Peer-To-Peer File-Transfer Protocol

(Using Centralized Directory Architecture)

AIM:

To create a Peer-To-Peer File-Transfer-Protocol using Centralized Directory Architecture.

PROJECT DESCRIPTION:

Peer-to-Peer(P2P) Technologies are being widely used for sharing the data between the servers and the clients. One of the major technology for file sharing that is implemented nowadays is the Peer-to-Peer File Sharing System using centralized directory architecture which is also known as Napster style Peer-To-Peer File-Transfer-Protocol. The older versions of the systems used to have a single server which stores the files in its directory that are received from the clients. The major drawback of these systems was that if a new file has been created in one of the peers, it must be transferred to the server before another peer can access it, which delays the process of transfer from one peer to another. This can be conquered using the Napster system which allows the peer to peer file transfer.

SYSTEM ARCHITECTURE:

- **Processor Type:** Intel(R) Core(TM) i7-9750H
- **CPU Frequency:** 2.60GHz
- **Memory:** 16GB

SYSTEM REQUIREMENTS:

- JDK and Java to be installed
- Server to execute the program on multiple systems

DESIGN:

Entire project is designed using Java where we have used the concepts of Socket Programming and Multi-threading. For establishing the connections between the Server and the Clients, we have used TCP/IP protocol using the sockets.

Major Components of the Project:

- Server and
- Client

Server (Central Index Server):

This server indexes the content of all the peers (i.e., Clients) that register with it. It also provides search facility to peers.

Server Functionalities:

- Registry

Client:

As a client, the user specifies a file name with the indexing server using "lookup". The indexing server returns a list of all other peers that hold the file. The user can pick one such peer and the client then connects to this peer and downloads the file.

Major function of the peer:

- Download

As a server, the peer waits for requests from other peers and sends the requested file when receiving a request. The Peers (i.e., Clients) here, act as both the client and the server. This server is different from the central index server which only indexes the files. But, the server functionality of the peer can be used to download the files from its directory. The peer acts a client to download the files from other peers into its directory.

The peers provide the following interface to the users:

- Register – registers the file into the server
- Download – downloads the file from another Client

TRADEOFFS:

Instead of using the Array List for indexing, we can make using of the Data Structures. Even though Array List works fine in our case, but in case of randomly searching the files, hashing techniques serves well.

ABOUT TECHNOLOGY:

A peer-to-peer network allows computer hardware and software to communicate without the need for a server. Unlike client-server architecture, there is no central server for processing requests in a P2P architecture. The peers directly interact with one another without the requirement of a central server.

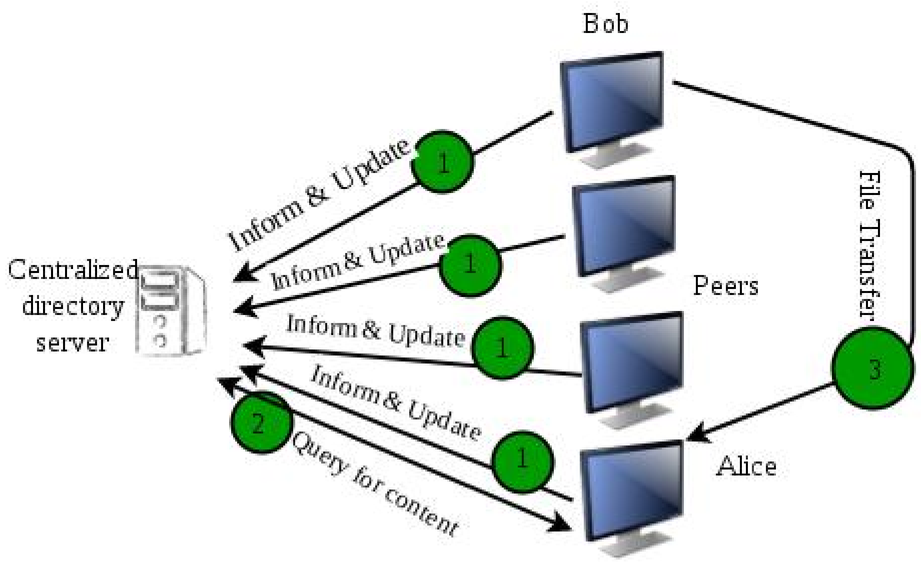
Now, when one peer makes a request, it is possible that multiple peers have a copy of that requested object. Now the problem is how to get the IP addresses of all those peers. This is decided by the underlying architecture supported by the P2P systems. By means of one of these methods, the client peer can get to know about all the peers which have the requested object/file and the file transfer takes place directly between these two peers.

CENTRALIZED DIRECTORY ARCHITECTURE:

- It is somewhat similar to client-server architecture in the sense that it maintains a huge central server to provide directory service.
- All the peers inform this central server of their IP address and the files they are making available for sharing.
- The server queries the peers at regular intervals to make sure if the peers are still connected or not.
- So basically this server maintains a huge database regarding which file is present at which IP addresses.

Working:

- Now whenever a requesting peer comes in, it sends its query to the server.
- Since the server has all the information of its peers, so it returns the IP addresses of all the peers having the requested file to the peer.
- Now the file transfer takes place between these two peers.
- The first system which made use of this method was Napster, for the purpose of Mp3 distribution.



P2P paradigm with a centralised directory

The major problem with such an architecture is that there is a single point of failure. If the server crashes, the whole P2P network crashes. Also, since all of the processing is to be done by a single server so a huge amount of the database has to be maintained and regularly updated.

CODE:

Server.java

```
import java.io.*;
import java.net.*;
import java.net.Socket;
import java.net.ServerSocket;
import java.util.Scanner;
import java.lang.Runnable;
import java.util.ArrayList;
import java.util.Collections;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.lang.Integer;

@SuppressWarnings("unused")

public class Server {
	public static ArrayList<FileInfo> globalArray = new ArrayList<FileInfo>();

	@SuppressWarnings("resource")
	// public static void main(String args[])
	public Server() throws NumberFormatException, IOException {

		ServerSocket serverSocket = null;
		Socket socket = null;
		try {
			System.out.println("*** Welcome to the Server ***");
			System.out.println(" ");
			serverSocket = new ServerSocket(7799);
			System.out.println("Server started!! ");
			System.out.println(" ");
```

```
				System.out.println("Waiting for the Client to be connected ..");
			} catch (IOException e) {
				e.printStackTrace();
			}
			while (true) {
				try {
					socket = serverSocket.accept();
					// serverSocket.close();
				} catch (IOException e) {
					System.out.println("I/O error: " + e);
				}
				new ServerTestClass(socket, globalArray).start();
			}
		}
}

class ServerTestClass extends Thread {
		protected Socket socket;
		ArrayList<FileInfo> globalArray;

		public ServerTestClass(Socket clientSocket, ArrayList<FileInfo> globalArray) {
			this.socket = clientSocket;
			this.globalArray = globalArray;
		}

		ArrayList<FileInfo> filesList = new ArrayList<FileInfo>();
		ObjectOutputStream oos;
		ObjectInputStream ois;
		String str;
		int index;

		@SuppressWarnings("unchecked")
		public void run() {
			try {
				InputStream is = socket.getInputStream();
				oos = new ObjectOutputStream(socket.getOutputStream());
				ois = new ObjectInputStream(is);
				filesList = (ArrayList<FileInfo>) ois.readObject();
				System.out.println("All the available files from the given directory have
been recieved to the Server!");
				for (int i = 0; i < filesList.size(); i++) {
```

```
                    globalArray.add(filesList.get(i));
                }
                System.out.println(
                        "Total number of files available in the Server that are
received from all the connected clients: "
                                + globalArray.size());
        }

        catch (IndexOutOfBoundsException e) {
                System.out.println("Index out of bounds exception");
        } catch (IOException e) {
                System.out.println("I/O exception");
        } catch (ClassNotFoundException e) {
                System.out.println("Class not found exception");
        }

        try {
                str = (String) ois.readObject();
        } catch (IOException | ClassNotFoundException ex) {
                Logger.getLogger(ServerTestClass.class.getName()).log(Level.SEVERE,
null, ex);
        }

        ArrayList<FileInfo> sendingPeers = new ArrayList<FileInfo>();
        System.out.println("Searching for the file name...!!!");

        for (int j = 0; j < globalArray.size(); j++) {
                FileInfo fileInfo = globalArray.get(j);
                Boolean tf = fileInfo.fileName.equals(str);
                if (tf) {
                        index = j;
                        sendingPeers.add(fileInfo);
                }
        }

        try {
                oos.writeObject(sendingPeers);
        } catch (IOException ex) {
                Logger.getLogger(ServerTestClass.class.getName()).log(Level.SEVERE,
null, ex);
        }
        }
}
```

Client.java

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Client {
        @SuppressWarnings({ "unchecked", "rawtypes", "resource", "unused" })
        // public static void main(String args[]) throws Exception
        public Client() {
                Socket socket;
                ArrayList al;
                ArrayList<FileInfo> arrList = new ArrayList<FileInfo>();
                Scanner scanner = new Scanner(System.in);
                ObjectInputStream ois;
                ObjectOutputStream oos;
                String string;
                Object o, b;
                String directoryPath = null;
                int peerServerPort = 0;

                try {
                        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));

                        System.out.println("*** Welcome to the Client ***");
                        System.out.println(" ");

                        System.out.print("Enter the peerid for this directory : ");
                        int readpid = Integer.parseInt(br.readLine());

                        System.out.print("Enter the port number on which the peer should act as
server : ");
                        peerServerPort = Integer.parseInt(br.readLine());
                        System.out.print("Enter the directory that contain the files : ");
                        directoryPath = br.readLine();

                        ServerDownload objServerDownload = new
ServerDownload(peerServerPort, directoryPath);
                        objServerDownload.start();

                        Socket clientThread = new Socket("localhost", 7799);

                        ObjectOutputStream objOutStream = new
ObjectOutputStream(clientThread.getOutputStream());
                        ObjectInputStream objInStream = new
ObjectInputStream(clientThread.getInputStream());

                        al = new ArrayList();

                        socket = new Socket("localhost", 7799);
                        System.out.println("Connection has been established with the client");

                        ois = new ObjectInputStream(socket.getInputStream());
                        oos = new ObjectOutputStream(socket.getOutputStream());

                        File folder = new File(directoryPath);
                        File[] listofFiles = folder.listFiles();
                        FileInfo currentFile;
                        File file;

                        for (int i = 0; i < listofFiles.length; i++) {
                                currentFile = new FileInfo();
                                file = listofFiles[i];
                                currentFile.fileName = file.getName();
                                currentFile.peerid = readpid;
                                currentFile.portNumber = peerServerPort;
                                arrList.add(currentFile);
                        }

                        oos.writeObject(arrList);
                        // System.out.println("The complete ArrayList :::"+arrList);
                        System.out.print(
                                        "Enter the desired file name that you want to downloaded
from the list of the files available in the Server : ");
```

```
			String fileNameToDownload = br.readLine();
			oos.writeObject(fileNameToDownload);

			System.out.println("Waiting for the reply from Server...!!");

			ArrayList<FileInfo> peers = new ArrayList<FileInfo>();
			peers = (ArrayList<FileInfo>) ois.readObject();
			if (peers.size() == 0) {
				System.out.println("Requested file not found in any of the
peers !!!");
			} else {
				for (int i = 0; i < peers.size(); i++) {
					int result = peers.get(i).peerid;
					int port = peers.get(i).portNumber;
					System.out.println("The file is stored at peer id " + result + "
on port " + port);
				}
				int clientAsServerPortNumber = 0, clientAsServerPeerid = 0;
				int chk = 1;
				while (chk != 0) {
					System.out.println("Enter the desired peer id from which
you want to download the file from :");
					clientAsServerPeerid = Integer.parseInt(br.readLine());

					for (int i = 0; i < peers.size(); i++) {
						int result = peers.get(i).peerid;
						int port = peers.get(i).portNumber;
						if (result == clientAsServerPeerid) {
							clientAsServerPortNumber = port;
							chk = 0;
							break;
						}
					}
					if (chk == 1) {
						System.out.println("Entered PeerID Not
Found.\nPlease Try Again...");
					}
				}
				clientAsServer(clientAsServerPeerid, clientAsServerPortNumber,
fileNameToDownload, directoryPath);
			}
```

```
		} catch (Exception e) {
			System.out.println("Error in establishing the Connection between the
Client and the Server!! ");
			System.out.println("Please cross-check the host address and the port
number..");
		}
	}

	public static void clientAsServer(int clientAsServerPeerid, int
clientAsServerPortNumber, String fileNamedwld,
			String directoryPath) throws ClassNotFoundException {
		try {
			@SuppressWarnings("resource")
			Socket clientAsServersocket = new Socket("localhost",
clientAsServerPortNumber);

			ObjectOutputStream clientAsServerOOS = new
ObjectOutputStream(clientAsServersocket.getOutputStream());
			ObjectInputStream clientAsServerOIS = new
ObjectInputStream(clientAsServersocket.getInputStream());

			clientAsServerOOS.writeObject(fileNamedwld);
			int readBytes = (int) clientAsServerOIS.readObject();

			// System.out.println("Number of bytes that have been transferred are
			// ::"+readBytes);

			byte[] b = new byte[readBytes];
			clientAsServerOIS.readFully(b);
			OutputStream fileOPstream = new FileOutputStream(directoryPath + "//"
+ fileNamedwld);

			@SuppressWarnings("resource")

			BufferedOutputStream BOS = new BufferedOutputStream(fileOPstream);
			BOS.write(b, 0, (int) readBytes);

			System.out.println("Requested file - " + fileNamedwld + ", has been
downloaded to your desired directory "
					+ directoryPath);
```

```
                    System.out.println(" ");
                    System.out.println("Display file " + fileNamedwld);

                    BOS.flush();
            } catch (IOException ex) {
                    Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
}
```

FileInfo.java

```
import java.io.Serializable;

@SuppressWarnings("serial")

public class FileInfo implements Serializable
{
        public int peerid;
        public String fileName;
        public int portNumber;
}
```

ServerDownload.java

```
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ServerDownload extends Thread
{
   int peerServerPort;
   String  directoryPath=null;
   ServerSocket dwldServerSocket;
   Socket dwldSocket=null;
   /*
   public ServerDownload()
   {

   }
   */
   ServerDownload(int peerServerPort,String directoryPath) {
        this.peerServerPort=peerServerPort;
        this.directoryPath=directoryPath;
   }
   public void run(){
        try {
                      dwldServerSocket = new ServerSocket(peerServerPort);
                      dwldSocket = dwldServerSocket.accept();
                      new ServerDownloadThread(dwldSocket,directoryPath).start();
       }
        catch (IOException ex) {
                      Logger.getLogger(ServerDownload.class.getName()).log(Level.SEVERE,
null, ex);
        }
   }
}
class ServerDownloadThread extends Thread
{
   Socket dwldThreadSocket;
   String directoryPath;
   public ServerDownloadThread(Socket dwldThreadSocket,String directoryPath)
   {
      this.dwldThreadSocket=dwldThreadSocket;
      this.directoryPath=directoryPath;
   }
   @SuppressWarnings({ "unused", "resource" })
        public void run()
   {
      try
      {
         ObjectOutputStream objOS = new
ObjectOutputStream(dwldThreadSocket.getOutputStream());
```

```
            ObjectInputStream objIS = new
ObjectInputStream(dwldThreadSocket.getInputStream());

            String fileName = (String)objIS.readObject();
            String fileLocation;// Stores the directory name
            while(true)
            {
              File myFile = new File(directoryPath+"//"+fileName);
              long length = myFile.length();

              byte [] byte_arr = new byte[(int)length];

              objOS.writeObject((int)myFile.length());
              objOS.flush();

              FileInputStream FIS=new FileInputStream(myFile);
              BufferedInputStream objBIS = new BufferedInputStream(FIS);
              objBIS.read(byte_arr,0,(int)myFile.length());

              //System.out.println("Sending the file of " +byte_arr.length+ " bytes");

              objOS.write(byte_arr,0,byte_arr.length);

              objOS.flush();
            }
        }
        catch(Exception e)
        {

        }
    }
}
```

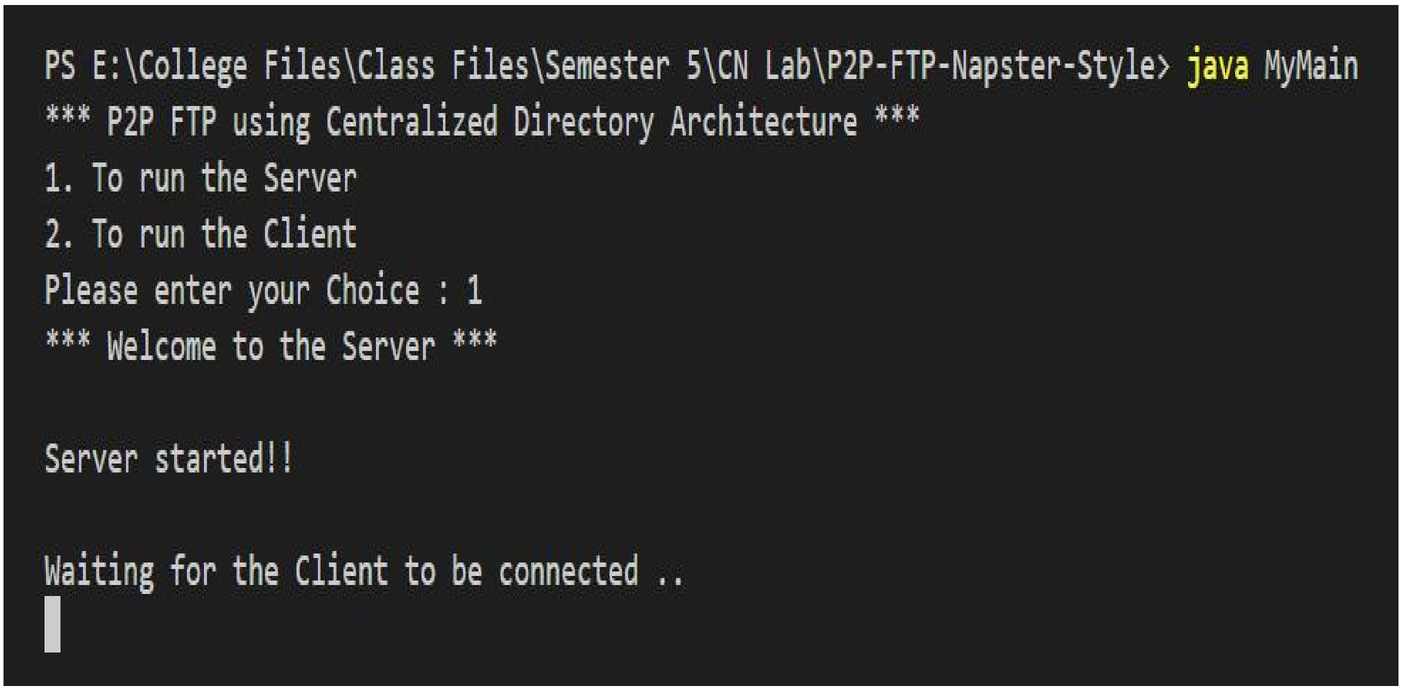
MyMain.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class MyMain {
          public static void main(String args[]) throws Exception {
```

```
		System.out.println("*** P2P FTP using Centralized Directory Architecture
***");
		System.out.println("1. To run the Server");
		System.out.println("2. To run the Client");
		System.out.print("Please enter your Choice : ");
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		int choice = Integer.parseInt(br.readLine());

		if (choice == 1) {
			Server s = new Server();
		} else if (choice == 2) {
			Client c = new Client();
		} else {
			System.out.println("Your choice is incorrect");
		}
	}
}
```

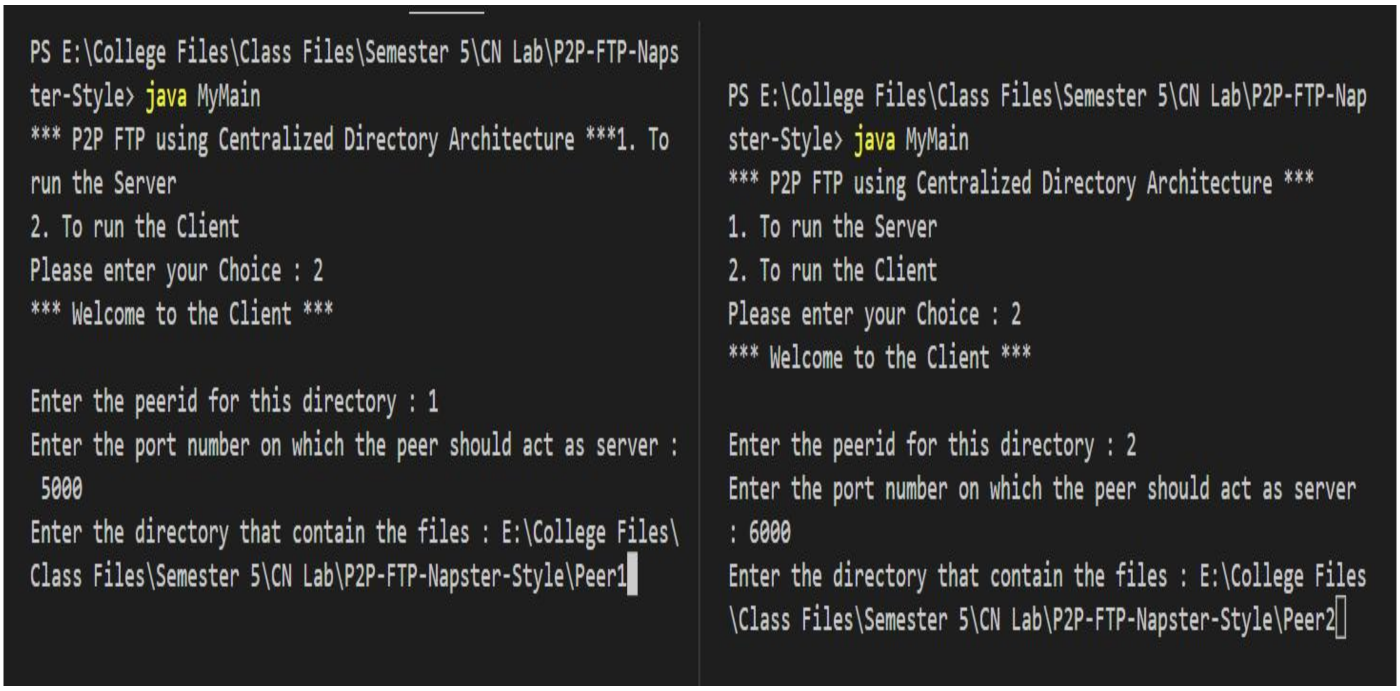
OUTPUT SCREENSHOTS:

Start of Server



```
PS E:\College Files\Class Files\Semester 5\CN Lab\P2P-FTP-Napster-Style> java MyMain
*** P2P FTP using Centralized Directory Architecture ***
1. To run the Server
2. To run the Client
Please enter your Choice : 1
*** Welcome to the Server ***

Server started!!

Waiting for the Client to be connected ..
```

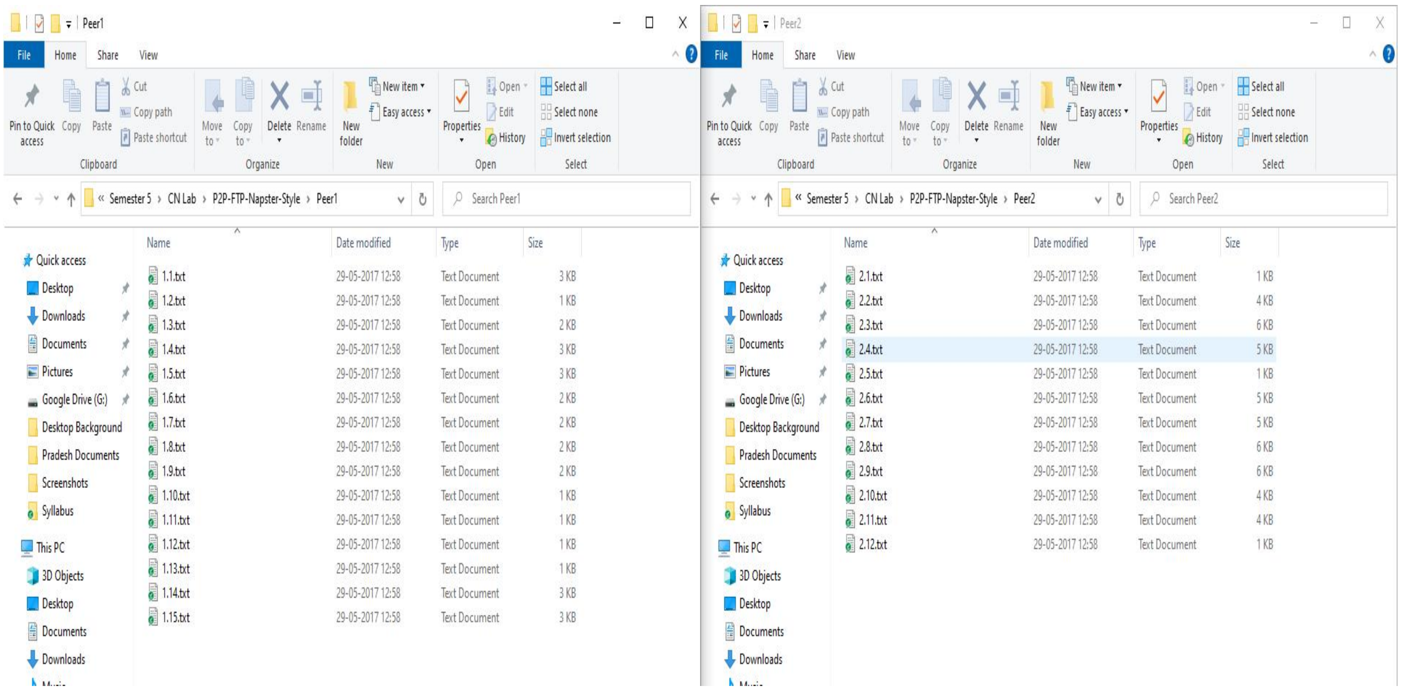
Start of Client



```
PS E:\College Files\Class Files\Semester 5\CN Lab\P2P-FTP-Naps
ter-Style> java MyMain
*** P2P FTP using Centralized Directory Architecture ***1. To
run the Server
2. To run the Client
Please enter your Choice : 2
*** Welcome to the Client ***

Enter the peerid for this directory : 1
Enter the port number on which the peer should act as server :
 5000
Enter the directory that contain the files : E:\College Files\
Class Files\Semester 5\CN Lab\P2P-FTP-Napster-Style\Peer1
```

```
PS E:\College Files\Class Files\Semester 5\CN Lab\P2P-FTP-Nap
ster-Style> java MyMain
*** P2P FTP using Centralized Directory Architecture ***
1. To run the Server
2. To run the Client
Please enter your Choice : 2
*** Welcome to the Client ***

Enter the peerid for this directory : 2
Enter the port number on which the peer should act as server
: 6000
Enter the directory that contain the files : E:\College Files
\Class Files\Semester 5\CN Lab\P2P-FTP-Napster-Style\Peer2
```

Directory Contents before P2P File Transfer



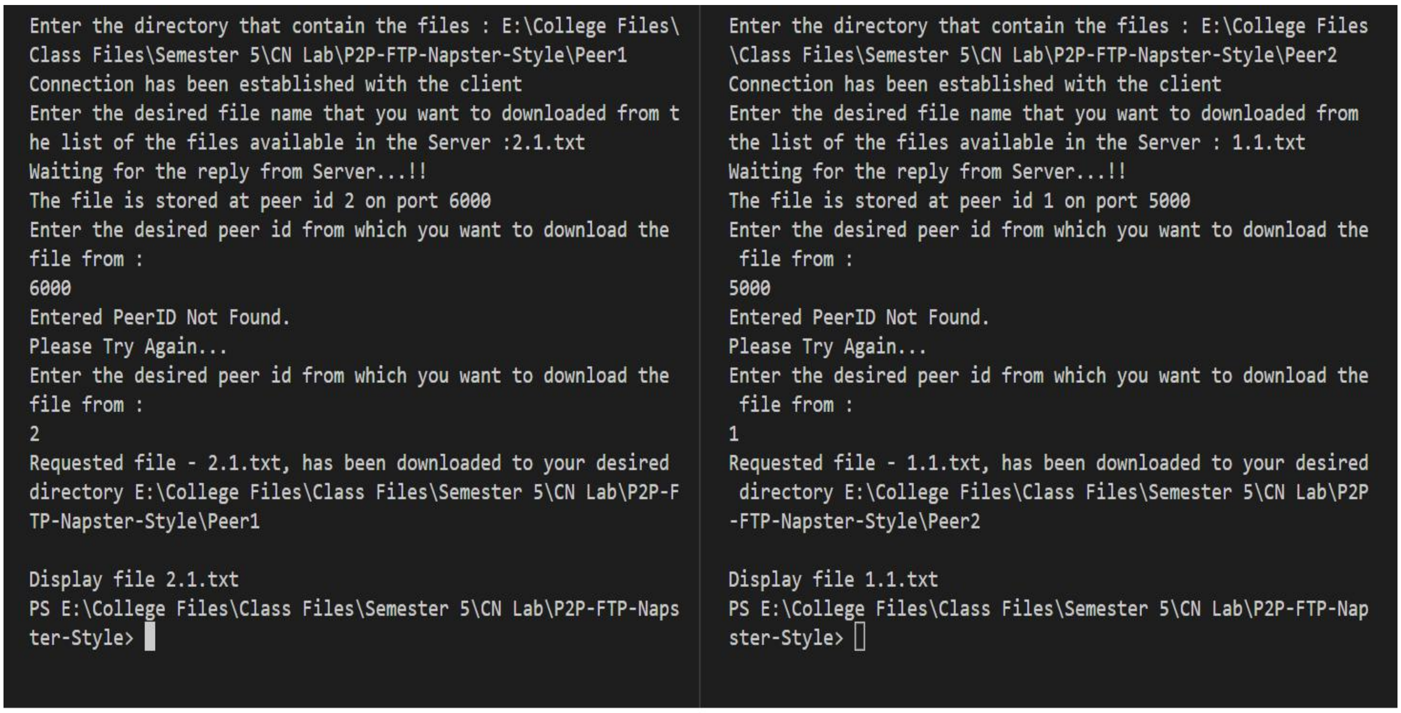
« Semester 5 › CN Lab › P2P-FTP-Napster-Style › Peer1

Name	Date modified	Type	Size
1.1.txt	29-05-2017 12:58	Text Document	3 KB
1.2.txt	29-05-2017 12:58	Text Document	1 KB
1.3.txt	29-05-2017 12:58	Text Document	2 KB
1.4.txt	29-05-2017 12:58	Text Document	3 KB
1.5.txt	29-05-2017 12:58	Text Document	3 KB
1.6.txt	29-05-2017 12:58	Text Document	2 KB
1.7.txt	29-05-2017 12:58	Text Document	2 KB
1.8.txt	29-05-2017 12:58	Text Document	2 KB
1.9.txt	29-05-2017 12:58	Text Document	2 KB
1.10.txt	29-05-2017 12:58	Text Document	1 KB
1.11.txt	29-05-2017 12:58	Text Document	1 KB
1.12.txt	29-05-2017 12:58	Text Document	1 KB
1.13.txt	29-05-2017 12:58	Text Document	1 KB
1.14.txt	29-05-2017 12:58	Text Document	3 KB
1.15.txt	29-05-2017 12:58	Text Document	3 KB

« Semester 5 › CN Lab › P2P-FTP-Napster-Style › Peer2

Name	Date modified	Type	Size
2.1.txt	29-05-2017 12:58	Text Document	1 KB
2.2.txt	29-05-2017 12:58	Text Document	4 KB
2.3.txt	29-05-2017 12:58	Text Document	6 KB
2.4.txt	29-05-2017 12:58	Text Document	5 KB
2.5.txt	29-05-2017 12:58	Text Document	1 KB
2.6.txt	29-05-2017 12:58	Text Document	5 KB
2.7.txt	29-05-2017 12:58	Text Document	5 KB
2.8.txt	29-05-2017 12:58	Text Document	6 KB
2.9.txt	29-05-2017 12:58	Text Document	6 KB
2.10.txt	29-05-2017 12:58	Text Document	4 KB
2.11.txt	29-05-2017 12:58	Text Document	4 KB
2.12.txt	29-05-2017 12:58	Text Document	1 KB

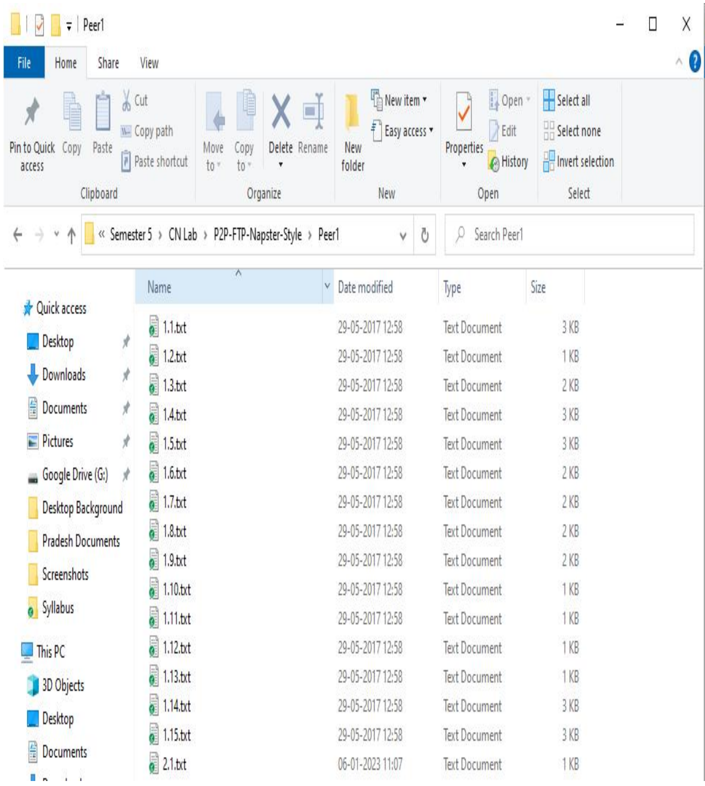
P2P File Transfer Input



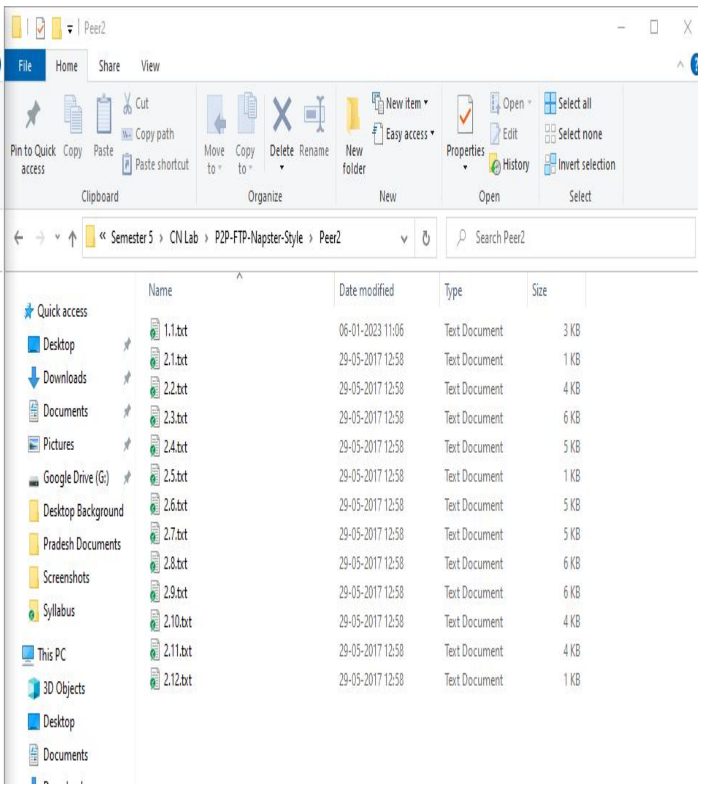
```
Enter the directory that contain the files : E:\College Files\
Class Files\Semester 5\CN Lab\P2P-FTP-Napster-Style\Peer1
Connection has been established with the client
Enter the desired file name that you want to downloaded from t
he list of the files available in the Server :2.1.txt
Waiting for the reply from Server...!!
The file is stored at peer id 2 on port 6000
Enter the desired peer id from which you want to download the
file from :
6000
Entered PeerID Not Found.
Please Try Again...
Enter the desired peer id from which you want to download the
file from :
2
Requested file - 2.1.txt, has been downloaded to your desired
directory E:\College Files\Class Files\Semester 5\CN Lab\P2P-F
TP-Napster-Style\Peer1

Display file 2.1.txt
PS E:\College Files\Class Files\Semester 5\CN Lab\P2P-FTP-Naps
ter-Style>
```

```
Enter the directory that contain the files : E:\College Files
\Class Files\Semester 5\CN Lab\P2P-FTP-Napster-Style\Peer2
Connection has been established with the client
Enter the desired file name that you want to downloaded from
the list of the files available in the Server : 1.1.txt
Waiting for the reply from Server...!!
The file is stored at peer id 1 on port 5000
Enter the desired peer id from which you want to download the
 file from :
5000
Entered PeerID Not Found.
Please Try Again...
Enter the desired peer id from which you want to download the
 file from :
1
Requested file - 1.1.txt, has been downloaded to your desired
 directory E:\College Files\Class Files\Semester 5\CN Lab\P2P
-FTP-Napster-Style\Peer2

Display file 1.1.txt
PS E:\College Files\Class Files\Semester 5\CN Lab\P2P-FTP-Nap
ster-Style>
```

Directories after P2P File Transfer



Name	Date modified	Type	Size
1.1.txt	29-05-2017 12:58	Text Document	3 KB
1.2.txt	29-05-2017 12:58	Text Document	1 KB
1.3.txt	29-05-2017 12:58	Text Document	2 KB
1.4.txt	29-05-2017 12:58	Text Document	3 KB
1.5.txt	29-05-2017 12:58	Text Document	3 KB
1.6.txt	29-05-2017 12:58	Text Document	2 KB
1.7.txt	29-05-2017 12:58	Text Document	2 KB
1.8.txt	29-05-2017 12:58	Text Document	2 KB
1.9.txt	29-05-2017 12:58	Text Document	2 KB
1.10.txt	29-05-2017 12:58	Text Document	1 KB
1.11.txt	29-05-2017 12:58	Text Document	1 KB
1.12.txt	29-05-2017 12:58	Text Document	1 KB
1.13.txt	29-05-2017 12:58	Text Document	1 KB
1.14.txt	29-05-2017 12:58	Text Document	3 KB
1.15.txt	29-05-2017 12:58	Text Document	3 KB
2.1.txt	06-01-2023 11:07	Text Document	1 KB



Name	Date modified	Type	Size
1.1.txt	06-01-2023 11:06	Text Document	3 KB
2.1.txt	29-05-2017 12:58	Text Document	1 KB
2.2.txt	29-05-2017 12:58	Text Document	4 KB
2.3.txt	29-05-2017 12:58	Text Document	6 KB
2.4.txt	29-05-2017 12:58	Text Document	5 KB
2.5.txt	29-05-2017 12:58	Text Document	1 KB
2.6.txt	29-05-2017 12:58	Text Document	5 KB
2.7.txt	29-05-2017 12:58	Text Document	5 KB
2.8.txt	29-05-2017 12:58	Text Document	6 KB
2.9.txt	29-05-2017 12:58	Text Document	6 KB
2.10.txt	29-05-2017 12:58	Text Document	4 KB
2.11.txt	29-05-2017 12:58	Text Document	4 KB
2.12.txt	29-05-2017 12:58	Text Document	1 KB

POSSIBLE IMPROVEMENTS:

- We can improve the performance using the Data Structures
- Could develop an User Interface.
- Port numbers can be eliminated.

RESULT:

Hence, we have successfully created and implemented Peer-To-Peer File-Tranfer-Protocol using Centralized Directory Architecture.