

پیاده‌سازی Tracker:

ابتدا با شروع از تابع main یک tracker روی پورت 8080 میسازیم.

```

Main.java x Tracker.java Connection.java Database.java File.java Packet.java Peer.java SystemLog.java
1 package org.example;
2
3 public class Main {
4     public static void main(String[] args) {
5         Tracker tracker = new Tracker( port: 8080);
6     }
7 }

```

حال در tracker و در تابع start با استفاده از پورت داده شده و کلاس DatagramSocket یک کانکشن UDP ایجاد می‌کنیم. در آغاز کار tracker پورت آن را چاپ می‌کنیم. در یک حلقه بی‌نهایت پیام‌ها را از peers دریافت می‌کنیم به این‌صورت که یک بافر برای ذخیره داده دریافتی در نظر می‌گیریم و با استفاده از کلاس DatagramPacket بسته را روی پورت موردنظر دریافت می‌کنیم. سپس در کلاس connection یک Thread برای پردازش درخواست ایجاد می‌شود تا برنامه به صورت multithread اجرا شود.

```

Main.java Tracker.java x Connection.java Database.java File.java Packet.java Peer.java SystemLog.java
1 package org.example;
2
3 import java.net.*;
4
5 public class Tracker { 2 usages
6     private final int port; 3 usages
7
8     public Tracker(int port) { 1 usage
9         this.port = port;
10        start();
11    }
12
13    private void start() { 1 usage
14        try (DatagramSocket socket = new DatagramSocket(port)) {
15            System.out.println("Tracker started on port " + port);
16
17            while (true) {
18                byte[] receiveBuffer = new byte[1024];
19                DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);
20                socket.receive(receivePacket);
21                Connection connection = new Connection(socket, receivePacket);
22                connection.start();
23            }
24        } catch (Exception e) {
25            e.printStackTrace();
26        }
27    }
28 }

```

سپس در کلاس connection ابتدا مقادیر اولیه ورودی مقداردهی می‌شوند.

```

Main.java Tracker.java Connection.java Database.java File.java Packet.java Peer.java SystemLog.java
7  import java.io.IOException;
8  import java.util.ArrayList;
9  import java.util.Map;
10 import java.util.Random;
11 import java.util.Scanner;
12
13 public class Connection extends Thread { 3 usages
14     private final DatagramSocket socket; 1 usage
15     private final DatagramPacket packet; 10 usages
16
17     public Connection(DatagramSocket socket, DatagramPacket packet) { 1 usage
18         this.socket = socket;
19         this.packet = packet;
20     }

```

حال در تابع run ابتدا یک thread جدید برای پردازش دستورات لاگ ایجاد می‌شود. سپس داده‌های دریافتی از بسته UDP خوانده و چاپ می‌شود. همچنین با استفاده از کلاس peer یک آبجکت جدید ایجاد کرده و لیست اطلاعات peerها آپدیت می‌شود. در نهایت داده دریافتی از peer در تابع handleRequest پردازش شده و پاسخ تولید شده به peer فرستاده ارسال می‌شود.

```

Main.java Tracker.java Connection.java Database.java File.java Packet.java Peer.java SystemLog.java
13 public class Connection extends Thread { 3 usages
22     @Override no usages
23     public void run() {
24         new Thread(Connection::handleLogs).start();
25         try {
26             String receivedData = new String(packet.getData(), offset: 0, packet.getLength());
27             System.out.println("Received from: " + packet.getAddress() + ":" + packet.getPort() + " -> " + receivedData);
28             Peer currentPeer= new Peer(generatePeerId(),packet.getPort(),packet.getAddress(), trackerPort: 8080);
29             if(!Database.checkPeerExists(currentPeer)){
30                 Database.addPeer(currentPeer);
31             }
32
33             String response = handleRequest(receivedData,currentPeer);
34             byte[] responseData = response.getBytes();
35             DatagramPacket responsePacket = new DatagramPacket(responseData, responseData.length,
36                 packet.getAddress(), packet.getPort());
37             try (DatagramSocket socket = new DatagramSocket()) {
38                 socket.send(responsePacket);
39             }
40
41         } catch (Exception e) {
42             System.out.println(e.getMessage());
43         }
44     }

```

در کلاس peer اطلاعات زیر برای هر peer در هنگام ایجاد ذخیره می‌شود. همچنین لیستی برای فایل‌های به اشتراک گذاشته شده توسط هر peer نیز در نظر گرفته شده است.

```

@ Main.java  @ Tracker.java  @ Connection.java  @ Database.java  @ File.java  @ Packet.java  @ Peer.java x  @ SystemLog.java
6  public class Peer { 18 usages
7      private String id; 3 usages
8      private int port; 3 usages
9      private InetAddress ip; 3 usages
10     private int trackerPort; 3 usages
11     private ArrayList<File> files; 3 usages
12
13     public Peer(String id, int port, InetAddress ip, int trackerPort) { 1 usage
14         this.id = id;
15         this.port = port;
16         this.ip = ip;
17         this.trackerPort = trackerPort;
18         this.files = new ArrayList<>();
19     }

```

در کلاس فایل دو ویژگی اسم و پورت مربوط به آن در نظر گرفته می‌شود.

```

@ Main.java  @ Tracker.java  @ Connection.java  @ Database.java  @ File.java x  @ Packet.java  @ Peer.java  @ SystemLog.java
1  package org.example;
2
3  public class File { 8 usages
4      private String name; 3 usages
5      private int port; 3 usages
6
7      public File(String name, int port) { 3 usages
8          this.name = name;
9          this.port = port;
10     }

```

در کلاس Database دو لیست foundPeers و allPeers داریم که لیست تمام peerهای متصل است و foundPeers برای نمایش لاگ‌ها استفاده می‌شود.

```

@ Main.java  @ Tracker.java  @ Connection.java  @ Database.java x  @ File.java  @ Packet.java  @ Peer.java  @ SystemLog.java
8  public class Database { 7 usages
9      public static ArrayList<Peer> foundPeers = new ArrayList<>(); 3 usages
10     private static ArrayList<Peer> allPeers = new ArrayList<>(); 7 usages
11
12
13     public static void addPeer(Peer peer) { 1 usage
14         allPeers.add(peer);
15     }
16
17     public static void removePeer(Peer peer) { 1 usage
18         for (Peer p : allPeers) {
19             if (p.getPort() == peer.getPort()) {
20                 allPeers.remove(peer);
21                 return;
22             }
23         }
24     }
25
26     public static boolean checkPeerExists(Peer peer) { 1 usage
27         for (Peer p : allPeers) {
28             if (p.getPort() == peer.getPort()) {
29                 return true;
30             }
31         }
32         return false;
33     }

```

در تابع `handleLogs` دستورات لاگ در خط فرمان `tracker` را مدیریت می‌کند و به ترتیب هر کدام برای نمایش درخواست‌های `get` و موفقیت یا عدم موفقیت آن، درخواست‌های `share` و وضعیت ارسال و دریافت یک فایل مشخص استفاده می‌شود.

```
13 public class Connection extends Thread { 3 usages
46 private static void handleLogs(){ 1 usage
47     Scanner scanner = new Scanner(System.in);
48     while (true) {
49         String command = scanner.nextLine().trim();
50
51         if (command.equals("logs request")) {
52             SystemLog.printRequests();
53         } else if (command.equals("all-logs")) {
54             SystemLog.printLogs();
55         } else if (command.startsWith("file_logs ")) {
56             String fileName = command.substring(10).trim();
57             SystemLog.printFileLogs(fileName);
58         }
59     }
60 }
```

کلاس `SystemLog` برای مدیریت و ذخیره لاگ‌ها استفاده می‌شود. در آن لاگ درخواست‌ها و تمام لاگ‌ها و لاگ‌های مربوط به هر فایل و درخواست‌های هر `peer` به صورت لیست‌ها و `map` ذخیره می‌شود.

```
8 public class SystemLog { 10 usages
9     private static List<String> logRequests = new ArrayList<>(); 3 usages
10    private static List<String> allLogs = new ArrayList<>(); 3 usages
11    private static Map<String, ArrayList<String>> fileLogs = new HashMap<>(); 4 usages
12    private static Map<String, ArrayList<String>> peerReq = new HashMap<>(); 3 usages
13
14    public static List<String> getLogRequests() { no usages
15        return logRequests;
16    }
17
18    public static void addLogRequest(String req){ 2 usages
19        logRequests.add(req);
20    }
21
22    public static List<String> getAllLogs() { no usages
23        return allLogs;
24    }
25
26    public static void addLog(String log){ 1 usage
27        allLogs.add(log);
28    }
29
30    public static void addFileLog(String file, String text){ 2 usages
31        fileLogs.putIfAbsent(file, new ArrayList<>());
32        fileLogs.get(file).add(text);
33    }
```

```

Main.java Tracker.java Connection.java Database.java File.java Packet.java Peer.java SystemLog.java x
8 public class SystemLog { 10 usages
35     public static void addPeerReq(String file, String text){ 1 usage
36         peerReq.putIfAbsent(file, new ArrayList<>());
37         peerReq.get(file).add(text);
38     }
39
40     public static void printRequests(){ 1 usage
41         logRequests.forEach(System.out::println);
42     }
43
44     public static void printLogs(){ 1 usage
45         allLogs.forEach(System.out::println);
46     }
47
48     public static void printFileLogs(String fileName){ 1 usage
49         if (fileLogs.containsKey(fileName)) {
50             System.out.println("Logs for file " + fileName + ": ");
51             fileLogs.get(fileName).forEach(System.out::println);
52         } else {
53             System.out.println("Error: File not found.");
54         }
55     }
56
57     public static Map<String, ArrayList<String>> getPeerReq() { 1 usage
58         return peerReq;
59     }
60 }

```

در تابع `handleRequest` درخواست‌های دریافتی از `peer` پردازش می‌شود. سپس برای راحتی پردازش درخواست با استفاده از فرمت کلاس `packet` تجزیه می‌شود.

```

Main.java Tracker.java Connection.java x Database.java File.java Packet.java Peer.java SystemLog.java
13 public class Connection extends Thread { 3 usages
62     private String handleRequest(String data, Peer currentPeer) throws IOException { 1 usage
63         if (data.isEmpty())
64             return null;
65
66         try {
67             Packet packet = new Gson().fromJson(data, Packet.class);
68             String topic = packet.topic;
69             String value = packet.value;
70             String command = packet.command;
71             String extra = packet.extra;

```

در دستور `exit` باید `peer` از دیتابیس حذف شود و پیام قطع ارتباط چاپ شود و در دستور `request logs` لیستی از تمام درخواست‌های ثبت شده توسط `peer` چاپ می‌شود.

```

Main.java Tracker.java Connection.java x Database.java File.java Packet.java Peer.java SystemLog.java
13 public class Connection extends Thread { 3 usages
62     private String handleRequest(String data, Peer currentPeer) throws IOException { 1 usage
73         switch (command) {
74             case "exit" ->{
75                 Database.removePeer(currentPeer);
76                 System.out.println("Peer" + " Disconnected!");
77                 return ("Peer Disconnected");
78             }
79             case "request logs" -> {
80                 StringBuilder sb = new StringBuilder();
81                 for (Map.Entry<String, ArrayList<String>> entry : SystemLog.getPeerReq().entrySet()) {
82                     sb.append("File: ").append(entry.getKey()).append("\n");
83                     for (String request : entry.getValue()) {
84                         sb.append(" → Request: ").append(formatRequest(request)).append("\n");
85                     }
86                 }
87                 return sb.toString();
88             }

```

در دستور get اگر فایل مورد نظر از قبل موجود بود تمام peerهایی که آن فایل را دارند پیدا کرده و سپس پورت یکی از آنها را به صورت تصادفی به peer ارسال می‌کند. اگر فایل موجود نباشد پیغام خطا را چاپ می‌کند. همچنین وضعیت لاگ‌ها را نیز آپدیت کرده و فایل را به لیست فایل‌هایی که peer دارد اضافه می‌کند.

```
13 public class Connection extends Thread { 3 usages
62 private String handleRequest(String data, Peer currentPeer) throws IOException { 1 usage
89     case "get" -> {
90         ArrayList<Peer> foundPeers = Database.getFilePort(extra);
91         if (!foundPeers.isEmpty()) {
92             SystemLog.addFileLog(extra, text: currentPeer.getId() + ": " + command);
93             SystemLog.addLogRequest(currentPeer.getId() + ": " + command + " " + extra + " "
94                 + ", Successful" + "\n" + foundPeersToString(foundPeers));
95             Random random = new Random();
96             Peer randomPeer = foundPeers.get(random.nextInt(foundPeers.size()));
97             SystemLog.addPeerReq(extra, text: currentPeer.getId() + "get"+extra+"from"+randomPeer.getId());
98             int filePort=Database.findFilePort(randomPeer.getPort(),extra);
99             currentPeer.addFile(new File(extra,Integer.parseInt(topic)));
100             Database.updatePeer(topic,extra);
101             return String.valueOf(filePort);
102         } else {
103             SystemLog.addLogRequest(generatePeerId() + ": " + command + extra +
104                 ",Fail" + "\n No Peer found for this file");
105             return "400: No such file exists";
106         }
107     }
```

در دستور share اطلاعات فایل به دیتابیس اضافه شده و به لیست فایل‌های peer و لاگ‌ها نیز اضافه می‌شود.

```
13 public class Connection extends Thread { 3 usages
62 private String handleRequest(String data, Peer currentPeer) throws IOException { 1 usage
108     case "share" -> {
109         SystemLog.addFileLog(extra, text: currentPeer.getId() + " shared");
110         SystemLog.addLog(currentPeer.getId() + " shared " + extra);
111         currentPeer.addFile(new File(extra,Integer.parseInt(topic)));
112         Database.updatePeer(topic,extra);
113         return "200: "+extra+" shared successfully!";
114     }
115     default -> {
116         return ("400: Invalid command.");
117     }
118 }
119 } catch (JsonSyntaxException e) {
120     return ("400: Missing topic, value, or command fields.");
121 }
122 }
```

در آخر نیز اگر دستور ناشناخته باشد پیغام خطا چاپ می‌شود.

دو تابع زیر نیز برای ساخت رشته‌ای که قرار است چاپ شود است.

```
124 @ private static String formatRequest(String request) { 1 usage
125     return request.replaceAll( regex: "(peer\\d+)(get)(.)(from)(peer\\d+)", replacement: "$1 $2 $3 $4 $5");
126 }
127
128 @ private String foundPeersToString(ArrayList<Peer> foundPeers) { 1 usage
129     StringBuilder sb = new StringBuilder();
130     for(Peer p:foundPeers){
131         sb.append(p.getId()).append(", ");
132     }
133     return sb.toString();
134 }
```

در کلاس packet درخواست به 4 بخش دستور، پورتی که در آن گوش می‌دهد، پورت tracker و فایل مورد نظر تقسیم می‌شود.

```
1 package org.example;
2
3 > import ...
4
5
6 public class Packet implements Serializable { 2 usages
7     String command; 2 usages
8     String topic; 2 usages
9     String value; 2 usages
10    String extra; 2 usages
11
12    public Packet(String command, String topic, String value, String extra) { no usages
13        this.command = command;
14        this.topic = topic;
15        this.value = value;
16        this.extra = extra;
17    }
18
19 > public String toJson() { return new Gson().toJson(src: this); }
22 }
```

پیاده‌سازی Peer:

ابتدا با شروع از تابع main یک peer روی پورت 8080 می‌سازیم.

```
1 package org.example;
2
3 import java.io.IOException;
4
5 > public class Main {
6 >     public static void main(String[] args) throws IOException {
7         Peer peer = new Peer(host: "localhost", port: 8080);
8     }
9 }
```

در کلاس peer ابتدا آدرس و پورت و دیگر متغیرهای ثابت کلاس را مقداردهی می‌کنیم.

```
9 public class Peer { 3 usages
10     private static final String TRACKER_ADDRESS = "127.0.0.1"; 4 usages
11     private static final int TRACKER_PORT = 8080; 4 usages
12     private static int PEER_SERVER_PORT; 4 usages
13     private static Set<String> sharedFiles = new HashSet<>(); 2 usages
14     private String serverAddress; 2 usages
15     private int serverPort; 2 usages
16
17     public Peer(String host, int port) throws IOException { 1 usage
18         this.serverAddress = host;
19         this.serverPort = port;
```

در این بخش ابتدا در یک thread جداگانه یک سرور فایل و یک سوکت برای ارتباط UDP با tracker ایجاد می‌کنیم و بعد حلقه‌ای برای گرفتن دستورات از کاربر ایجاد می‌شود.

```
9 public class Peer { 3 usages
17 public Peer(String host, int port) throws IOException { 1 usage
21     new Thread(Peer::startFileServer).start();
22
23     Scanner scanner = new Scanner(System.in);
24     DatagramSocket socket = new DatagramSocket();
25     while (true) {
26         String[] params = scanner.nextLine().split(" ");
27         if (params[0].equals("exit")) {
28             endPeer(socket);
29             System.exit(status: 0);
30         } else if ((params[0] + " " + params[1]).equals("request logs")) {
31             requestLogs(socket);
32         } else if (params[0].equals("share")) {
33             String f = params[3];
34             sharedFiles.add(f);
35             registerFile(f, socket);
36         } else if (params[0].equals("get")) {
37             String selectedFile = params[3];
38             requestFile(selectedFile, socket);
39         }
40     }
41 }
42 }
```

در تابع startFileServer یک سرور فایل محلی روی پورت آزاد بین دو peer ایجاد می‌شود و هر بار که یک درخواست فایل به سرور می‌رسد، یک رشته جدید برای پردازش درخواست ایجاد می‌شود.

```
9 public class Peer { 3 usages
63 private static void startFileServer() { 1 usage
64     try (ServerSocket serverSocket = new ServerSocket(port: 0)) {
65         PEER_SERVER_PORT = serverSocket.getLocalPort();
66         System.out.println("File server running on port " + PEER_SERVER_PORT);
67
68         while (true) {
69             Socket clientSocket = serverSocket.accept();
70             new Thread(() -> handleFileRequest(clientSocket)).start();
71         }
72     } catch (Exception e) {
73         e.printStackTrace();
74     }
75 }
```


در تابع `handleRequest` پردازش درخواست‌های فایل از `Peers` انجام می‌شود. اگر فایل خواسته‌شده در لیست فایل‌های به اشتراک گذاشته‌شده باشد، فایل ارسال می‌شود. در غیر این صورت، یک پیام خطا ارسال می‌شود.

```
9 public class Peer { 3 usages
77 private static void handleFileRequest(Socket clientSocket) { 1 usage
78     try {
79         BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
80         OutputStream out = clientSocket.getOutputStream();
81
82         String fileName = reader.readLine();
83         System.out.println("Received request for file: " + fileName);
84
85
86         if (sharedFiles.contains(fileName)) {
87             System.out.println("File '" + fileName + "' sent to " + clientSocket.getInetAddress().getHostAddress());
88         } else {
89             out.write("ERROR: File not found\n".getBytes());
90             System.out.println("Requested file '" + fileName + "' not found.");
91         }
92
93         clientSocket.close();
94     } catch (Exception e) {
95         e.printStackTrace();
96     }
97 }
```

اگر دستور `exit` بود در تابع `endPeer` یک دستور برای `tracker` ارسال می‌شود و با چاپ پیغام آن برنامه تمام می‌شود.

```
9 public class Peer { 3 usages
53 @ private static void endPeer(DatagramSocket socket) throws IOException { 1 usage
54     String message = new Packet(command: "exit", topic: "", value: "", extra: "").toJson();
55     byte[] data = message.getBytes();
56     DatagramPacket packet = new DatagramPacket(data, data.length,
57         InetAddress.getByAddress(TRACKER_ADDRESS), TRACKER_PORT);
58     socket.send(packet);
59     System.out.println(receivePacket(socket));
60 }
```

اگر دستور `request logs` بود در تابع `requestLogs` درخواست دریافت لاگ به `tracker` ارسال شده و پاسخ آن چاپ می‌شود.

```
9 public class Peer { 3 usages
44 @ private static void requestLogs(DatagramSocket socket) throws IOException { 1 usage
45     String message = new Packet(command: "request logs", topic: "", value: "", extra: "").toJson();
46     byte[] data = message.getBytes();
47     DatagramPacket packet = new DatagramPacket(data, data.length,
48         InetAddress.getByAddress(TRACKER_ADDRESS), TRACKER_PORT);
49     socket.send(packet);
50     System.out.println(receivePacket(socket));
51 }
```

اگر دستور share بود در تابع registerFile یک پیغام share به tracker ارسال شده تا فایل در لیست فایل‌ها ذخیره شود.

```
9 public class Peer { 3 usages
99 @ private static void registerFile(String fileName, DatagramSocket socket) throws IOException { 1 usage
100     String message = new Packet(command: "share", String.valueOf(PEER_SERVER_PORT), value: "8080", fileName).toJson();
101     byte[] data = message.getBytes();
102     DatagramPacket packet = new DatagramPacket(data, data.length,
103         InetAddress.getByName(TRACKER_ADDRESS), TRACKER_PORT);
104     socket.send(packet);
105     System.out.println(receivePacket(socket));
106     System.out.println("File registered with tracker: " + fileName);
107 }
```

اگر دستور get بود در تابع requestFile یک دستور دریافت فایل به tracker ارسال می‌شود اگر فایل موجود باشد پورت peer را از tracker دریافت می‌کند.

```
9 public class Peer { 3 usages
109 @ private static void requestFile(String fileName, DatagramSocket socket) throws IOException { 1 usage
110     String message = new Packet(command: "get", String.valueOf(PEER_SERVER_PORT), value: "8080", fileName).toJson();
111     byte[] data = message.getBytes();
112     DatagramPacket packet = new DatagramPacket(data, data.length,
113         InetAddress.getByName(TRACKER_ADDRESS), TRACKER_PORT);
114     socket.send(packet);
115
116     byte[] buffer = new byte[1024];
117     DatagramPacket responsePacket = new DatagramPacket(buffer, buffer.length);
118     socket.receive(responsePacket);
119     String response = new String(responsePacket.getData(), offset: 0, responsePacket.getLength());
120
121     if (response.equals("400: No such file exists")) {
122         System.out.println("No peers have this file.");
123     } else {
124         System.out.println("Peers with file: " + response);
125         String peerAddress = "127.0.0.1";
126         int peerPort = Integer.parseInt(response);
127         downloadFile(peerAddress, peerPort, fileName);
128     }
129 }
```

سپس در تابع downloadFile فایل را از peer دیگر دریافت کرده و آن را ذخیره می‌کند.

```
9 public class Peer { 3 usages
131 private static void downloadFile(String peerAddress, int peerPort, String fileName) { 1 usage
132     try {
133         Socket socket = new Socket(peerAddress, peerPort);
134         OutputStream out = socket.getOutputStream();
135         out.write((fileName + "\n").getBytes());
136
137         FileOutputStream fos = new FileOutputStream(name: "downloaded_" + fileName);
138         InputStream in = socket.getInputStream();
139         byte[] buffer = new byte[1024];
140         int bytesRead;
141
142         while ((bytesRead = in.read(buffer)) != -1) {
143             fos.write(buffer, off: 0, bytesRead);
144         }
145
146         fos.close();
147         in.close();
148         socket.close();
149         System.out.println("File '" + fileName + "' downloaded from " + peerAddress + ":" + peerPort);
150     } catch (Exception e) {
151         e.printStackTrace();
152     }
153 }
```

دو تابع کمکی زیر برای ارسال و دریافت پیغام‌ها به tracker استفاده می‌شوند.

```
155 @ private void sendPacket(DatagramSocket clientSocket, String m) throws IOException { no usages
156     String message = m;
157     byte[] sendData = message.getBytes();
158     InetAddress serverIP = InetAddress.getByName(serverAddress);
159     DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverIP, serverPort);
160     clientSocket.send(sendPacket);
161     System.out.println("Sent: " + message);
162 }
163
164 @ private static String receivePacket(DatagramSocket clientSocket) throws IOException { 3 usages
165     byte[] receiveBuffer = new byte[1024];
166     DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);
167     clientSocket.receive(receivePacket);
168     String receivedData = new String(receivePacket.getData(), offset: 0, receivePacket.getLength());
169     return receivedData;
170 }
171 }
```

اجرای برنامه:

ابتدا برنامه tracker را اجرا می‌کنیم.

```
Run Run x
/usr/lib/jvm/java-21-openjdk/bin/java -javaagent:/usr/share/idea/lib/idea_rt.jar=40773:/usr/share/idea/bin -Dfile.encoding=
Tracker started on port 8080
```

سپس برای مثال دو برنامه peer را ران می‌کنیم.

```
Run Run x Run x
/usr/lib/jvm/java-21-openjdk/bin/java -javaagent:/usr/share/idea/lib/idea_rt.jar=44177:/usr/share/idea/bin -Dfile.encoding=
File server running on port 34365
Peer running on port 40964
```

```
Run Run x Run x
/usr/lib/jvm/java-21-openjdk/bin/java -javaagent:/usr/share/idea/lib/idea_rt.jar=36203:/usr/share/idea/bin -Dfile.encoding=
File server running on port 34721
Peer running on port 54956
```

حال برای مثال در یک peer یک فایل را share می‌کنیم و در دیگری آن فایل را دریافت می‌کنیم.

```
Run Run x Run x
/usr/lib/jvm/java-21-openjdk/bin/java -javaagent:/usr/share/idea/lib/idea_rt.jar=44177:/usr/share/idea/bin -Dfile.encoding=
File server running on port 34365
Peer running on port 40964
share 34365 8080 x.txt
200: x.txt shared successfully!
File registered with tracker: x.txt
Received request for file: x.txt
File 'x.txt' sent to 127.0.0.1
```

```
Run Run Run
File server running on port 34721
Peer running on port 54956
get 34721 8080 x.txt
Peers with file: 34365
File 'x.txt' downloaded from 127.0.0.1:34365
```

اگر درخواست برای فایل بدهیم که وجود ندارد:

```
Run Run Run
File server running on port 34721
Peer running on port 54956
get 34721 8080 x.txt
Peers with file: 34365
File 'x.txt' downloaded from 127.0.0.1:34365
get 34721 8080 n.txt
No peers have this file.
```

حال اگر دستورات لاگ مربوط به tracker را اجرا کنیم:

```
Run Run
Received from: /127.0.0.1:54956 -> {"command": "get", "topic": "34721", "value": "8080", "extra": "n.txt"}
logs request
peer54956: get x.txt , Successful
peer40964,
peer54956: getn.txt,Fail
No Peer found for this file
all-logs
peer40964 shared x.txt
file_logs x.txt
Logs for file x.txt:
peer40964 shared
peer54956: get
```

و درخواست گرفتن لاگ در peer:

```
Run Run Run
File server running on port 34721
Peer running on port 54956
get 34721 8080 x.txt
Peers with file: 34365
File 'x.txt' downloaded from 127.0.0.1:34365
get 34721 8080 n.txt
No peers have this file.
request logs
File: x.txt
→ Request: peer54956 get x.txt from peer40964
```