DataOutputStream(socket.↩
           getOutputStream());
5
6      byte option = dIn.readByte();
7      switch (option) {
8          case 0:
9
10             Boolean end = false;
11             ArrayList<String> fileNames = new ArrayList<String↩
                   >();
12             int numFiles = 0, port = 0;
13             String directory = null, address = null;
14
15             while (!end) {
16                 byte messageType = dIn.readByte();
17
18                 switch (messageType) {
19                     case 1:
20                         numFiles = dIn.readInt();
21                         break;
22                     case 2:
```

```
23                          System.out.println("\nNew peer ↩
                               registering with " + numFiles + " ↩
                               files:");
24                          for (int i = 0; i < numFiles; i++) {
25                              fileNames.add(dIn.readUTF());
26                              System.out.println(fileNames.get(i↩
                                   ));
27                          }
28                          break;
29                      case 3:
30                          directory = dIn.readUTF();
31                          break;
32                      case 4:
33                          address = dIn.readUTF();
34                          break;
35                      case 5:
36                          port = dIn.readInt();
37                          break;
38                      default:
39                          end = true;
40                  }
41              }
42              boolean check = false;
43              for (int i = 0; i < CentralIndexingServer.↩
                    saveAddress.size(); i++){
44                  if (CentralIndexingServer.saveAddress.get(i).↩
                        equals(address) && CentralIndexingServer.↩
                        savePort.get(i).equals(port)){
45                      peerid = i + 1;
46                      check = true;
47
48                      break;
49                  }
50              }
51
52              if (check == false){
53                  CentralIndexingServer.savePort.add(port);
54                  CentralIndexingServer.saveAddress.add(address)↩
                        ;
55                  peerid = CentralIndexingServer.saveAddress.↩
                        size();
56              }
57
58              System.out.println("\nPeer ID: " + peerid + ", ↩
                    Address: " + address + ", Port: " + port);
```

```
59              System.out.println("↩
                    ----------------------------------");
60
61              synchronized (this) {
62                  CentralIndexingServer.registry(peerid, ↩
                        numFiles, fileNames, directory, address, ↩
                        port);
63              }
64
65              dOut.writeInt(peerid);
66              dOut.flush();
67              socket.close();
68              break;
```

In this void run method of server side, we create a switch case to handle multiple actions of client, this switch case has 5 main cases, first case is to connect peer to server, second case is for search file, case number three is to list all the available file in the system, next case is to detect peer disconnection, final case is to track if the user modified their shared folder, system will update it.

For the option number zeroth, the case, which receive the shared folder, IP address, port id, will be triggered. In this case, we create an array list to hole the list of file name user share. After that we implement the while loop to iterate through 5 cases, each case responsible to take one information of user, which are: number of files, list of file name, the directory, address and port, the final case is to modify the while loop condition to end it. In addition, we also create a way to track if a peer is register for the first time or he/she has already register before and now he/she just re-register. If he/she register for the first time, the system will assign the unique peer id to it, if they disconnect and re-connect, the exactly same peer id will be assign to them again, not the new one. For this method, we create the array list to store the address of each log in peer, if the register address and port is the same with any available address and port in the list, that peer will be assigned to their unique id, which is the i+1(the peer id start from 1). On the other hand, if they are not in the list, their information will be stored and they will be assigned a new unique peer id.

– Client

```
1  public void runClient(String folderDirectory, int clientPort, ↩
       String serverAddress, int serverPort) throws IOException {
2  String dir = folderDirectory;
3  File folder = new File(dir);
4  String address = InetAddress.getLocalHost().getHostAddress();
5
```

```
6  ArrayList<String> fileNames = Util.listFilesForFolder(folder);
7
8  final Peer peer = new Peer(dir, fileNames, fileNames.size(), ↩
       address, clientPort);
9
10 Socket socket = new Socket(serverAddress, serverPort);
11 peer.register(socket);
12
13 int peerId = peer.getPeerId();
14
15 if (count == 0){
16     JOptionPane.showMessageDialog(null, "Peer " + peerId + " ↩
           is registered");
17 }
18 count++;
19
20 new Thread(() -> {
21     try {
22         WatchService watcher = FileSystems.getDefault().↩
               newWatchService();
23         Path dir1 = Paths.get(dir);
24         dir1.register(watcher, StandardWatchEventKinds.↩
               ENTRY_CREATE, StandardWatchEventKinds.ENTRY_DELETE,↩
                StandardWatchEventKinds.ENTRY_MODIFY);
25
26         while (true) {
27             WatchKey key;
28             try {
29                 key = watcher.take();
30             } catch (InterruptedException ex) {
31                 return;
32             }
33
34             for (WatchEvent<?> event : key.pollEvents()) {
35                 WatchEvent.Kind<?> kind = event.kind();
36
37                 if (kind == StandardWatchEventKinds.OVERFLOW) ↩
                       {
38                     continue;
39                 }
40
41                 WatchEvent<Path> ev = (WatchEvent<Path>) event↩
                       ;
42                 Path fileName = ev.context();
43
```

```
44                  System.out.println(kind.name() + ": " + fileName);
45
46                  if (kind == StandardWatchEventKinds.ENTRY_CREATE || kind == StandardWatchEventKinds.ENTRY_DELETE || kind == StandardWatchEventKinds.ENTRY_MODIFY) {
47                      try {
48                          runClient(folderDirectory, clientPort, serverAddress, serverPort);
49                          Socket newSocket = new Socket(serverAddress, serverPort);
50                          peer.notifyFileDeletion(deletedFiles, newSocket);
51                      } catch (IOException e) {
52                          System.err.println("Failed to register peer: " + e);
53                      }
54                  }
55              }
56
57              boolean valid = key.reset();
58              if (!valid) {
59                  break;
60              }
61          }
62      } catch (IOException ex) {
63          System.err.println(ex);
64      }
65  }).start();
```

For the client-side code, we create a method named runClient to connect the client to the central server, in this method we store all the necessary register information of user, we also created an array list in this method to hold all the filename of peers, then we registered peer to the server via the register method on the server-side, which I have introduce before. In addition, we used the WebService API of java to keep track the condition of shared folder, for instance if user change the content, modify name, create or delete any file in their shared-folder, the WebService will detect and will announce the system so that the peer that has the modified folder will be registered once again to refresh his/her folder.

- Method

```
1     public void register(Socket socket) throws IOException {
```

```
2
3           long start = System.currentTimeMillis();
4           DataOutputStream dOut = new DataOutputStream(socket.↩
                getOutputStream());
5
6           dOut.writeByte(0);
7           dOut.flush();
8
9           dOut.writeByte(1);
10          dOut.writeInt(numFiles);
11          dOut.flush();
12          dOut.writeByte(2);
13          for(String str : fileNames)
14              dOut.writeUTF(str);
15          dOut.flush();
16          dOut.writeByte(3);
17          dOut.writeUTF(directory);
18          dOut.flush();
19          dOut.writeByte(4);
20          dOut.writeUTF(address);
21          dOut.flush();
22          dOut.writeByte(5);
23          dOut.writeInt(port);
24          dOut.flush();
25          dOut.writeByte(-1);
26          dOut.flush();
27
28          DataInputStream dIn = new DataInputStream(socket.↩
                getInputStream());
29          this.peerId = dIn.readInt();
30
31          dOut.close();
32          dIn.close();
33          socket.close();
34
35          System.out.println("Running as Peer " + peerId + "! " ↩
                + "It Took " + (System.currentTimeMillis() - start)↩
                 + "ms for register.");
36      }
```

This is the register method that the client used to connect and send register information to the central server, on the client side, we have introduce the method to take each information a time by using switch case. Therefore in this method, we also send the information one by one using writeByte and change the option when sending so that the server can switch

to the correct case to receive the correct type of information.

- List code:
  - Server

```
case 2: // Handle "LIST_FILES" request
    Set<Peer> allPeers2 = CentralIndexingServer.getAllPeers();
    Set<String> allFileNames = new HashSet<>(); // Use a HashSet to avoid duplicates

    for (Peer p : allPeers2) {
        //p.refreshFileList();
        allFileNames.addAll(p.getFileNames());
    }

    try {
        dOut.writeInt(allFileNames.size()); // Send the number of unique files
        dOut.flush();
        for (String file : allFileNames) {
            dOut.writeUTF(file); // Send each unique file name
            dOut.flush();
        }
    } catch (IOException e) {
        System.err.println("An I/O error occurred while sending file list: " + e.getMessage());
    }
    socket.close();
    break;
```

The list file on the server, just simply iterate through all the peers and then collect all their filenames then send it to the peer who require. In this method, we use set datatype for both peers and filenames to avoid duplicate file and peer.

  - Client

```
public List<String> listServerFiles(Socket socket) throws IOException {
    DataOutputStream dOut = new DataOutputStream(socket.getOutputStream());
    DataInputStream dIn = new DataInputStream(socket.getInputStream());

    // Send request to list files (option 2)
    dOut.writeByte(2);
```

```
7       dOut.flush();
8
9       // Read the number of files from the server
10      int numFiles = dIn.readInt();
11      List<String> fileNames = new ArrayList<>();
12
13      // Read each file name
14      for (int i = 0; i < numFiles; i++) {
15          fileNames.add(dIn.readUTF());
16      }
17
18      dOut.close();
19      dIn.close();
20      socket.close();
21
22      return fileNames; // Return the list of file names
23  }
```

On the server side, when user choose to list file, server will send the option signal number 2 to server, the server will switch to case number two, which is list file(these switch case on the server side have been mentioned above), then it will read the number of file and then create a for loop to read all the file names from the server, finally it will return the file list.

- Lookup code:
  - Server

```
1   case 1:
2       String fileName = dIn.readUTF();
3       Set<Peer> allPeers = CentralIndexingServer.getAllPeers();
4       // for (Peer p : allPeers) {
5        p.refreshFileList();
6       // }
7       Boolean b = search(fileName);
8
9       try {
10          Thread.sleep(1);
11      } catch (InterruptedException e) {
12          e.printStackTrace();
13      }
14
15      if (b) {
16          dOut.writeByte(1);
```

```
17          dOut.writeInt(peerList.size());
18          dOut.flush();
19
20          for (Peer p : peerList) {
21              dOut.writeUTF(p.getAddress() + ":" + p.getPort() + ":"↩
                    + p.getPeerId());
22              dOut.flush();
23          }
24      } else {
25          dOut.writeByte(0);
26          dOut.flush();
27      }
28      socket.close();
29      break;
```

on the server side, this is the case number one, in this case, first we will let the server receive the requirement file from peers then we create an search method to check if the file is available. If the file is available, we will send the signal to the client side to handle the rest, if the looked up file is not exist we also send an announcement to the client side. This is the search method we created to search if file is exist or not:

```
1  public Boolean search(String fileName) {
2      newPeerList();
3      return (CentralIndexingServer.getIndex().containsKey(fileName)↩
           )
4              ? addAllPeer(CentralIndexingServer.getIndex().get(↩
                   fileName))
5              : false;
6  }
```

– Client

```
1      public String[] lookup(String fileName, Socket socket, int ↩
           count) throws IOException{
2          DataOutputStream dOut = new DataOutputStream(socket.↩
               getOutputStream());
3
4          //Option to look for a file
5          dOut.writeByte(1);
6
7          String [] peerAddress = new String[0];
8
9          //File name
```

```
10          dOut.writeUTF(fileName);
11          dOut.flush();
12          //System.out.println("Reading from the server...");
13
14          System.out.println("Peer " + peerId + " - looking for file
                . (" + count + ")");
15
16          //Reading the peer Address that has the file
17          DataInputStream dIn = new DataInputStream(socket.
                getInputStream());
18          byte found = dIn.readByte();
19
20          if(found == 1){
21              int qt = dIn.readInt();
22
23              peerAddress = new String[qt];
24
25              for(int i = 0; i < qt; i++){
26                  try{
27                      peerAddress[i] = dIn.readUTF();
28                  }catch (EOFException e){
29                      i--;
30                  }
31                  // String paddress[] = peerAddress[i].split(":");
32                  // System.out.println("Peer " + paddress[2] + " - 
                        " + paddress[0] +":" + paddress[1] + " has the
                         file " + fileName + "! - Looked by Peer " + 
                        peerId);
33              }
34          } else if(found == 0){
35              System.out.println("File not found in the system");
36              peerAddress = new String[0];
37          }
38
39          dOut.close();
40          dIn.close();
41          socket.close();
42          return peerAddress;
43      }
```

On the client-side, firstly, the client will send signal to trigger the look up case on server side, the it will send the file name and receive the result from the server after searching for the file, if the file is available, the system will return the ip of the peer who keeping that file. If the file is not available, the system will announce user that the file is not exist.

- Download code:
  - Method

```
1  public static String copy(InputStream in, OutputStream out) throws IOException {
2          String message = "";
3          byte[] buffer = new byte[1024];
4          int count = 0;
5          long start = System.currentTimeMillis();
6          try {
7              while ((count = in.read(buffer)) != -1) {
8                  out.write(buffer, 0, count);
9              }
10         } catch (IOException e) {
11             message = "Can't continue download file, Peer is disconnected";
12             throw e; // rethrow the exception to be handled by the caller
13         }
14         // System.out.println("Download took " + (System.currentTimeMillis() - start) + " ms.");
15         message = "Download took " + (System.currentTimeMillis() - start) + " ms.";
16         return message;
17     }
```

In this method, we took 2 parameter which are out is the peer who want to download the file and in is peer who keep the file. We use the while loop to read all bytes of file from the input peer then write it to the peer who want to download the file. We also put an exception to handle when the keeper of the file is disconnect while someone is downloading his/her file. Finally we display the how long it took to download the file, if it downloaded successfully.

  - Client

```
1  public String download(String peerAddress, int port, String fileName, int i, String inputdirectory) throws IOException {
2      String message = "";
3      Socket socket = new Socket(peerAddress, port);
4      DataOutputStream dOut = new DataOutputStream(socket.getOutputStream());
5      dOut.writeUTF(fileName);
6      InputStream in = socket.getInputStream();
```

```
7
8       String peerDirectory = inputdirectory;
9       File folder = new File(peerDirectory);
10      Boolean created = false;
11      if (!folder.exists()) {
12          try {
13              created = folder.mkdir();
14          } catch (Exception e) {
15              // System.out.println("Couldn't create the folder,↩
                    the file will be saved in the current ↩
                   directory!");
16              message = "Couldn't create the folder, the file ↩
                   will be saved in the current directory!";
17          }
18      } else {
19          created = true;
20      }
21
22      if (i != -1) {
23          fileName = fileName + i;
24      }
25
26      OutputStream out = (created) ? new FileOutputStream(↩
           peerDirectory + "/" + fileName) : new FileOutputStream(↩
           fileName);
27
28      try {
29          message = Util.copy(in, out);
30          System.out.println(message);
31      } catch (IOException e) {
32          // System.err.println("Error during file transfer: " +↩
                e.getMessage());
33          message = "Error during file transfer: " + e.↩
               getMessage();
34          // Handle the error (e.g., by retrying the download, ↩
               notifying the user, etc.)
35      }
36      dOut.close();
37      out.close();
38      in.close();
39      socket.close();
40      return message;
41  }
```

To download a given file and store it to a predetermined directory, the download

method establishes a connection with a peer. Sending the filename over the network, the procedure first creates a message string and connects to the peer via a socket. After that, it becomes ready to receive data via an input stream from the peer.

The method checks if the specified directory exists and attempts to create it if it does not. It sets a flag to indicate whether the directory was successfully created. The filename is modified if necessary, and an output stream is created to write the file to the specified directory or the current directory.

In order to handle any IO exceptions that may arise during the transfer, the method tries to copy data from the input stream to the output stream. In order to reflect any errors, it modifies the message string appropriately. After closing the socket and all streams, the procedure returns a message with the download operation's outcome.

- Disconnect Code

Server side:

```
1 case 3:
2     String message = dIn.readUTF();
3
4     if (message.startsWith("DISCONNECT ")) {
5         String peerId = message.substring(11); // Get the peerId ↩
              from the message
6         System.out.println("Peer " + peerId + " has disconnected")↩
              ;
7
8     }
9     break;
```

This code snippet receive a string from the client, if the string start with "DISCONNECT" then it will take the peer id by extract the input string and take from the 11th character. Finally it will print out which peer has disconnect.

```
1 public void disconnect(Socket socket) throws IOException {
2     DataOutputStream dOut = new DataOutputStream(socket.↩
          getOutputStream());
3     try {
4         dOut.writeByte(3);
5         dOut.flush();
6
7         dOut.writeUTF("DISCONNECT " + this.peerId);
```

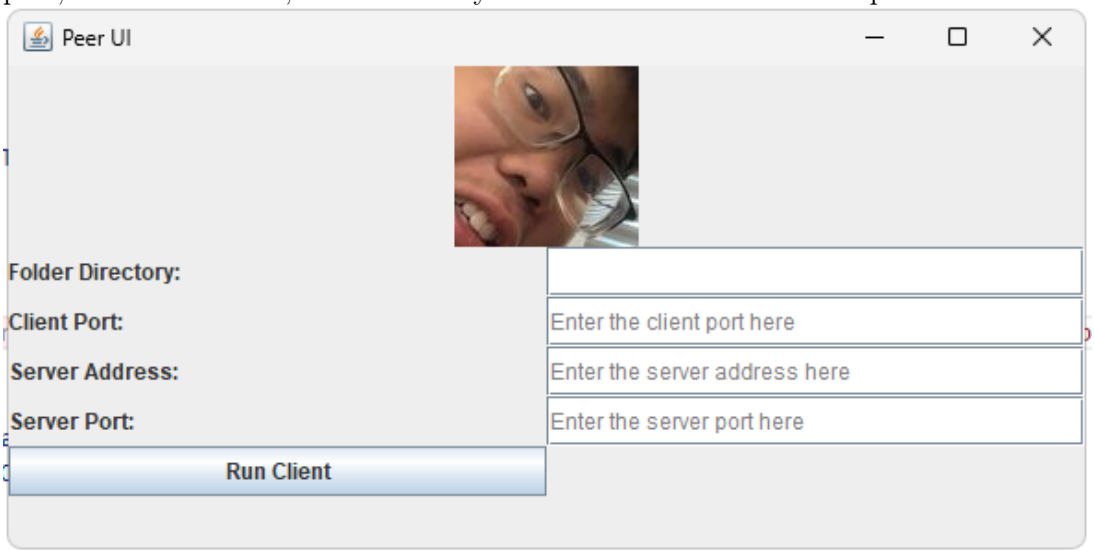
```
8           dOut.flush();
9       } catch (IOException e) {
10          System.out.println("Error while disconnecting: " + e.↩
                getMessage());
11      }
12      dOut.close();
13  }
```

This method will send signal number 3 to the server, this will trigger case number 3 on server side which will check the disconnect status. After that, it will send the DISCONNECT corresponding to the peer id.

3.7 GUI

3.7.1 General

When running the Client.java code, the GUI will pop up and require peer to input their peer port, address of server, the folder they want to share and the server port like this:



This is the code we use to create this GUI:

```
1     public JTextField folderDirectoryField = createCustomTextField("↩
          Enter the folder directory here");
2   public JTextField clientPortField = createCustomTextField("Enter the ↩
      client port here");
3   public JTextField serverAddressField = createCustomTextField("Enter ↩
      the server address here");
4   public JTextField serverPortField = createCustomTextField("Enter the ↩
      server port here");
5   public JTextField fileNameField;
6
7   public Client() {
```

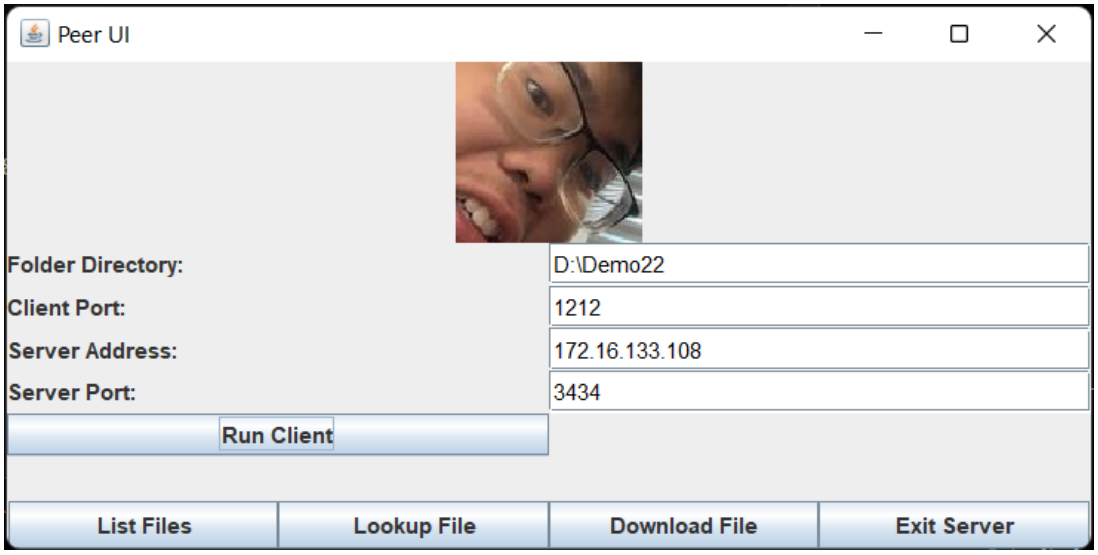
```
8       frame = new JFrame("Peer UI");
9       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10      frame.setSize(600, 300);
11      frame.setLocationRelativeTo(null);
12
13
14      ImageIcon background = new ImageIcon("/Users/binnu/Library/↩
            CloudStorage/OneDrive-student.vgu.edu.vn/Semester 6/Distributed↩
             System/P2P_PROJECT/p2p-file-sharing-system/simple-p2p-file-↩
            sharing copy/background.jpg"); // Replace with your image path
15      backgroundLabel = new JLabel();
16      backgroundLabel.setIcon(background);
17      backgroundLabel.setHorizontalAlignment(JLabel.CENTER);
18      frame.add(backgroundLabel, BorderLayout.NORTH);
19
20
21      panel = new JPanel();
22      panel.setLayout(new GridLayout(6, 2));
23
24      panel.add(new JLabel("Folder Directory:"));
25      folderDirectoryField.setForeground(Color.GRAY);
26      folderDirectoryField.setText("Enter the folder directory here");
27      panel.add(folderDirectoryField);
28
29      panel.add(new JLabel("Client Port:"));
30      clientPortField.setForeground(Color.GRAY);
31      clientPortField.setText("Enter the client port here");
32      panel.add(clientPortField);
33
34      panel.add(new JLabel("Server Address:"));
35      serverAddressField.setForeground(Color.GRAY);
36      serverAddressField.setText("Enter the server address here");
37      panel.add(serverAddressField);
38
39      panel.add(new JLabel("Server Port:"));
40      serverPortField.setForeground(Color.GRAY);
41      serverPortField.setText("Enter the server port here");
42      panel.add(serverPortField);
43
44      runClientButton = new JButton("Run Client");
45      runClientButton.addActionListener(new runClientButtonListener());
46      panel.add(runClientButton);
47
48      frame.add(panel);
49      frame.setVisible(true);
```

```
50
51 }
```

When user enter all of the necessary information and press run client button. This is the code to take user's input and put those parameter into runClient method:

```
1     public class runClientButtonListener implements ActionListener {
2
3     public void actionPerformed(ActionEvent e) {
4         String folderDirectory = folderDirectoryField.getText();
5         int clientPort = Integer.parseInt(clientPortField.getText());
6         String serverAddress = serverAddressField.getText();
7         int serverPort = Integer.parseInt(serverPortField.getText());
8
9         try {
10            runClient(folderDirectory, clientPort, serverAddress, ↩
                  serverPort);
11
12        } catch (IOException ex) {
13            ex.printStackTrace();
14        }
15    }
16 }
```

after user press run client button, another panel at the bottom of the frame will be pop up to provide user more options like "list file", "look up", "download file" or "exit".



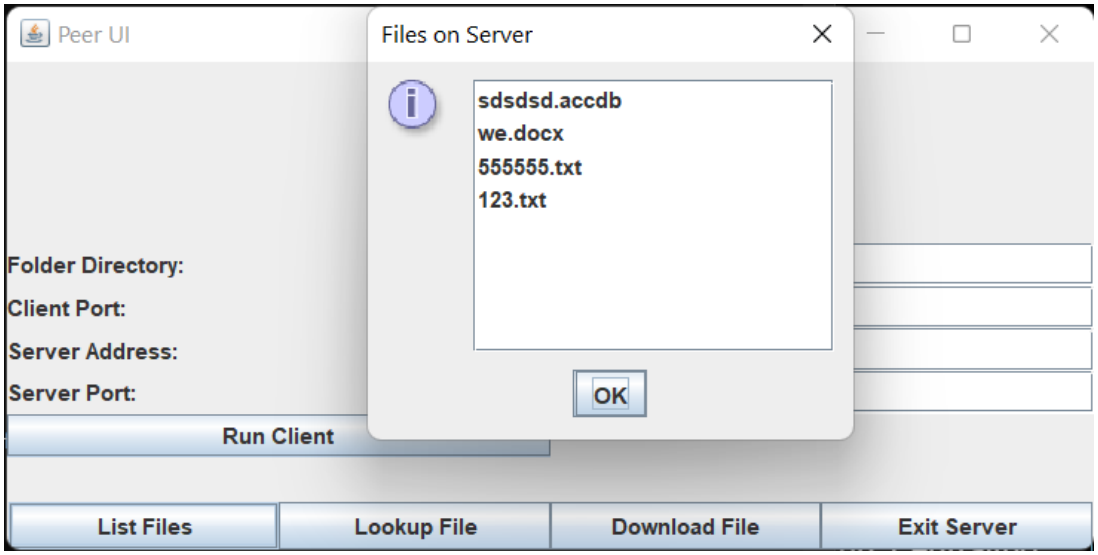
And this is the code to create those button:

```
        JPanel buttonPanel = new JPanel(new GridLayout(1, 4));
    // buttonPanel.add(new JLabel("Options:"));
    listFilesButton = new JButton("List Files");
    lookupButton = new JButton("Lookup File");
    downloadButton = new JButton("Download File");
    exitButton = new JButton("Exit Server");

    buttonPanel.add(listFilesButton);
    buttonPanel.add(lookupButton);
    buttonPanel.add(downloadButton);
    buttonPanel.add(exitButton);

    frame.add(buttonPanel, BorderLayout.SOUTH);
```

When User press the list file button, another small windows will appear and display all the available files in the network of all peers like this:

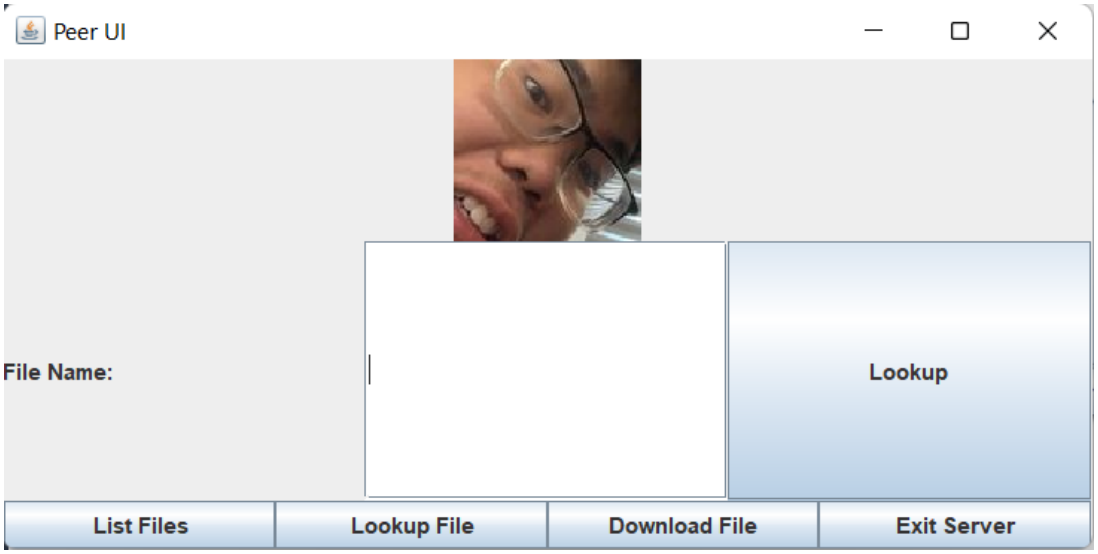


This is the code behind, it use listServerFiles method to list file of all peers, this function have been covered above in the main features:

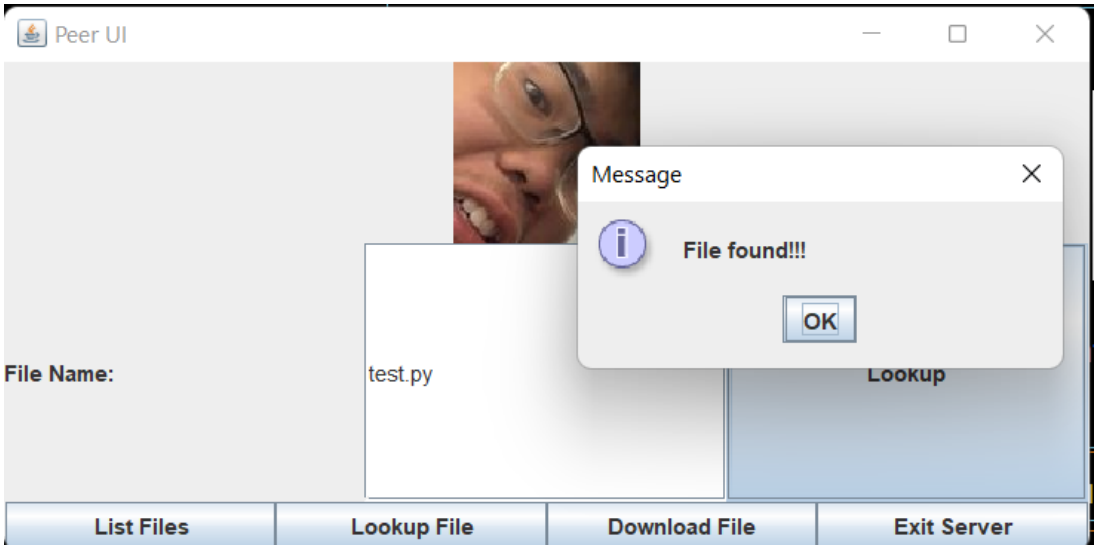
```
listFilesButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            new Thread(new Runnable() {
                public void run() {
                    try {
                        System.out.println("Attempting to list files ↩
```

```
                    from server...");
7                   List<String> serverFiles = peer.↩
                        listServerFiles(new Socket(↩
                        serverAddressField.getText(), Integer.↩
                        parseInt(serverPortField.getText()));
8                   if (serverFiles.isEmpty()) {
9                       System.out.println("No files found on the ↩
                            server.");
10                  } else {
11                      System.out.println("Files received from ↩
                            server: " + serverFiles);
12                  }
13                  SwingUtilities.invokeLater(new Runnable() {
14                      public void run() {
15                          JList<String> fileList = new JList<>(↩
                                serverFiles.toArray(new String[0]))↩
                                ;
16                          JOptionPane.showMessageDialog(frame, ↩
                                new JScrollPane(fileList), "Files ↩
                                on Server", JOptionPane.↩
                                INFORMATION_MESSAGE);
17                      }
18                  });
19              } catch (IOException ex) {
20                  ex.printStackTrace();
21                  System.out.println("Failed to list files from ↩
                        server: " + ex.getMessage());
22              }
23          }
24      }).start();
25  }
26 });
```

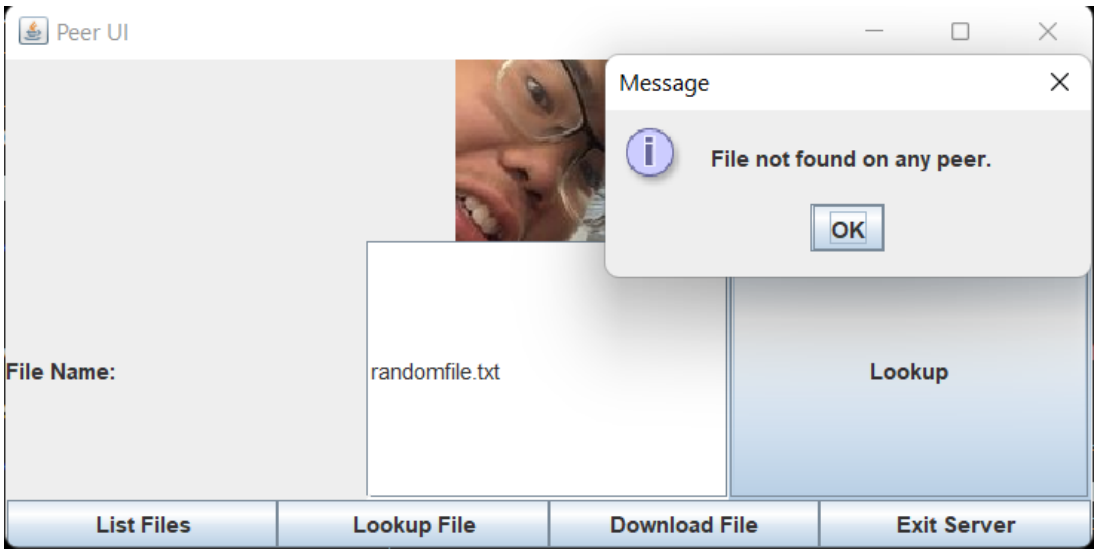
The second button I want to mention is look up button, When user press this button, another frame will pop up. In this new frame, there will an empty box that allow user to input the file they want to download, after that, they will press the Lookup button next to the empty box so that the system will search for the file.



If the file is available, a new windows will appear and announce user about that file like this:



Another example when we try to look up for a non-existence file:

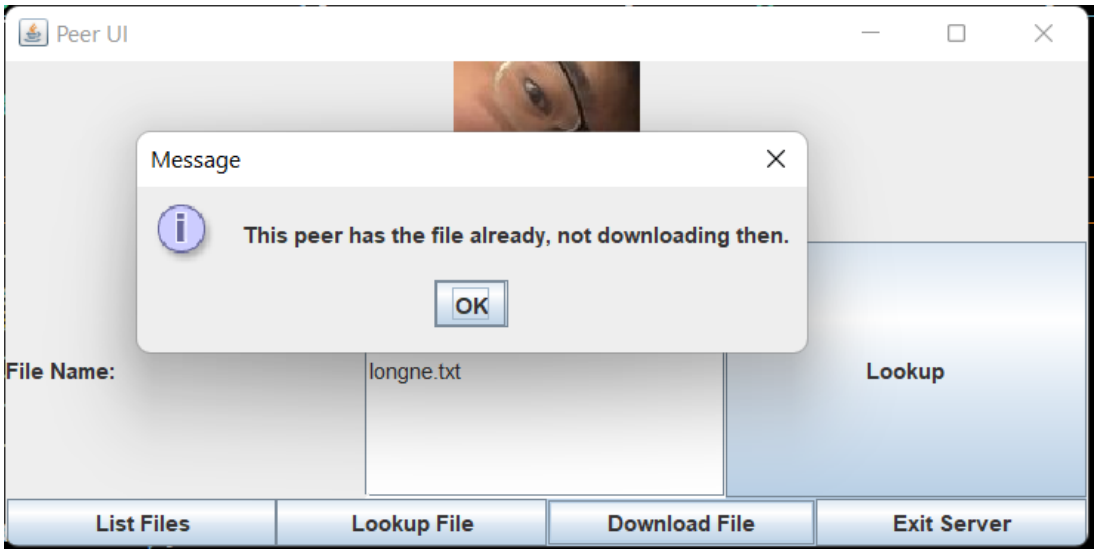


This is all the code we created to handle all the cases when user press lookup button:

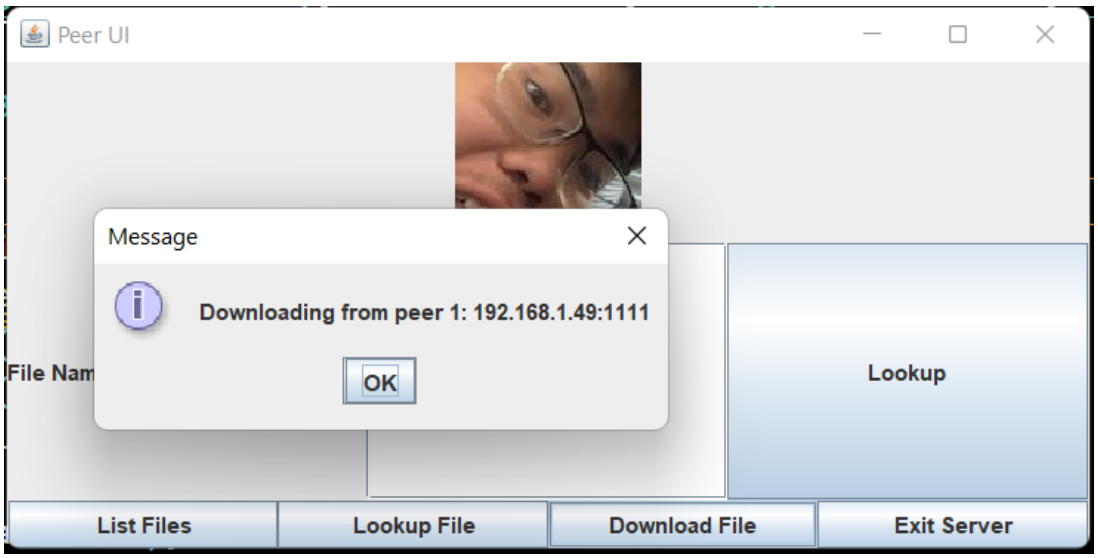
```
1     lookupButton.addActionListener(new ActionListener() {
2     public void actionPerformed(ActionEvent e) {
3         panel.removeAll();
4         panel.setLayout(new GridLayout(1,3));
5         panel.add(new JLabel("File Name:"));
6         fileNameField = new JTextField();
7         panel.add(fileNameField);
8         JButton lookupButton2 = new JButton("Lookup");
9         lookupButton2.addActionListener(new ActionListener() {
10            public void actionPerformed(ActionEvent e) {
11                String fileName = fileNameField.getText();
12                try {
13                    peerAddress = peer.lookup(fileName, new Socket(↩
                          serverAddressField.getText(), Integer.parseInt(↩
                          serverPortField.getText())), 1);
14                    // Display the lookup result
15                    StringBuilder message = new StringBuilder();
16
17
18                    if(peerAddress.length == 0) {
19                        message.append("File not found on any peer.");
20                    } else {
21
22                        message.append("File found!!!");
23                    }
24                    JOptionPane.showMessageDialog(null, message.↩
                          toString());
25
26                } catch (IOException ex) {
```

```
                    ex.printStackTrace();
                }
            }
        });
        panel.add(lookupButton2);
        panel.revalidate();
        panel.repaint();
    }
});
```

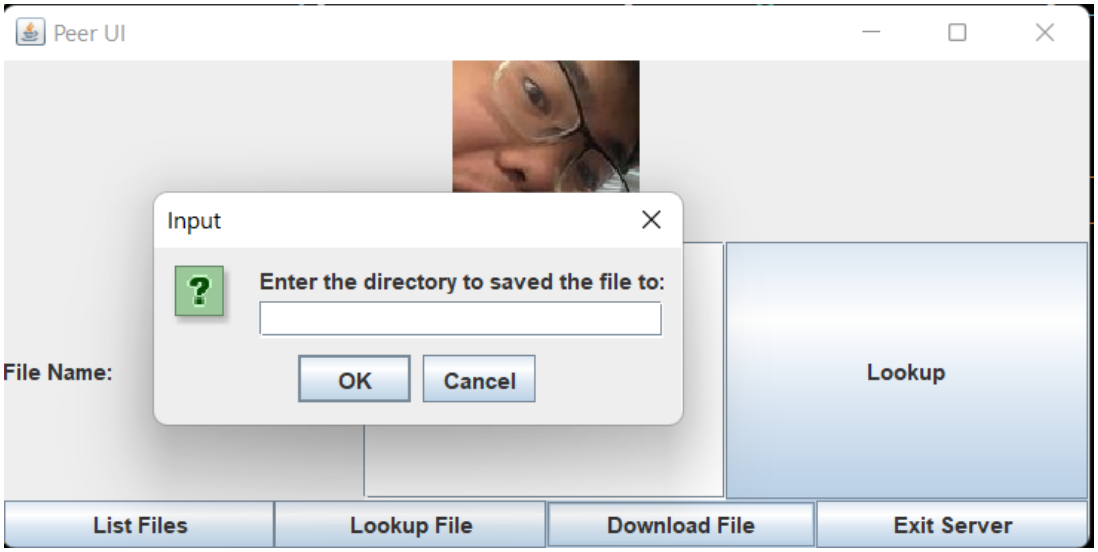
The next button is download button, when user has already found the file that they want to download, they will press the download button. If the file is currently exist in their shared folder, an announcement will be pop up and they can not download it anymore because it already exist: *Important: make sure you have already looked up for your file before download, this function cannot download non-existence file



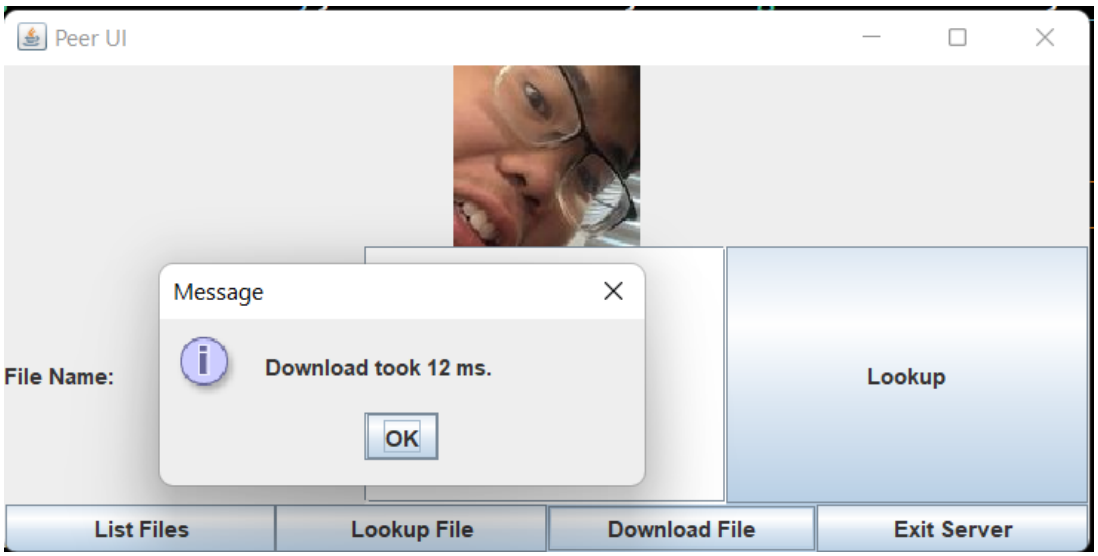
On the other hand, if that peer have not owned the file yet, a new windows pop up and tell user that the file will be downloaded from peer who keep it, after that user need to enter the folder directory they want to download it to their device and hit OK. Finally, when the file is downloaded successfully there will be an announcement appear to tell user that file has downloaded successfully and how much time took to download it. Here are sequence of images of stages we have mentioned:



Announcing that file will be downloaded from which peer.



Ask peer to provide the directory they want to download into. In this step, try to choose another directory with the one you share to the network to avoid conflicting with the check folder check function.



Tell user that their file has been transferred safe and sound

This is the code that we created to handle all the mentioned cases of download button:

```
1    downloadButton.addActionListener(new ActionListener() {
2    public void actionPerformed(ActionEvent e) {
3        String fileName = fileNameField.getText();
4        String message = "";
5        String inputDirectory = "";
6        if (peerAddress.length == 0) {
7            System.out.println("Lookup for the peer first.");
8        } else if (peerAddress.length == 1 && Integer.parseInt(↩
             peerAddress[0].split(":")[2]) == peer.getPeerId()) {
9            JOptionPane.showMessageDialog(null, "This peer has the ↩
                 file already, not downloading then.");
10       } else if (peerAddress.length == 1) {
11           String[] addrport = peerAddress[0].split(":");
12           JOptionPane.showMessageDialog(null, "Downloading from peer↩
                 " + addrport[2] + ": " + addrport[0] + ":" + addrport↩
                 [1]);
13           try {
14               inputDirectory = JOptionPane.showInputDialog("Enter ↩
                     the directory to saved the file to:");
15               if (inputDirectory == null || inputDirectory.trim().↩
                     isEmpty()) {
16                   JOptionPane.showMessageDialog(null, "No directory ↩
                         entered. Download cancelled.");
17                   return;
18               }
```

```
19                  message = peer.download(addrport[0], Integer.parseInt(↩
                        addrport[1]), fileName, -1, inputDirectory);
20                  JOptionPane.showMessageDialog(null, message);
21              } catch (IOException ex) {
22                  ex.printStackTrace();
23              }
24          } else {
25              // Display a dialog to select the peer to download from
26              String[] options = new String[peerAddress.length];
27              for (int i = 0; i < peerAddress.length; i++) {
28                  String[] addrport = peerAddress[i].split(":");
29                  options[i] = addrport[0] + ":" + addrport[1];
30              }
31              String selectedPeer = (String) JOptionPane.showInputDialog↩
                    (frame, "Select the peer to download from:", "Download ↩
                    File",
32                      JOptionPane.QUESTION_MESSAGE, null, options, ↩
                            options[0]);
33              if (selectedPeer != null) {
34                  String[] addrport = selectedPeer.split(":");
35                  try {
36                      inputDirectory = JOptionPane.showInputDialog("↩
                            Enter the directory to saved the file to:");
37                      if (inputDirectory == null || inputDirectory.trim↩
                            ().isEmpty()) {
38                          JOptionPane.showMessageDialog(null, "No ↩
                                directory entered. Download cancelled.");
39                          return;
40                      }
41                      message = peer.download(addrport[0], Integer.↩
                            parseInt(addrport[1]), fileName, -1, ↩
                            inputDirectory);
42                      JOptionPane.showMessageDialog(null, message);
43                  } catch (IOException ex) {
44                      ex.printStackTrace();
45                  }
46              }
47          }
48
49      }
50  });
```

4 Conclusion

4.1 Summary

P2P file-sharing systems are not a new topic, but this is the first time team members have thoroughly researched and developed a system like this. It caused quite a few difficulties for us. We are grateful for the assigned project; it has provided us with many valuable experiences about networking technology. In particular, the project helps us improve our programming and debugging abilities.

4.2 Task Division

- Pham Phi Long is responsible for laying down the foundation of the system, which includes setting up a centralized server, peer discovery, file discovery, and file download mechanisms.
- Nguyen Truc Linh and Thai Minh Kien focused on developing additional functionalities based on the established foundation. Some of the features includes listing files, choose peer to download, peer disconnect and more. They also responsibled for the fault tolerance problem in the system.
- Dang Hoang Anh Khoi handled the front-end, creating an intuitive and engaging user interface for the P2P file sharing system. He ensures that users can easily navigate, search, and download files or folders from peers. He also upgraded and optimized the foundation of the system.
- Dao The Hien is in charge of writing reports, scalability tests, and proposing new functionalities for the system. He ensured that all features are thoroughly tested and documented.