**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC KINH TẾ - KỸ THUẬT CÔNG NGHIÊP**

🙦🙦🕮🙦🙦



**BÀI TẬP LỚN**

**MÔN: CÔNG NGHỆ JAVA**

**Câu hỏi:**

***Lập trình client-server sử dụng UDP socket***

|  |  |  |
| --- | --- | --- |
| GVHD | : | VŨ VĂN ĐỐC |
| LỚP | : | DHTI16A6HN |
| NHÓM | : | 04 |
| THÀNH VIÊN | : | HOÀNG THANH CHIẾN – 22103100356 |
|  |  | NGUYỄN VIẾT HUY - 22103100320 |
|  |  | TRẦN XUÂN CƯ |
|  |  | NGUYỄN QUÝ CƯỜNG |
|  |  | NGUYỄN TIẾN LƯỢNG - 22103100328 |
|  |  | NGUYỄN XUÂN DUY |

**Hà Nội, ngày 5 tháng 12 năm 2025**

# **MỤC LỤC**

[**MỤC LỤC** 1](#_Toc183898887)

[**1. Mở đầu** 2](#_Toc183898888)

[**1.1 Bối cảnh thực hiện đề tài** 2](#_Toc183898889)

[**1.2 Mục tiêu của báo cáo** 2](#_Toc183898890)

[**2. Tổng quan về giao thức UDP** 2](#_Toc183898891)

[**2.1 Một số thuật ngữ UDP** 2](#_Toc183898892)

[**❖ Packet** 2](#_Toc183898893)

[**❖ Datagram** 3](#_Toc183898894)

[**❖ MTU** 3](#_Toc183898895)

[**❖ Port** 3](#_Toc183898896)

[❖ **Multicasting** 4](#_Toc183898897)

[**2.2 Hoạt động của giao thức UDP** 4](#_Toc183898898)

[**2.3 Các nhược điểm của giao thức UDP** 4](#_Toc183898899)

[**2.4 Các ưu điểm của UDP** 5](#_Toc183898900)

[**3. Lớp DatagramPacket** 6](#_Toc183898901)

[**3.1 Constructor của DatagramPacket** 6](#_Toc183898902)

[**❖ Gửi dữ liệu** 6](#_Toc183898903)

[**❖ Nhận dữ liệu** 6](#_Toc183898904)

# **1. Mở đầu**

## **1.1 Bối cảnh thực hiện đề tài**

Trong thời đại công nghệ thông tin phát triển mạnh mẽ, các ứng dụng mạng đang đóng vai trò quan trọng trong việc kết nối và trao đổi dữ liệu giữa các thiết bị. Mô hình client-server là một trong những mô hình phổ biến nhất, được ứng dụng trong nhiều lĩnh vực như truyền phát dữ liệu, game online, và các dịch vụ trực tuyến.

UDP (User Datagram Protocol) là một giao thức truyền tải dữ liệu không kết nối, nổi bật với tốc độ nhanh và giao thức đơn giản. Mặc dù không đảm bảo độ tin cậy như TCP, UDP được ưa chuộng trong các ứng dụng yêu cầu tốc độ cao và chấp nhận mất mát dữ liệu nhỏ.

Việc nghiên cứu lập trình client-server bằng UDP Socket mang lại sự hiểu biết sâu sắc về cách thức truyền tải dữ liệu qua mạng, đồng thời cung cấp nền tảng để triển khai các ứng dụng thực tế, đặc biệt là trong các lĩnh vực yêu cầu hiệu năng cao.

## **1.2 Mục tiêu của báo cáo**

* Tìm hiểu nguyên lý hoạt động của giao thức UDP trong lập trình mạng.
* Nắm vững cách sử dụng DatagramSocket và DatagramPacket trong Java.
* Xây dựng một số mô hình client-server cơ bản, minh họa cách truyền và nhận dữ liệu qua UDP Socket.

# **2. Tổng quan về giao thức UDP**

Trong lập trình mạng, việc xây dựng mô hình client-server là một trong những yêu cầu cơ bản để các ứng dụng giao tiếp qua mạng. UDP (User Datagram Protocol) là một giao thức không kết nối, nhẹ và nhanh, thường được sử dụng trong các trường hợp yêu cầu tốc độ cao và mức độ tin cậy không phải là ưu tiên chính.

**2.1 Một số thuật ngữ UDP**

Trước khi kiểm tra xem giao thức UDP hoạt động như thế nào, chúng ta cần làm quen với một số thuật ngữ. Trong phần dưới đây, chúng ta sẽ định nghĩa một số thuật ngữ cơ bản có liên quan đến giao thức UDP.

### **❖ Packet**

Trong lĩnh vực truyền dữ liệu, một gói tin (packet) là một chuỗi số nhị phân bao gồm dữ liệu và các tín hiệu điều khiển. Các gói tin này được gửi từ nguồn đến đích (host). Bên trong mỗi gói tin, thông tin được tổ chức theo một định dạng nhất định.

### **❖ Datagram**

Datagram là một gói tin tự trị, chứa toàn bộ dữ liệu cần thiết để định tuyến từ nguồn đến đích mà không yêu cầu bất kỳ thông tin bổ sung nào.

### **❖ MTU**

MTU, viết tắt của **Maximum Transmission Unit**, là một thuộc tính của tầng liên kết, chỉ số byte dữ liệu tối đa có thể được truyền trong một gói tin. MTU cũng được hiểu là kích thước gói dữ liệu lớn nhất mà một môi trường mạng cụ thể có thể xử lý. Chẳng hạn, mạng Ethernet có giá trị MTU cố định là 1500 byte. Trong giao thức UDP, nếu kích thước của một datagram vượt quá MTU, giao thức IP sẽ thực hiện phân đoạn, chia datagram thành các phần nhỏ hơn (gọi là các đoạn) để đảm bảo mỗi đoạn đều có kích thước nhỏ hơn hoặc bằng MTU.

### **❖ Port**

UDP sử dụng các cổng để chuyển hướng dữ liệu đến các tiến trình cụ thể đang chạy trên máy tính. Để thực hiện việc này, UDP xác định đường đi của gói tin dựa trên số hiệu cổng được chỉ định trong phần header của datagram. Các cổng được biểu diễn bằng số 16-bit, với phạm vi từ 0 đến 65535. Chúng được xem như các điểm cuối của kết nối logic và được phân thành ba loại chính:

* Các cổng phổ biến: Từ 0 đến 1023
* Các cổng đã đăng ký: 1024 đến 49151
* Các cổng động/dành riêng 49152 đến 65535

Lưu ý rằng các cổng UDP có thể xử lý nhiều thông điệp đồng thời. Trong một số trường hợp, các dịch vụ TCP và UDP có thể chia sẻ cùng một số hiệu cổng, chẳng hạn như cổng 7 (Echo) hoặc cổng 23 (Telnet). UDP có các cổng thông dụng sau:

|  |  |
| --- | --- |
| Cổng UDP | Mô tả |
| 15 | Netstat-Network Status- Tình trạng mạng |
| 53 | DNS-Domain Name Server |
| 69 | TFTP-Trivial File Transfer Protocol – Giao thức truyền tệp thông thường |
| 137 | NetBIOS NameService |
| 138 | Dịch vụ Datagram NetBIOS |
| 161 | SNMP |

### ❖ **Multicasting**

Multicasting là một phương pháp chuẩn mở cho phép phân phối cùng một thông tin đến nhiều người dùng cùng lúc. Đây là một tính năng quan trọng của giao thức UDP, cho phép truyền thông theo kiểu **một-nhiều**. Ví dụ về ứng dụng của multicasting bao gồm việc gửi tin nhắn hoặc email đến nhiều người nhận, phát sóng đài radio trên Internet, hoặc thực hiện các buổi trình diễn trực tuyến.

## **2.2 Hoạt động của giao thức UDP**

Khi một ứng dụng sử dụng giao thức UDP gửi dữ liệu đến một host khác trên mạng, UDP thêm một header dài 8 byte vào dữ liệu. Header này bao gồm:

* Số hiệu cổng nguồn và cổng đích
* Tổng chiều dài của dữ liệu
* Thông tin kiểm tra lỗi (checksum)

Sau đó, giao thức IP sẽ thêm header riêng của nó vào phía trước mỗi datagram UDP, tạo thành một datagram IP hoàn chỉnh để truyền qua mạng.

## **2.3 Các nhược điểm của giao thức UDP**

* UDP không sử dụng các tín hiệu bắt tay trước khi gửi dữ liệu, điều này có nghĩa là không có quá trình xác nhận giữa bên gửi và bên nhận. Vì lý do này, phía gửi không thể xác định được liệu datagram đã đến đích hay chưa. Do đó, UDP không đảm bảo việc dữ liệu được truyền đến đích một cách thành công và không có cơ chế để kiểm tra hoặc xác nhận sự giao nhận.
* UDP là giao thức phi liên kết và không hỗ trợ việc duy trì các phiên. Vì vậy, UDP không có khả năng sử dụng các chỉ số phiên để quản lý các kết nối giữa các host.
* UDP không đảm bảo độ tin cậy trong việc truyền dữ liệu, tức là không đảm bảo rằng chỉ có một bản sao dữ liệu sẽ đến đích. Khi gửi dữ liệu đến các hệ thống cuối, UDP phân chia dữ liệu thành các đoạn nhỏ và gửi chúng đi. Tuy nhiên, UDP không đảm bảo rằng các đoạn này sẽ đến đích theo đúng thứ tự như khi chúng được tạo ra từ nguồn.
* Kiểm soát luồng: UDP không có cơ chế kiểm soát luồng, điều này có thể dẫn đến tình trạng mất mát dữ liệu nếu ứng dụng gửi quá nhiều dữ liệu một cách nhanh chóng mà không xem xét khả năng xử lý của hệ thống nhận. Kết quả là, một ứng dụng sử dụng UDP được thiết kế kém có thể làm giảm băng thông của mạng, gây tắc nghẽn hoặc làm giảm hiệu suất mạng.

**2.4 Các ưu điểm của UDP**

* UDP là giao thức phi liên kết, nghĩa là không cần thiết lập kết nối giữa bên gửi và bên nhận trước khi truyền dữ liệu. Vì UDP không sử dụng các tín hiệu handshaking để thiết lập kết nối, nên có thể tránh được thời gian trễ, giúp việc truyền tải dữ liệu nhanh chóng hơn.
* UDP hỗ trợ các liên kết **1-1** (point-to-point) và **1-n** (one-to-many), cho phép truyền tải dữ liệu từ một nguồn đến một hoặc nhiều đích mà không cần thiết lập kết nối phức tạp.
* UDP có header dài 8 byte cho mỗi gói tin, điều này giúp giảm thiểu kích thước dữ liệu cần truyền và sử dụng băng thông hiệu quả.

mà nó đã được tạo ra

MTU là viết tắt của Maximum Transmission Unit. MTU là một đặc trưng của tầng

liên kết mô tả số byte dữ liệu tối đa có thể truyền trong một gói tin. Mặt khác, MTU là

gói dữ liệu lớn nhất mà môi trường mạng cho trước có thể truyền. Ví dụ, Ethernet có

MTU cố định là 1500 byte. Trong UDP, nếu kích thước của một datagram lớn hơn

MTU, IP sẽ thực hiện phân đoạn, chia datagram thành các phần nhỏ hơn (các đoạn), vì

vậy mỗi đoạn nhỏ có kích thước nhỏ hơn MTU.

MTU là viết tắt của Maximum Transmission Unit. MTU là một đặc trưng của tầng

liên kết mô tả số byte dữ liệu tối đa có thể truyền trong một gói tin. Mặt khác, MTU là

gói dữ liệu lớn nhất mà môi trường mạng cho trước có thể truyền. Ví dụ, Ethernet có

MTU cố định là 1500 byte. Trong UDP, nếu kích thước của một datagram lớn hơn

MTU, IP sẽ thực hiện phân đoạn, chia datagram thành các phần nhỏ hơn (các đoạn), vì

vậy mỗi đoạn nhỏ có kích thước nhỏ hơn MTU

MTU là viết tắt của Maximum Transmission Unit. MTU là một đặc trưng của tầng

liên kết mô tả số byte dữ liệu tối đa có thể truyền trong một gói tin. Mặt khác, MTU là

gói dữ liệu lớn nhất mà môi trường mạng cho trước có thể truyền. Ví dụ, Ethernet có

MTU cố định là 1500 byte. Trong UDP, nếu kích thước của một datagram lớn hơn

MTU, IP sẽ thực hiện phân đoạn, chia datagram thành các phần nhỏ hơn (các đoạn), vì

vậy mỗi đoạn nhỏ c

quen với một số thuật ngữ. Trong phần dưới đây, chúng ta sẽ định nghĩa một số thuật

ngữ cơ bản có liên quan đến giao thức UDP.

quen với một số thuật ngữ. Trong phần dưới đây, chúng ta sẽ định nghĩa một số thuật

ngữ cơ bản có liên quan đến giao thức UDP.

quen với một số thuật ngữ. Trong phần dưới đây, chúng ta sẽ định nghĩa một số thuật

ngữ cơ bản có liên quan đến giao thức UDP.

quen với một số thuật ngữ. Trong phần dưới đây, chúng ta sẽ định nghĩa một số thuật

ngữ cơ bản có liên quan đến giao thức UDP.

quen với một số thuật ngữ. Trong phần dưới đây, chúng ta sẽ định nghĩa một số thuật

ngữ cơ bản có liên quan đến giao thức UDP

# **3. Lớp DatagramPacket**

Trong Java, một datagram UDP được biểu diễn bởi lớp DatagramPacket: Lớp DatagramPacket trong Java là một lớp cuối (final) kế thừa từ Object, cung cấp các phương thức để làm việc với gói tin UDP. Cụ thể, lớp này hỗ trợ:

1. **Nhận và thiết lập địa chỉ nguồn và đích** từ header IP của gói tin.
2. **Nhận và thiết lập thông tin về cổng nguồn và đích.**
3. **Nhận và thiết lập độ dài dữ liệu** được truyền qua gói tin.

## **3.1 Constructor của DatagramPacket**

Lớp DatagramPacket cung cấp nhiều constructor khác nhau để khởi tạo gói tin, phù hợp với việc gửi hoặc nhận dữ liệu:

### **❖ Gửi dữ liệu**

|  |  |
| --- | --- |
| DatagramPacket(byte[] buf, int length, InetAddress address, int port) | Tạo một gói tin datagram để gửi các gói tin có độ dài được chỉ định đến số cổng được chỉ định trên máy chủ được chỉ định |
| DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port) | Tạo một gói tin datagram để gửi các gói tin có độ dài được chỉ định với độ lệch offset đến số cổng được chỉ định trên máy chủ được chỉ định |
| DatagramPacket(byte[] buf, int offset, int length, SocketAddress address) | Tạo một gói tin datagram để gửi các gói tin có độ dài được chỉ định với độ lệch offset đến số cổng được chỉ định trên máy chủ được chỉ định |
| DatagramPacket(byte[] buf, int length, SocketAddress  address) | Tạo một gói tin datagram để gửi các gói tin có độ dài được chỉ định đến số cổng được chỉ định trên máy chủ được chỉ định |

Mỗi constructor của lớp DatagramPacket được sử dụng để tạo một đối tượng mới, đại diện cho một gói tin UDP có thể được gửi đến một host khác. Dữ liệu trong gói tin được điền đầy từ vùng đệm và có chiều dài cụ thể được chỉ định bởi tham số length byte bắt đầu từ vị trí offset hoặc vị trí 0 nếu offset không được sử dụng.

**Ví dụ: sử dụng constructor:** DatagramPacket(byte[] buf,int offset,int length, InetAddress address, int port)

byte[] data = "HelloUDP".getBytes();

DatagramPacket packet = new DatagramPacket(data, 3, 5, InetAddress.getByName("localhost"), 12345); // Dữ liệu được lấy từ data[3] đến data[7].

### **❖ Nhận dữ liệu**

|  |  |
| --- | --- |
| DatagramPacket(byte[] buf, int length) | Tạo một DatagramPacket để nhận các gói tin có độ dài được chỉ định |
| DatagramPacket(byte[] buf, int offset, int length) | Tạo một DatagramPacket để nhận các gói tin có độ dài được chỉ định, chỉ định một độ lệch trong bộ đệm |

Khi một socket nhận một datagram, phần dữ liệu của gói tin sẽ được lưu trữ trong vùng đệm b. Cách lưu trữ phụ thuộc vào constructor được sử dụng:

1. Constructor đầu tiên (DatagramPacket(byte[] b, int length)):

* Dữ liệu được lưu bắt đầu từ vị trí b[0] và tiếp tục đến khi:
* Toàn bộ gói tin được lưu trữ, hoặc
* Đã lưu đủ length byte dữ liệu.

1. Constructor thứ hai (DatagramPacket(byte[] b, int length, int offset)):

* Dữ liệu được lưu trữ bắt đầu từ vị trí b[offset] và tiếp tục lưu cho đến khi:
* Toàn bộ gói tin được lưu trữ, hoặc
* Đã lưu đủ length byte dữ liệu.
* Yêu cầu: Chiều dài hợp lệ của vùng đệm b phải thỏa mãn điều kiện:
* Chiều dài của b phải nhỏ hơn hoặc bằng b.length-offset.

**Ví dụ: Sử dụng constructor với offset:**

byte[] buffer = new byte[1024];

int offset = 100; // Bắt đầu lưu trữ từ vị trí 100 trong buffer

int length = 500; // Độ dài tối đa của dữ liệu được lưu

DatagramPacket packet = new DatagramPacket(buffer, length, offset);

**Ngoại lệ**

Nếu chiều dài của DatagramPacket vượt quá kích thước vùng đệm (b.length - offset), constructor sẽ ném ra ngoại lệ **IllegalArgumentException**. Đây là một ngoại lệ thuộc loại **RuntimeException**, do đó chương trình không bắt buộc phải xử lý ngoại lệ này bằng các khối try-catch.

**3.2 Các phương thức trong DatagramPacket**

**❖ Các phương thức lấy dữ liệu**

|  |  |
| --- | --- |
| public InetAddress getAddress() | Trả về địa chỉ của máy đã gửi datagram hoặc chỉ của host mà datagram được định địa chỉ |
| public int getPort() | Trả về số hiệu cổng nguồn hoặc đích của datagram |
| public byte[] getData() | Trả vềmảng byte chứa dữ liệu trong datagram |
| public int getOffset() | Trả về chỉ số bắt đầu của mảng byte mà dữ liệu thực sự bắt đầu |
| public int getLength() | Trả về chiều dài của dữ liệu trong gói (tính bằng byte) |
| public SocketAddress getSocketAddress() | trả về một đối tượng SocketAddress, chưa địa chỉ IP và số hiệu cổng của nguồn hoặc đích của gói tin |

**❖ Các phương thức thay đổi dữ liệu**

|  |  |
| --- | --- |
| public void setData(byte[] buf) | đặt dữ liệu cho gói |
| public void setData(byte[] buf,int offset,int length) | đặt dữ liệu, độ dài và độ lệch của gói. |
| public void setAddress([InetAddress](https://docs.oracle.com/javase/8/docs/api/java/net/InetAddress.html) iaddr) | đặt địa chỉ IP của máy mà datagram này đang được tới |
| public void setPort(int iport) | đặt số hiệu cổng đích mà datagram này sẽ được gửi tới. |
| public void setSocketAddress([SocketAddress](https://docs.oracle.com/javase/8/docs/api/java/net/SocketAddress.html) address) | Đặt địa chỉ socket cho gói, bao gồm cả địa chỉ IP và cổng |
| public void setLength(int length) | Đặt chiều dài của dữ liệu trong gói. |

**4. Lớp DatagramSocket**

Trong Java, lớp DatagramSocket thuộc gói java.net được sử dụng để gửi và nhận các gói tin (datagrams) qua giao thức UDP. Lớp này hỗ trợ:

* Tạo và quản lý một kết nối socket để gửi và nhận dữ liệu.
* Gửi gói tin tới địa chỉ cụ thể trên mạng.
* Nhận gói tin từ các địa chỉ khác qua cổng đã chỉ định.
* Đóng socket sau khi hoàn tất việc truyền tải để giải phóng tài nguyên hệ thống.

**4.1 Constructor của lớp DatagramSocket**

|  |  |
| --- | --- |
| DatagramSocket() | Tạo một datagram socket và gán nó cho cổng bất kỳ có sẵn trên máy chủ cục bộ. |
| DatagramSocket(int port) | Tạo một DatagramSocket mới và gán cổng cụ thể cho socket. |
| DatagramSocket(int port, InetAddress laddr) | o một DatagramSocket mới với cổng và địa chỉ IP cụ thể. |
| DatagramSocket(SocketAddress bindaddr) | tạo một socket và liên kết nó với một địa chỉ cụ thể |

**4.2 Phương thức gửi và nhận gói tin**

|  |  |
| --- | --- |
| void send(DatagramPacket p) | Gửi một gói tin DatagramPacket tới địa chỉ và cổng đã được chỉ định. |
| void receive(DatagramPacket p) | Nhận một gói tin và lưu dữ liệu vào DatagramPacket được truyền vào. |

**4.3 Phương thức quản lý kết nối**

|  |  |
| --- | --- |
| void close() | Đóng DatagramSocket và giải phóng các tài nguyên liên quan. |
| boolean isClosed(): | Trả về true nếu DatagramSocket đã bị đóng, false nếu chưa. |
| boolean isBound(): | Trả về true nếu socket đã được liên kết với một địa chỉ cụ thể, false nếu chưa |

**4.4 Phương thức lấy dữ liệu**

|  |  |
| --- | --- |
| int getLocalPort() | Trả về cổng mà socket đang sử dụng. |
| InetAddress getLocalAddress() | Trả về địa chỉ IP của máy local mà socket đang sử dụng. |
| SocketAddress getLocalSocketAddress() | Trả về địa chỉ và cổng mà socket đang được liên kết. |
| int getSoTimeout() | rả về thời gian chờ (timeout) trong mili giây cho socket. Nếu thời gian này trôi qua mà không nhận được gói tin, sẽ xảy ra SocketTimeoutException. |

**4.5 Phương thức cài đặt và cấu hình socket**

|  |  |
| --- | --- |
| void setSoTimeout(int timeout) | Đặt thời gian chờ (timeout) cho socket. Thời gian này được chỉ định bằng mili giây. |
| void setReceiveBufferSize(int size) | Đặt kích thước bộ đệm nhận cho socket. |
| int getReceiveBufferSize() | Trả về kích thước bộ đệm nhận hiện tại của socket |
| void setSendBufferSize(int size) | Đặt kích thước bộ đệm gửi cho socket. |
| int getSendBufferSize() | Trả về kích thước bộ đệm gửi hiện tại của socket. |
| void setTrafficClass(int tc) | Đặt lớp lưu lượng cho socket (tùy thuộc vào hệ điều hành hỗ trợ). |
| int getTrafficClass() | Trả về lớp lưu lượng hiện tại của socket. |

**4.6 Xử lý các ngoại lệ**

|  |  |
| --- | --- |
| void setReuseAddress(boolean on) | Đặt liệu địa chỉ có thể được sử dụng lại hay không, hữu ích khi bạn muốn nhanh chóng tái sử dụng một địa chỉ cổng sau khi socket được đóng. |
| boolean getReuseAddress() | Trả về trạng thái của cài đặt reuseAddress. |

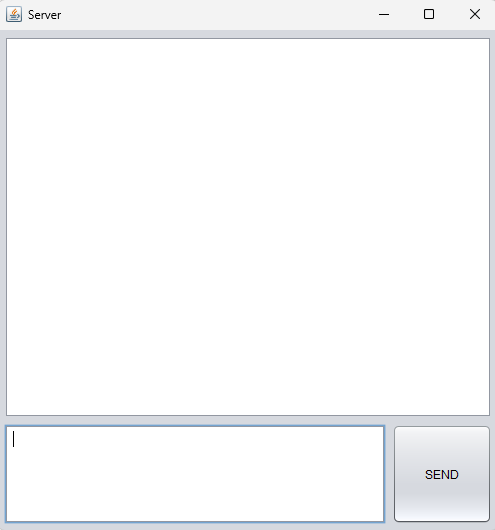
**4.7 Các phương thức khác**

|  |  |
| --- | --- |
| void joinGroup(SocketAddress mcastAddr) | Tham gia một nhóm multicast. |
| void leaveGroup(SocketAddress mcastAddr) | Rời khỏi một nhóm multicast. |
| void joinGroup(InetAddress group, NetworkInterface netIf) | Tham gia một nhóm multicast trên một giao diện mạng cụ thể. |
| void leaveGroup(InetAddress group, NetworkInterface netIf) | Rời khỏi một nhóm multicast trên một giao diện mạng cụ thể. |

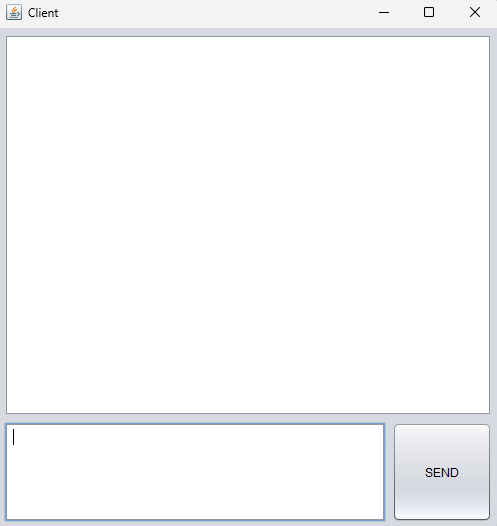
**5. Lập trình Client-Server sử dụng UDP socket**

**5.1 Chat giữa Client-Server**

**Xây dựng giao diện JFrame từ TextArea , TextPanel , Button**



Hình 5.1.1 Giao diện Server



Hình 5.1.2 Giao diện Client

Click Button SEND sẽ đưa nội dung từ TextPanel lên TextArea và gửi dữ liệu

// Lấy Thời Gian Hiện Tại

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss");

String currentTime = LocalDateTime.now().format(formatter);

// Đưa nội dung từ TextPanel lên TextArea

jTextArea1.append("[ Server ] - " + currentTime +"\n");

jTextArea1.append(jTextPane1.getText() + "\n" + "\n");

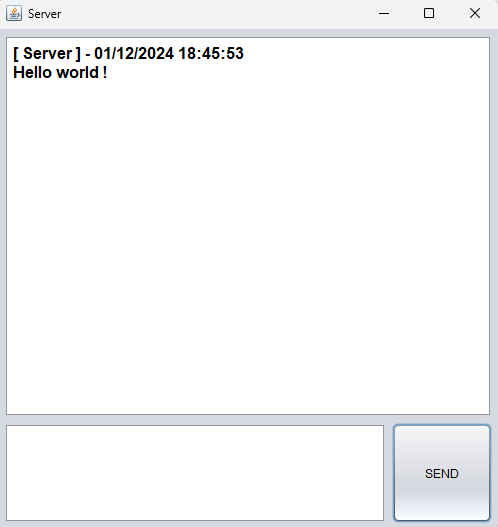
// Gửi dữ liệu

byte[] data = jTextPane1.getText().getBytes();

DatagramPacket packet = new DatagramPacket(data, data.length, IP, port\_client);

socket.send(packet);

jTextPane1.setText("");



Hình 5.1.3 Server gửi nội dung “Hello world !”

Nhận chuỗi dữ liệu được gửi và đưa lên TextArea

byte[] buffer = new byte[1024];

DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

socket.receive(packet);

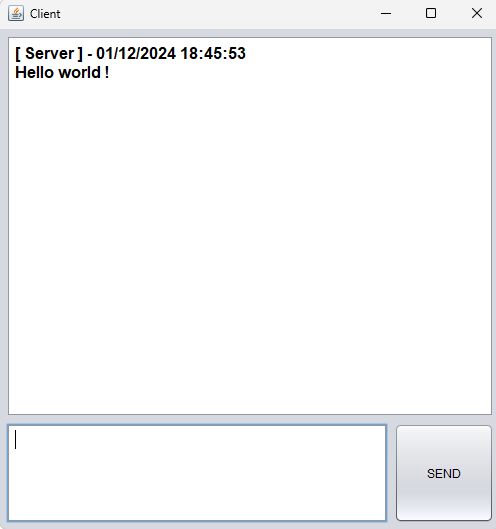
String message = new String(packet.getData(), 0, packet.getLength());

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss");

String currentTime = LocalDateTime.now().format(formatter);

jTextArea1.append("[ Server ] - " + currentTime +"\n");

jTextArea1.append(message + "\n" + "\n");

****

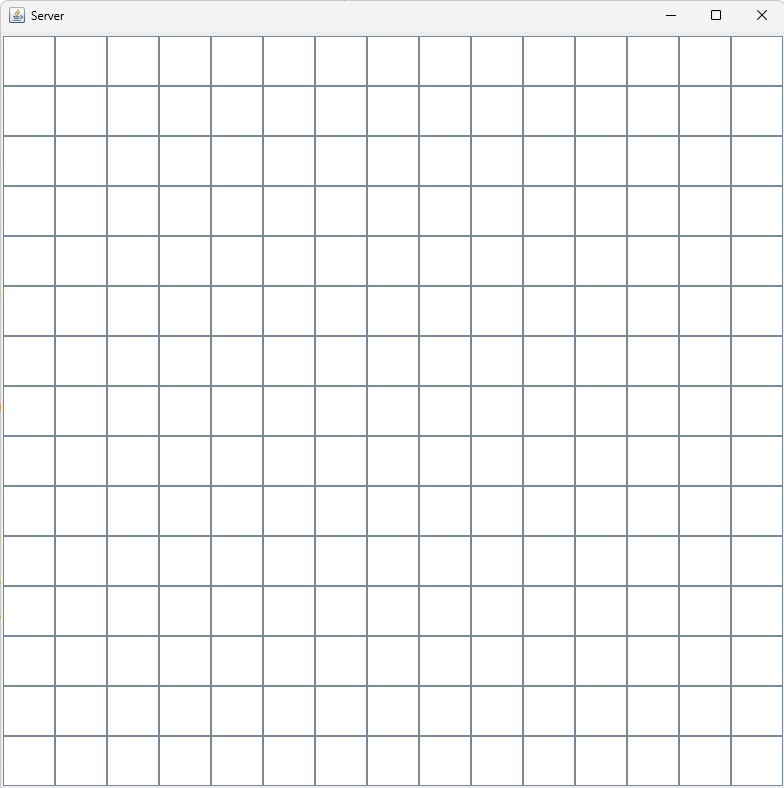
Hình 5.1.4 Client Nhận nội dung “Hello world”

**5.2 Lập trình game cờ Caro**

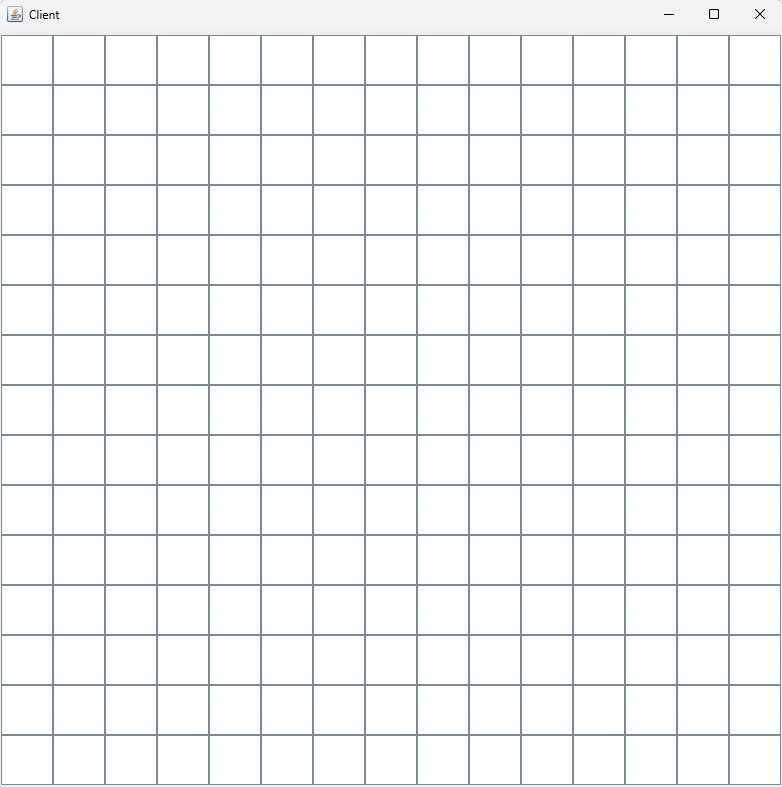
Sử dụng mảng 2 chiều kích thước n x n kiều JButton để làm bàng cờ cho Server và Client

Mỗi ô trên bàn cờ là sẽ là một đối tượng JButton

JButton[][] buttons = new JButton[SIZE][SIZE];



Hình 2.1. Bảng cờ của Server



Hình 2.2. Bảng cờ của Client

Server và Client đều được xây dựng phương pháp gửi dữ liệu như sau

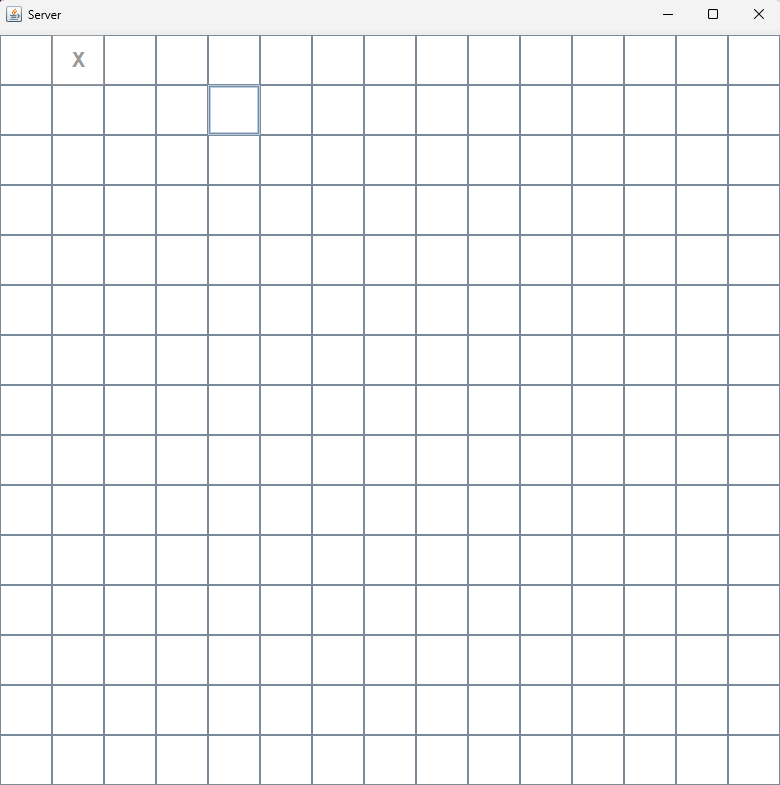
String str = x + "-" + y ;

byte[] data = str.getBytes();

DatagramPacket packet = new DatagramPacket(data, data.length, IP, port);

socket.send(packet);

Ví dụ khi Server đánh X vào ô có tọa độ (0,1) tương ứng x = 0 và y = 0 sẽ gửi chuỗi “0-1” tới Client



Hình 2.3 Server đánh X vào ô (0,1)

Nhận và xử lí chuỗi để lấy các dữ liệu cần thiết từ gói tin để sử dụng cho chương trình

byte[] buffer = new byte[1024];

DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

socket.receive(packet);

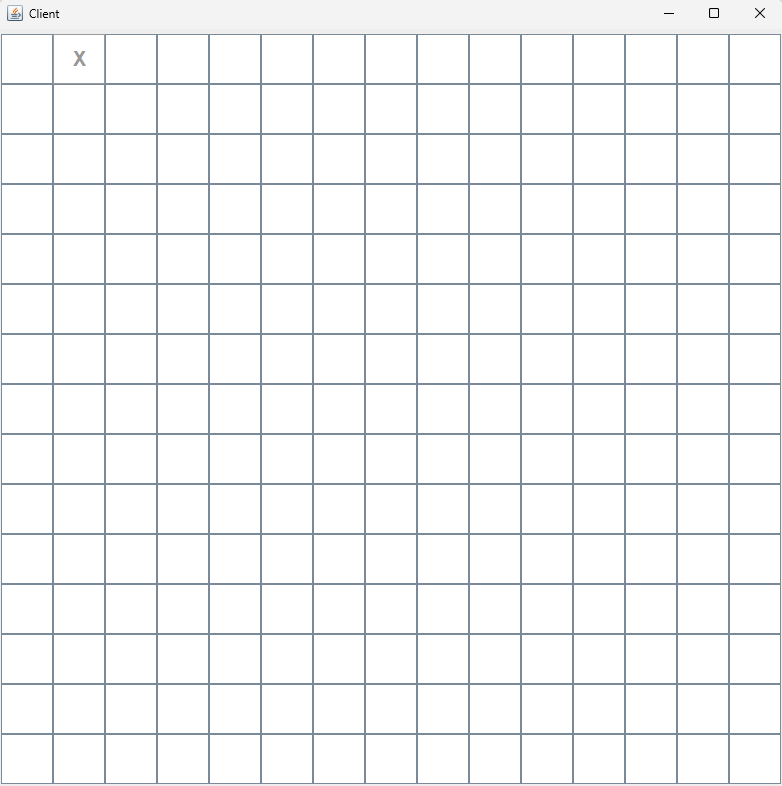
String message = new String(packet.getData(), 0, packet.getLength());

String[] parts = message.split("-");

int x = Integer.parseInt(parts[0]);

int y = Integer.parseInt(parts[1]);

Từ đoạn code trên , ta có x = 0 và y = 1 , cập nhật X tại ô (0,1) trên bảng cờ của Client



Hình 2.4 Client cập nhật X tại ô (0,1)

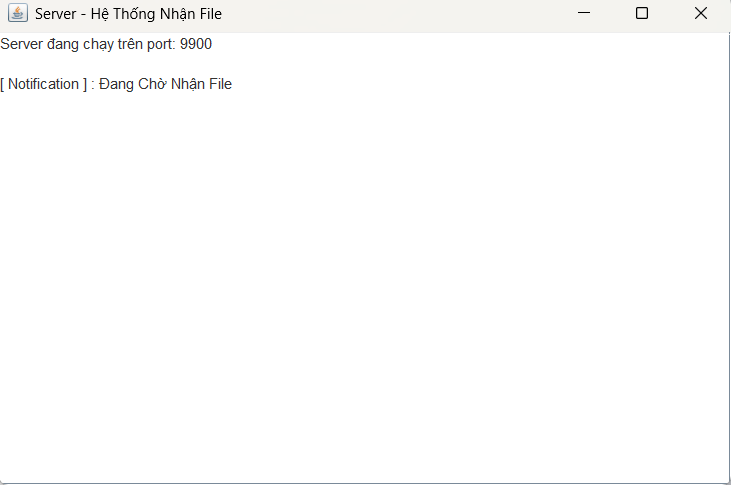
**5.3 Phần Mềm Nộp Bài Thi**

DatagramSocket và DatagramPacket gửi các gói dữ liệu dạng bytes vậy nên ta cũng có thể gửi các tệp tin sau khi tệp tin được chia thành các gói byte nhỏ

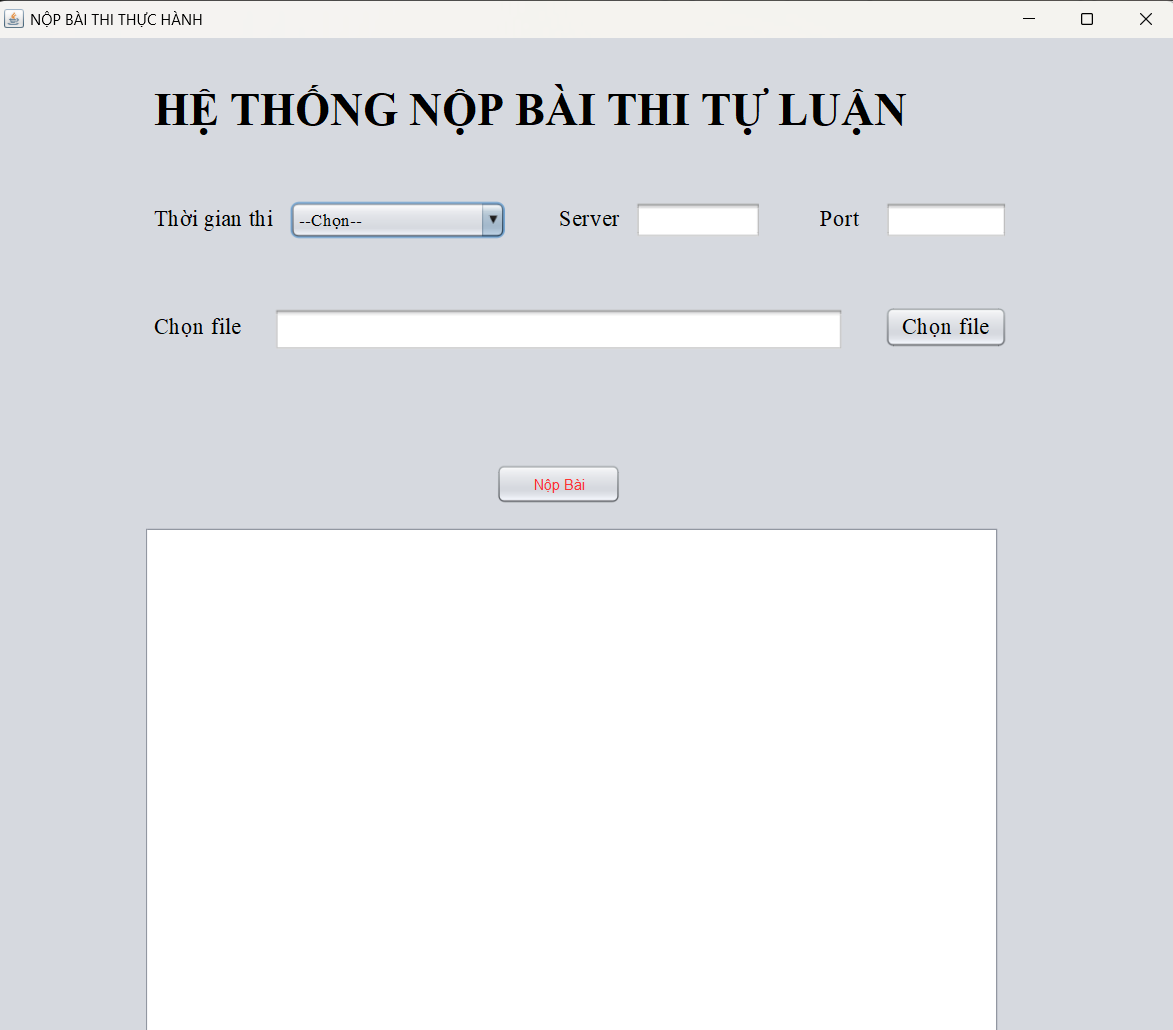
\* Xây dựng giao diện JFrame phía Client từ TextArea, TextFeild, Button, Label, Combobox, ScrollPane.

\* Xây dựng giao diện JFrame phía Server từ TextArea, ScrollPane.

Ta có 2 bảng giao diện phía Cilent và phía Server:



Hình 8.1. Giao diện Server



8.2. Giao diện Client

- Người dùng cần nhập Server, Port, chọn thời gian thi, click vào button Chọn file để chọn file muốn gửi, kích thước file không quá 65536KB

- Sau đó click button Nộp Bài để gửi file sẽ đưa các nội dung lên TextArea để cho người dùng biết

\* Quy trình gửi và nhận giữa Client và Server

//Tạo một đối tượng DatagramSocket với số cổng bất kỳ

DatagramSocket socket = new DatagramSocket();

InetAddress address = InetAddress.getByName(serverAddress);

//Khai báo bộ đệm nhập/xuất mảng kiểu byte, gói tin gửi/nhận là đối tượng DatagramPacket và thực hiện gửi/nhận thông qua phương thức send()/receive()

// Gửi thời gian thi

byte[] timeData = selectedTime.getBytes();

DatagramPacket timePacket = new DatagramPacket(timeData, timeData.length, address, port);

socket.send(timePacket);

txtLog.append("Ca thi: " + selectedTime + "\n");

// Gửi thông tin file

String fileName = file.getName();

byte[] fileNameBytes = fileName.getBytes();

DatagramPacket fileNamePacket = new DatagramPacket(fileNameBytes, fileNameBytes.length, address, port);

socket.send(fileNamePacket);

txtLog.append("Nộp bài với file: " + fileName + "\n");

// Gửi kích thước file

long fileSize = file.length();

ByteArrayOutputStream baos = new ByteArrayOutputStream();

DataOutputStream dos = new DataOutputStream(baos);

dos.writeLong(fileSize);

byte[] fileSizeBytes = baos.toByteArray();

DatagramPacket fileSizePacket = new DatagramPacket(fileSizeBytes, fileSizeBytes.length, address, port);

socket.send(fileSizePacket);

txtLog.append("Kích thước file: " + fileSize + " bytes\n");

// Gửi dữ liệu file

String timestamp = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());//Lấy thời gian hiện tại

FileInputStream fis = new FileInputStream(file);

byte[] buffer = new byte[4096];

int read;

int totalSent = 0;

while ((read = fis.read(buffer)) != -1) {//dòng này đọc một phần dữ liệu từ tệp và lưu vào buffer, trả về số byte đã đọc hoặc là -1 nếu đã đọc hết tệp

DatagramPacket fileDataPacket = new DatagramPacket(buffer, read, address, port);

socket.send(fileDataPacket);

totalSent += read;//Cập nhật tổng số byte đã gửi

txtLog.append("Đã gửi: " + read + " bytes\n");//Hiển thị số byte đã gửi trong lần gửi này lên log.

}

textArea.append("[ Success ] : Nhận File Thành Công !\n\n");

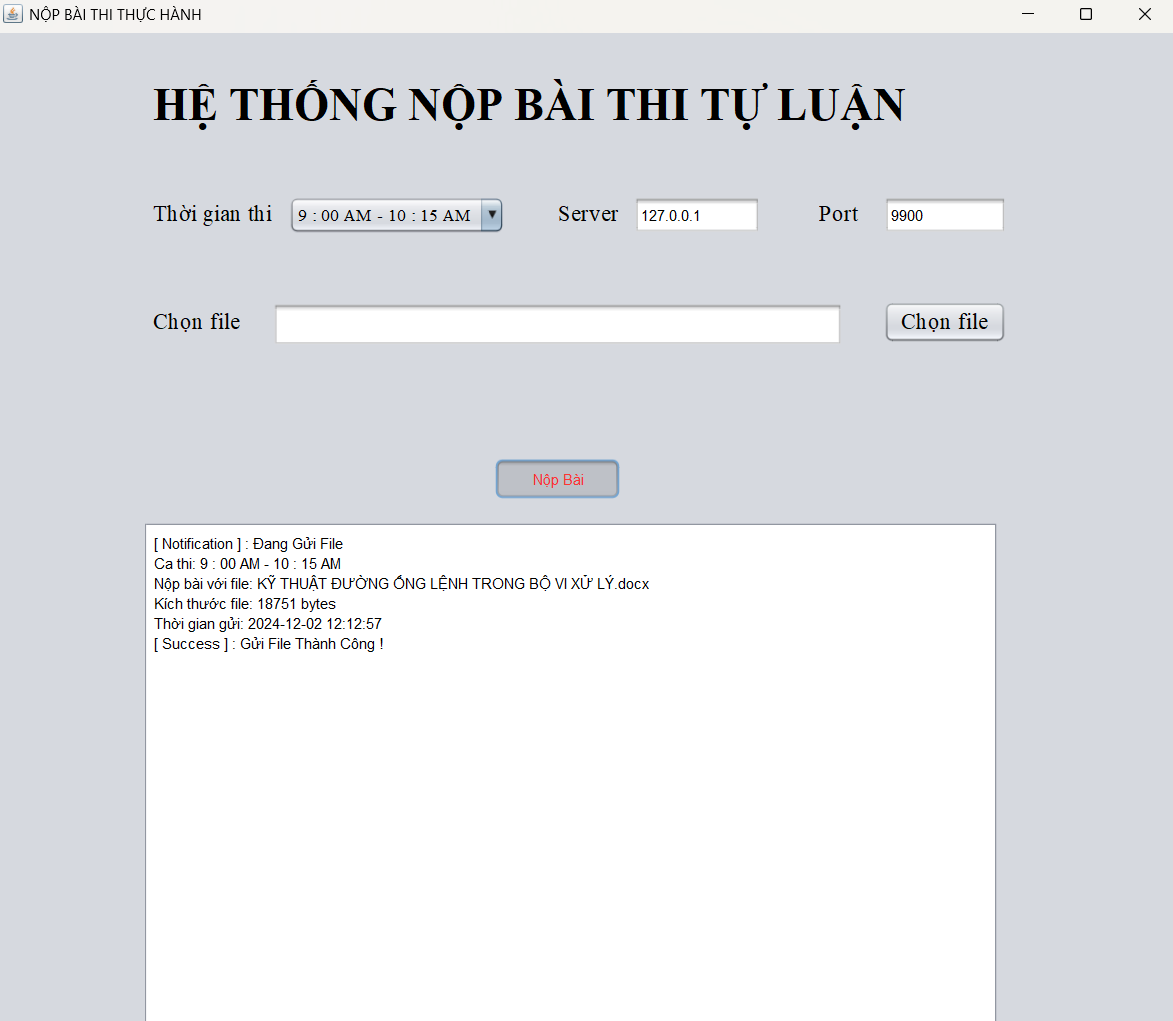
//Sau khi gửi file thành công, xóa đường dẫn file

txtFile.setText(“”);

fis.close();

socket.close();

//Đóng socket, kết thúc hoặc quay về chờ gói tin tiếp theo



Hình 8.1.1. Client sau khi gửi file thành công

- Phía server sẽ nhận dữ liệu được gửi và đưa lên TextArea các thông tin đã nhận

//Tạo một đối tượng DatagramSocket với số cổng xác định

private static int port = 9900;

DatagramSocket serverSocket = new DatagramSocket(port);

//Khai báo bộ đệm nhập/xuất mảng kiểu byte, gói tin gửi/nhận là đối tượng DatagramPacket và thực hiện gửi/nhận thông qua phương thức send()/receive()

// Nhận thời gian thi

DatagramPacket timePacket = new DatagramPacket(buffer, buffer.length);

serverSocket.receive(timePacket);

String examTime = new String(timePacket.getData(), 0, timePacket.getLength());

textArea.append("Ca thi: " + examTime + "\n");

String examTime\_Folder = examTime.replace(":","H");

File directory = new File(examTime\_Folder);

if (!directory.exists())//Kiểm tra xem thư mục có tồn tại hay không

{

directory.mkdir();//Tạo ra thư mục mới với tên được chỉ định

}

// Nhận tên file

DatagramPacket fileNamePacket = new DatagramPacket(buffer, buffer.length);

serverSocket.receive(fileNamePacket);

String fileName = new String(fileNamePacket.getData(), 0, fileNamePacket.getLength());

textArea.append("Tên file: " + fileName + "\n");

// Nhận kích thước file

DatagramPacket fileSizePacket = new DatagramPacket(buffer, buffer.length);

serverSocket.receive(fileSizePacket);

ByteArrayInputStream bais = new ByteArrayInputStream(fileSizePacket.getData());

DataInputStream dis = new DataInputStream(bais);

long fileSize = dis.readLong();

textArea.append("Kích thước file: " + fileSize + " bytes\n");

// Nhận dữ liệu file

FileOutputStream fos = new FileOutputStream("received\_" + fileName);

long receivedBytes = 0;

while (receivedBytes < fileSize) {//Khi số byte vẫn còn nhỏ hơn kích thuớc file thì vòng while lặp đến khi >= thì dừng lặp

DatagramPacket fileDataPacket = new DatagramPacket(buffer, buffer.length);

serverSocket.receive(fileDataPacket);

fos.write(fileDataPacket.getData(), 0, fileDataPacket.getLength());//Ghi dữ liệu vào tệp

receivedBytes += fileDataPacket.getLength();

textArea.append("Đã nhận: " + receivedBytes + " bytes\n");//Hiển thị tổng số byte đã nhận được đến thời điểm hiện tại lên TextArea

}

String timestamp = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());

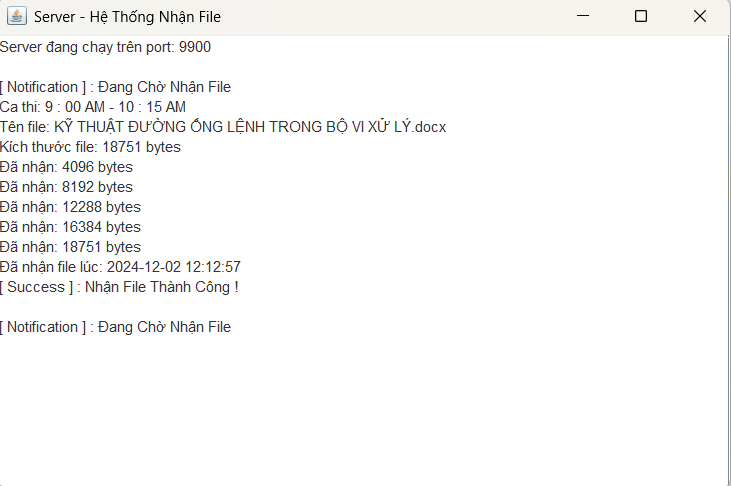
textArea.append("Đã nhận file lúc: " + timestamp + "\n\n");

textArea.append("[ Success ] : Nhận File Thành Công !\n\n");

fos.close();

serverSocket.close();

//Đóng socket, kết thúc hoặc quay về chờ gói tin tiếp theo



Hình 8.2.1. Server nhận file sau khi Client gửi thành công

- Sau khi nhận file thành công sẽ thực hiện tạo và khởi chạy một luồng mới để chạy phương thức receiveData()

new Thread(() -> receiveData()).start();

public final class DatagramPacket extends Object

Lớp này cung cấp các phương thức để nhận và thiết lập các địa chỉ nguồn, đích từ

header IP, nhận và thiết lập các thông tin về cổng nguồn và đích, nhận và thiết lập độ

dài dữ liệu. Các trường thông tin còn lại không thể truy nhập được từ mã Java thuần

túy.

DatagramPacket sử dụng các constructor khác nhau tùy thuộc vào gói tin được sử

dụng để gửi hay nhận dữ liệu