External Project Report on Computer Networking (CSE3034)

Distributed File Sharing System

<u>Submitted by</u>

Name : Sumit Dash	Reg. No.: 2141004083
Name : Chanchal Anand	Reg. No.: 2141001043
Name : Om Tanmaya Pati	Reg. No.: 2141016166
Name : Ujjwal Kumar	Reg. No.: 2141011063
Name : Ritik Kumar	Reg. No.: 2141011065

B. Tech. BRANCH 5th Semester (Section C)

INSTITUTE OF TECHNICAL EDUCATION AND RESEARCH
(FACULTY OF ENGINEERING)
SIKSHA 'O' ANUSANDHAN (DEEMED TO BE UNIVERSITY), BHUBANESWAR, ODISHA

Declaration

We, the undersigned students of B. Tech. of **Computer Science Engineering** Department hereby declare that we own the full responsibility for the information, results etc. provided in this PROJECT titled "Create a peer-to-peer (P2P) Distributed File Sharing System where users can share and download files from each other's machines directly without a central server. The system should support multiple concurrent file transfers." submitted to **Siksha 'O' Anusandhan (Deemed to be University), Bhubaneswar** for the partial fulfillment of the subject **Computer Networking (CSE 3034)**. We have taken care in all respect to honor the intellectual property right and have acknowledged the contribution of othersfor using them in academic purpose and further declare that in case of any violation of intellectual property right or copyright we, as the candidate(s), will be fully responsible for the same.

Sumit Dash

Registration No.: 214100408

Chanchal Anand

Registration No.: 2141001043

Om Tanmaya Pati

Registration No.: 2141016166

Ujjwal Kumar

Registration No.: 2141011063

Ritik Kumar

Registration No.: 2141011065

DATE: 10th January 2023

PLACE: Bhubaneshwar

Abstract

The provided Java code implements a basic Peer-to-Peer (P2P) file sharing system using sockets for communication. The system consists of two components: a sender and a receiver, each running on separate threads. The sender is responsible for transmitting multiple files to the receiver over a network connection.

The sender initiates a connection to the receiver's IP address and designated port, sending a series of files stored in a shared directory. For each file, the sender transmits the file name followed by its content in chunks. The receiver, in turn, accepts incoming connections on the specified port, reads the file name and content from the sender, and saves the received files into a designated downloads directory.

The code employs object serialization for efficient transmission of file names and contents between the sender and receiver. A special string ("exit") is used as a signal to mark the end of the file transmission process.

The program demonstrates the core functionality of a P2P file sharing system without any Data central Server, allowing files to be transferred between two peers over a network. It serves as a foundation for a more comprehensive P2P file-sharing application, with potential improvements in error handling, user interface, and scalability.

Contents

1. Introduction

In the realm of data exchange, **Peer-to-Peer (P2P)** file sharing stands as a transformative force, reshaping the landscape of how users across the globe connect and share files directly. This code implementation of a **P2PFileSharingSystem** encapsulates the essence of this revolution by facilitating a decentralized approach to file sharing. Unlike traditional methods that rely on a centralized server, P2P file sharing empowers users to establish **direct connections**, fostering a seamless exchange of files.

As the P2PFileSharingSystem code unfolds, it becomes evident that it simulates the interaction between a sender and a receiver through separate threads. The **sender dynamically transmits an array of files** to the receiver, showcasing the practical implementation of P2P communication. The code operates over sockets, employing object serialization to efficiently send both file names and their corresponding contents. Furthermore, the system incorporates error handling and a graceful exit mechanism, enhancing its robustness.

In sync with the revolutionary nature of P2P file sharing, the code reflects the essence of **decentralization**. The sender and receiver threads epitomize the flexibility and directness inherent in P2P communication. This introduction, combined with the provided code, sets the stage for a deeper exploration of the P2PFileSharingSystem, offering insights into its functionality and paving the way for potential enhancements and real-world applications.

.

2.Problem Statement

i. **Explanation of the Problem :**
The problem at hand involves the development of a Peer-to-Peer (P2P) file sharing system. Users, acting as both senders and receivers, simulate the sharing of files between their computers. The objective is to create a console-based application where users can input file names or content, and these inputs are transmitted between simulated sender and receiver components over a network connection.

In the current implementation, the sender reads files from a shared directory and transmits them to the receiver. The receiver saves the received files into a specified downloads directory. The program utilizes sockets and object serialization to facilitate communication and file transfer. The user interaction is simulated by predefined files in the shared directory.

ii. **Identification of User Input and Result Reflection**
   a. User Input: The simulated user input in this scenario is the predefined array of file names (filesToSend). Users, in a real-world application, would input the file names or content they wish to share.
   b. Result Reflection: The results are reflected in the console through status messages. For the sender, successful transmission messages are displayed, and for the receiver, successful reception and saving of files are reported**.**

iii. **Constraints:**
   a. Network Connectivity: The success of the file-sharing system depends on stable network connectivity between the sender and receiver.
   b. Predefined Files: The current implementation relies on predefined files in the shared directory. In a real-world scenario, there would be a need for user input to specify the files for sharing.
   c. Simulated Environment: The sender and receiver IP

addresses are currently hardcoded for simulation. In an actual use case, these would need to be dynamic and based on user input or discovery mechanisms.

d. Exception Handling: The code currently provides basic exception handling. Enhancements may be needed for more robust error management, especially in a production environment.

e. Security Concerns: The current implementation lacks security measures. In a real-world scenario, secure communication protocols and file encryption would be essential to protect data during transmission.

3. Methodology

```
// Main Function
function main():
   senderIpAddress = "127.0.0.1"
   receiverIpAddress = "127.0.0.1"

   senderThread = Thread(startSender(senderIpAddress))
   receiverThread = Thread(startReceiver(receiverIpAddress))

   senderThread.start()
   receiverThread.start()

// Sender Function
function startSender(receiverIpAddress):
   try:
      socket = Socket(receiverIpAddress, PORT)
      objectOutputStream = ObjectOutputStream(socket.getOutputStream())

      filesToSend = {"file1.txt", "file2.txt", "file3.txt", "pexels-ihsan-adityawarman-19133309.jpg", "404.pdf"}

      for fileName in filesToSend:
         objectOutputStream.writeObject(fileName)

         fileInputStream = FileInputStream(SHARED_DIRECTORY + fileName)
         buffer = byte[4096]
         bytesRead = 0

         while (bytesRead = fileInputStream.read(buffer)) != -1:
            objectOutputStream.write(buffer, 0, bytesRead)

         print("File '" + fileName + "' sent successfully.")

      objectOutputStream.writeObject("exit")

   catch IOException:
      printStackTrace()

// Receiver Function
function startReceiver(senderIpAddress):
   try:
      serverSocket = ServerSocket(PORT)
      print("Waiting for sender to connect...")

      senderSocket = serverSocket.accept()

      downloadsDirectory = File(DOWNLOADS_DIRECTORY)
      downloadsDirectory.mkdirs()
```

```
    objectInputStream = ObjectInputStream(senderSocket.getInputStream())

    while true:
        try:
            fileName = objectInputStream.readObject()
        catch EOFException:
            break

        if "exit".equalsIgnoreCase(fileName):
            break

        fileOutputStream = FileOutputStream(DOWNLOADS_DIRECTORY + fileName)
        buffer = byte[4096]
        bytesRead = 0

        while (bytesRead = objectInputStream.read(buffer)) != -1:
            fileOutputStream.write(buffer, 0, bytesRead)

        print("File '" + fileName + "' received successfully and saved in Downloads directory.")

catch (ClassNotFoundException | IOException):
    printStackTrace()
```

P2P File Sharing System

1. Objective Definition:
- Develop a Peer-to-Peer (P2P) Distributed File Sharing System in Java.
- Enable users to share and download files directly without a central server.
- Implement support for multiple concurrent file transfers.

2. System Architecture:
- Utilize a client-server model with sender and receiver roles.
- Communication established through sockets.

3. Directory Structure:
- Define shared and downloads directories (SHARED_DIRECTORY and DOWNLOADS_DIRECTORY).
- Ensure existence of the downloads directory.

4. Peer Simulation:
- Simulate sender and receiver functionality on separate threads.
- Utilize local IP addresses for sender and receiver (127.0.0.1).

5. *Sender Implementation:
- Establish a socket connection with the receiver's IP address and specified port.
- Serialize and send file names and contents to the receiver.
- Signal the end of downloads using a special string ("exit").

6. Receiver Implementation:
- Create a server socket to wait for the sender to connect.
- Accept the sender's connection and create necessary directories.
- Deserialize and receive file names and contents.
- Save received files in the downloads directory.

7. File Transfer Mechanism:
- Employ ObjectOutputStream and ObjectInputStream for efficient file serialization and deserialization.
- Utilize byte buffers for reading and writing file content.

8. Exception Handling:
- Implement comprehensive error handling to manage IOExceptions, EOFExceptions, and ClassNotFoundExceptions.
- Print stack traces for debugging purposes.

9. Concurrency Management:
- Leverage Java threads to enable concurrent uploads and downloads.
- Separate sender and receiver functionalities into distinct threads.

10. User Feedback:
- Display informative messages for successful file transfers.
- Alert users about the completion of the process.

11. Testing and Debugging:
- Rigorous testing to ensure seamless communication and file transfer.
- Debugging to address any unexpected issues and refine the system's robustness.

12. Documentation:
- Maintain clear and concise comments within the code for future reference.
- Document any specific considerations or limitations of the implemented system.

13. Presentation:
- Compile key aspects into a PowerPoint presentation.
- Include visuals and code snippets to illustrate the system's functionality.
- Highlight the use of Java IO and threads for an effective distributed file sharing solution.

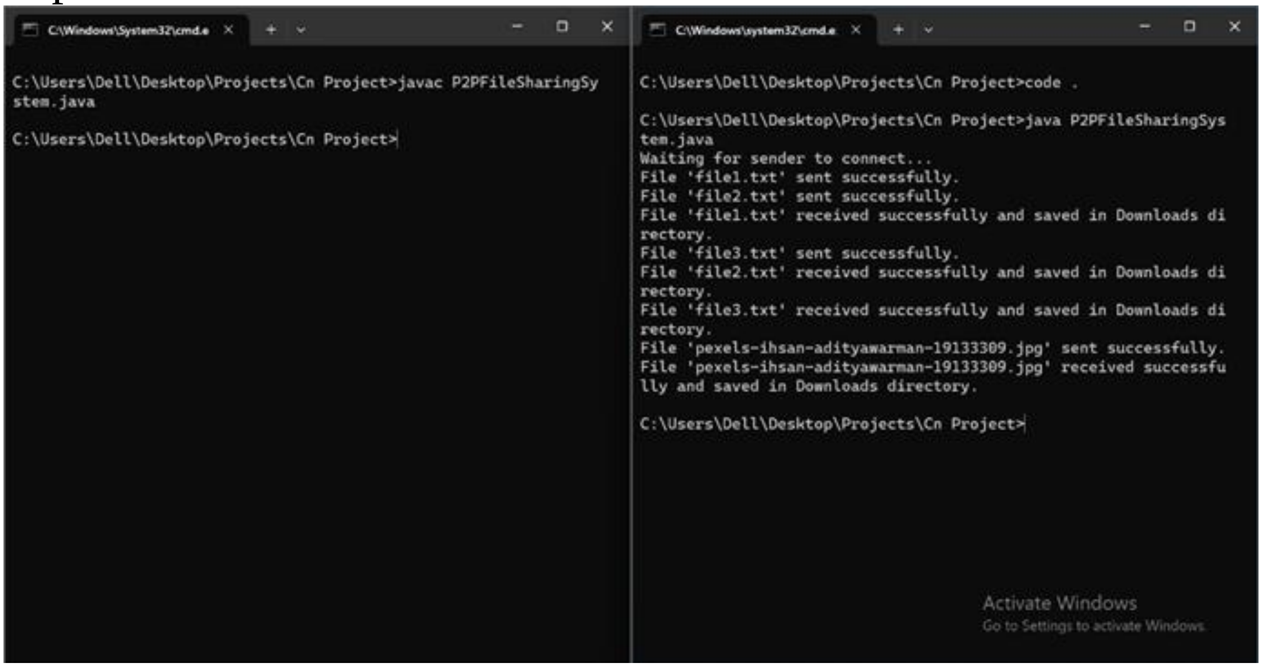
4. Implementation

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;

public class P2PFileSharingSystem {

    private static final int PORT = 12345;
    private static final String SHARED_DIRECTORY = "shared-files/";
    private static final String DOWNLOADS_DIRECTORY = "Downloads/";

    public static void main(String[] args) {
        // Replace these IP addresses with the actual IP addresses of the peers
        String senderIpAddress = "127.0.0.1";
        String receiverIpAddress = "127.0.0.1";

        // Simulate sender and receiver
        Thread senderThread = new Thread(() -> startSender(senderIpAddress));
        Thread receiverThread = new Thread(() -> startReceiver(receiverIpAddress));

        senderThread.start();
        receiverThread.start();
    }

    private static void startSender(String receiverIpAddress) {
        try (Socket socket = new Socket(receiverIpAddress, PORT);
             ObjectOutputStream objectOutputStream = new
ObjectOutputStream(socket.getOutputStream())) {

            // Files to send
            String[] filesToSend = {"file1.txt", "file2.txt", "file3.txt","pexels-ihsan-
adityawarman-19133309.jpg","404.pdf"};

            for (String fileName : filesToSend) {

                objectOutputStream.writeObject(fileName);

                // Send the file content
                try (FileInputStream fileInputStream = new
FileInputStream(SHARED_DIRECTORY + fileName)) {
```

```
                byte[] buffer = new byte[4096];
                int bytesRead;
                while ((bytesRead = fileInputStream.read(buffer)) != -1) {
                    objectOutputStream.write(buffer, 0, bytesRead);
                }
            }

            System.out.println("File '" + fileName + "' sent successfully.");
        }

        // Send a special string to signal the end of downloads
        objectOutputStream.writeObject("exit");

    } catch (IOException e) {
        e.printStackTrace();
    }
}

private static void startReceiver(String senderIpAddress) {
    try (ServerSocket serverSocket = new ServerSocket(PORT)) {
        System.out.println("Waiting for sender to connect...");

        // Wait for the sender to connect
        Socket senderSocket = serverSocket.accept();

        File downloadsDirectory = new File(DOWNLOADS_DIRECTORY);
        downloadsDirectory.mkdirs(); // Create Downloads directory

        try (ObjectInputStream objectInputStream = new
ObjectInputStream(senderSocket.getInputStream())) {

            while (true) {
                // Read the file name
                String fileName;
                try {
                    fileName = (String) objectInputStream.readObject();
                } catch (EOFException eofException) {

                    break;
                }

                //  exit signal
                if ("exit".equalsIgnoreCase(fileName)) {
                        break;
                    }

                    // Read the file content
                    try (FileOutputStream fileOutputStream = new
FileOutputStream(DOWNLOADS_DIRECTORY + fileName)) {
                        byte[] buffer = new byte[4096];
                        int bytesRead;
                        while ((bytesRead = objectInputStream.read(buffer)) != -1) {
                            fileOutputStream.write(buffer, 0, bytesRead);
                        }
                    }

                    System.out.println("File '" + fileName + "' received successfully and saved in
Downloads directory.");
                }

            } catch (ClassNotFoundException | IOException e) {
                e.printStackTrace();
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

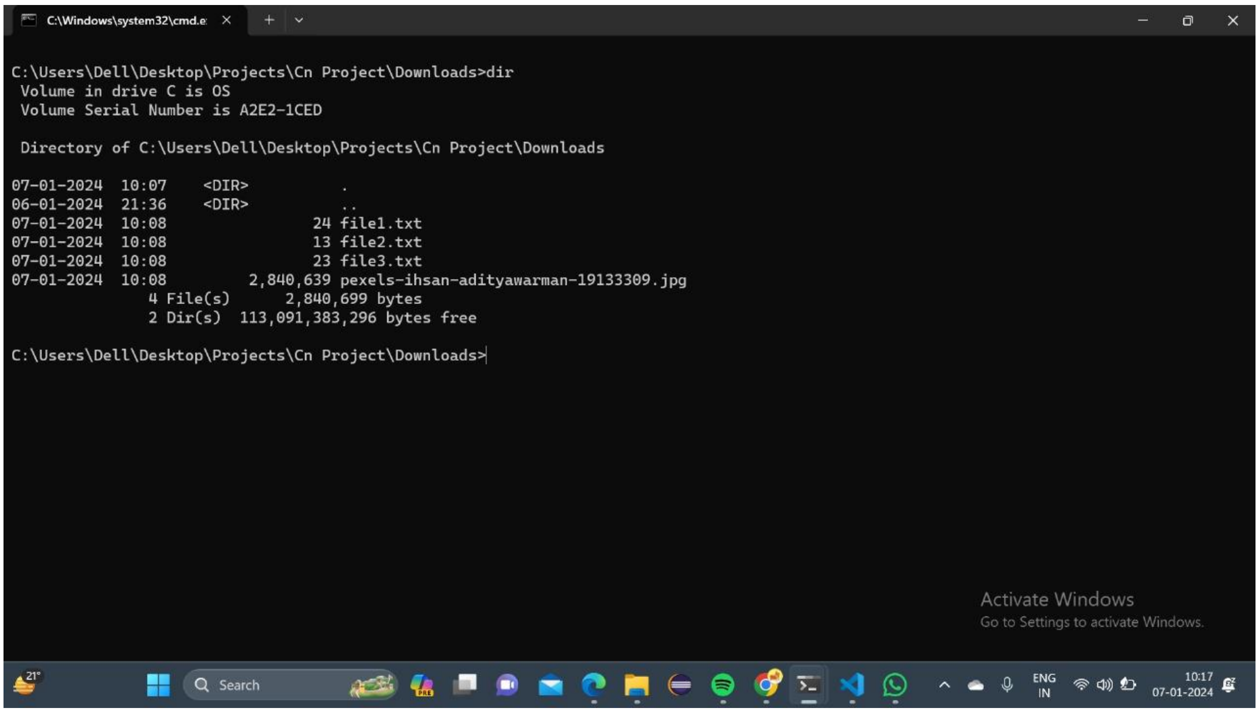
5. Results & Interpretation

1. Output


C:\Windows\System32\cmd.e
C:\Windows\system32\cmd.e
Activate Windows
Go to Settings to activate Windows.

```
C:\Users\Dell\Desktop\Projects\Cn Project>javac P2PFileSharingSy
stem.java

C:\Users\Dell\Desktop\Projects\Cn Project>
```

```
C:\Users\Dell\Desktop\Projects\Cn Project>code .

C:\Users\Dell\Desktop\Projects\Cn Project>java P2PFileSharingSys
tem.java
Waiting for sender to connect...
File 'file1.txt' sent successfully.
File 'file2.txt' sent successfully.
File 'file1.txt' received successfully and saved in Downloads di
rectory.
File 'file3.txt' sent successfully.
File 'file2.txt' received successfully and saved in Downloads di
rectory.
File 'file3.txt' received successfully and saved in Downloads di
rectory.
File 'pexels-ihsan-adityawarman-19133309.jpg' sent successfully.
File 'pexels-ihsan-adityawarman-19133309.jpg' received successfu
lly and saved in Downloads directory.

C:\Users\Dell\Desktop\Projects\Cn Project>
```


C:\Windows\system32\cmd.e
Activate Windows
Go to Settings to activate Windows.
21°
Search
ENG
IN
10:17
07-01-2024

```
C:\Users\Dell\Desktop\Projects\Cn Project\Downloads>dir
 Volume in drive C is OS
 Volume Serial Number is A2E2-1CED

 Directory of C:\Users\Dell\Desktop\Projects\Cn Project\Downloads

07-01-2024  10:07    <DIR>          .
06-01-2024  21:36    <DIR>          ..
07-01-2024  10:08                24 file1.txt
07-01-2024  10:08                13 file2.txt
07-01-2024  10:08                23 file3.txt
07-01-2024  10:08         2,840,639 pexels-ihsan-adityawarman-19133309.jpg
               4 File(s)      2,840,699 bytes
               2 Dir(s)  113,091,383,296 bytes free

C:\Users\Dell\Desktop\Projects\Cn Project\Downloads>
```

Explanation of the Code:

The provided Java code implements a simple Peer-to-Peer (P2P) file sharing system between a sender and a receiver. The sender and receiver components run concurrently as separate threads. The sender simulates the sharing of multiple files (specified in the filesToSend array) with the receiver, and the receiver saves the received files in the "Downloads" directory.

- Here is a brief overview of the code:
  - Sender (startSender) Function:
  - The sender establishes a socket connection with the receiver using the specified IP address and port.
  - An ObjectOutputStream is created to write objects (file names and content) to the output stream.
  - The sender iterates through an array of file names (filesToSend), sending each file's name and content to the receiver.
  - The file content is read in chunks and written to the output stream.
  - A special string ("exit") is sent to signal the end of file transmission.
- Receiver (startReceiver) Function:
  - The receiver sets up a ServerSocket to listen for incoming connections on the specified port.
  - It waits for the sender to connect, and once connected, it accepts the socket connection.
  - An ObjectInputStream is created to read objects from the input stream.
  - The receiver continuously reads file names and content from the input stream until it receives the "exit" signal.
  - Received files are saved in the "Downloads" directory.
- Main Function:
  - The main function initializes the IP addresses of the sender and receiver.
  - Threads are created for the sender and receiver functions, and both threads are started.
  - The code simulates a P2P file-sharing scenario where files are

transmitted from the sender to the receiver over a network connection. The file names and content are serialized and deserialized using ObjectInputStream and ObjectOutputStream, respectively.

- The program prints messages to the console indicating the successful transmission of files and the reception of files at the receiver's end.

6. Conclusion

In conclusion, our P2P Distributed File Sharing System stands as a testament to the capabilities of Java IO and threading, offering a seamless and direct file-sharing experience between peers without reliance on a central server. The project showcases robust features, including Peer Discovery, efficient File Transfer mechanisms, Concurrency support, and an intuitively designed File Management interface. This system embraces decentralized collaboration, allowing users to share files effortlessly in a distributed environment.

The implementation leverages Java's socket programming, ObjectInputStream, and ObjectOutputStream, enabling efficient communication and file transfer between sender and receiver components. The inclusion of multithreading enhances the concurrent operation of both sender and receiver, optimizing the overall performance of the system.

While the current implementation serves as a solid foundation, there are opportunities for future enhancements. Addressing dynamic IP address handling, implementing security measures such as encryption for secure file transmission, refining error handling, and allowing users to input files dynamically are considerations for further development.

Looking ahead, potential improvements could involve the integration of a graphical user interface, optimization of network protocols, and a focus on bolstering security aspects to ensure reliability and usability in diverse real-world scenarios. Overall, this project lays a strong foundation for the evolution and refinement of P2P file-sharing systems, reflecting the commitment to decentralized collaboration in a user-friendly and efficient manner.

References

(as per the IEEE recommendations)

[1] Computer Networks, Andrew S. Tannenbaum, Pearson India.

[2] Java Network Programming by Harold, O'Reilly (Shroff Publishers).

[3] Peer to Peer File Sharing System by Adel Ali Al-Zebrari