



---

# BÁO CÁO

## ĐỒ ÁN THỰC HÀNH MẠNG MÁY TÍNH

### CHỦ ĐỀ ĐỒ ÁN: REMOTE DESKTOP

---

**Sinh viên:** Trần Ngọc Đại - 22120045

Nguyễn Hưng - 22120122

Huỳnh Thanh Tường - 22120411

**Lớp:** 22TNT1TN

**Giảng viên hướng dẫn:** Thầy Đỗ Hoàng Cường

*Ho Chi Minh, December 2023.*

# Mục lục

<b>Chương 1 Giới thiệu .....</b>	<b>1</b>
1.1 Thông tin nhóm .....	1
1.1.1 Thông tin thành viên .....	1
1.1.2 Phân chia và tiến độ thực hiện nhiệm vụ .....	1
1.2 Tổng quan về đề tài .....	1
1.3 Sản phẩm .....	1
1.1.1 Link video demo .....	1
1.1.2 Link mã nguồn .....	1
<b>Chương 2 Chương trình và mã nguồn .....</b>	<b>2</b>
2.1 Thư viện hỗ trợ .....	2
2.1.1 OpenCV .....	2
2.1.2 SFML .....	2
2.1.3 Windows API .....	2
2.2 Tổng quan về mô hình Client - Server .....	3
2.2.1 Client .....	3
2.2.2 Server .....	4
2.3 Chi tiết mã nguồn .....	4
2.3.1 Socket .....	4
2.3.1.1 Client .....	4
2.3.1.2 Server .....	6
2.3.2 Sự kiện .....	7
2.3.2.1 Màn hình .....	7
2.3.2.2 Chuột .....	11
2.3.2.3 Bàn phím .....	17
2.3.3 Giao diện .....	20
2.3.3.1 Button .....	20
2.3.3.2 Event .....	21
2.3.3.3 Slide 1 .....	22
2.3.3.4 Slide 2 .....	24
<b>Chương 3 Hướng dẫn sử dụng .....</b>	<b>27</b>
3.1 Cài đặt thư viện .....	27
3.1.1 OpenCV2 .....	27
3.1.2 SFML .....	30
3.1.3 Lưu ý về cài đặt môi trường .....	30
3.2 Sử dụng chương trình .....	31
3.2.1 Lấy địa chỉ IP của server .....	31
3.2.2 Tại server .....	31
3.2.3 Tại Client .....	32
<b>Tài liệu tham khảo .....</b>	<b>35</b>

## Chương 1

# Giới thiệu

### 1.1 Thông tin nhóm

#### 1.1.1 Thông tin thành viên

**Sinh viên:** Trần Ngọc Đại - 22120045

Nguyễn Hưng - 22120122

Huỳnh Thanh Tường - 22120411

**Lớp:** 22TNT1TN

#### 1.1.2 Phân chia và tiến độ thực hiện nhiệm vụ

Thành viên	Nhiệm vụ	Hoàn thành
Huỳnh Thanh Tường	Gửi, nhận, hiển thị ảnh, tạo class Button, class Event	100%
Trần Ngọc Đại	Gửi, nhận, hiển thị sự kiện chuột và Slide 2	100%
Nguyễn Hưng	Gửi, nhận, hiển thị sự kiện bàn phím và Slide 1	100%

Bảng 1.1 – Bảng phân chia nhiệm vụ

### 1.2 Tổng quan về đề tài

Đồ án Remote desktop bằng C++ là một dự án nhằm tạo ra một ứng dụng cho phép người dùng điều khiển một máy tính từ xa thông qua mạng. Dự án này sử dụng mô hình Client - Server. Máy chủ được điều khiển, nhận và thực hiện các yêu cầu từ máy khách. Máy khách là giao diện người dùng cho người dùng điều khiển máy tính từ xa. Mục tiêu của dự án là cung cấp khả năng điều khiển máy tính từ xa qua mạng và thực hiện các thao tác như điều khiển bàn phím, chuột và truyền tập tin giữa hai máy tính.

### 1.3 Sản phẩm

#### 1.1.1 Link video demo

<https://youtu.be/J82dbCpMdWE>

#### 1.1.2 Link mã nguồn

<https://github.com/tuongkhtn/Remote-Desktop.git>

## Chương 2

# Chương trình và mã nguồn

### 2.1 Thư viện hỗ trợ

#### 2.1.1 OpenCV

OpenCV (Open Source Computer Vision Library) là một thư viện mã nguồn mở được sử dụng trong lập trình C++ để xử lý ảnh và thị giác máy tính. Nó cung cấp các công cụ và thuật toán mạnh mẽ để làm việc với ảnh và video, bao gồm xử lý, phân tích, và hiển thị hình ảnh.

Với OpenCV trong C++, bạn có thể thực hiện các chức năng như lọc nhiễu, biến đổi màu sắc, phát hiện cạnh, phát hiện đối tượng, nhận dạng khuôn mặt, và nhiều hơn nữa. Thư viện này cho phép bạn xử lý hình ảnh từ các nguồn khác nhau, như webcam, tệp tin ảnh và video.

OpenCV cung cấp một API dễ sử dụng và hỗ trợ cho nhiều nền tảng và thiết bị khác nhau. Nó là một công cụ quan trọng cho các ứng dụng thị giác máy tính, xử lý ảnh y tế, nhận dạng đối tượng và nhiều lĩnh vực khác liên quan đến xử lý ảnh và video trong lập trình C++.

#### 2.1.2 SFML

SFML (Simple and Fast Multimedia Library) là một thư viện đa nền tảng được sử dụng trong lập trình C++ để xây dựng các ứng dụng đồ họa và đa phương tiện. Nó cung cấp các công cụ và chức năng để vẽ đồ họa, xử lý âm thanh, xử lý cửa sổ và xử lý sự kiện, giúp người lập trình dễ dàng tạo ra các ứng dụng đồ họa và trò chơi trực quan và hiệu suất cao trên nền tảng Windows, macOS và Linux.

#### 2.1.3 Windows API

Windows API (Application Programming Interface) là một tập hợp các giao diện lập trình ứng dụng được cung cấp bởi hệ điều hành Windows. Nó cung cấp các chức năng và dịch vụ cho các ứng dụng Windows để tương tác với hệ điều hành, giao tiếp với phần cứng và thực hiện các tác vụ khác nhau. Ứng dụng sử dụng một số hàm từ Windows API để hỗ trợ xử lý các chức năng lấy thông tin sự kiện chuột và bàn phím, cũng như giao tiếp với hệ điều hành để mô phỏng sự kiện trên máy chủ.

## 2.2 Tổng quan về mô hình Client - Server

### 2.2.1 Client

Trong chương trình của client, ta kết nối với sever thông qua socket để truyền các dữ liệu về màn hình, chuột và bàn phím. Class **Client** được cài đặt với các thuộc tính và phương thức sau:

Client	
screenClientSocket: SOCKET	
mouseClientSocket: SOCKET	
keyboardClientSocket: SOCKET	
pclIndex: int	
connectPars: vector<std::pair<char*, int>>	
serverAddr: SOCKADDR_IN	
setPclIndex(index: int): void	
getPclIndex(index: int): int	
setConnectPars(connectPars: vector<pair<char*, int>> ): void	
getConnectPars(): vector<std::pair<char*, int>>	
getServerAddr(): SOCKADDR_IN	
initSocket(socket: SOCKET): void	
connectSocketToServer(socket: SOCKET, serverAddr:SOCKADDR_IN, connectPars:vector<pair<char*, int>>, index: int): void	
initClient(): void	
connectClientToServer(): void	
releaseClient(): void	
run(): void	
end(): void	

Hình 2.1 – Sơ đồ class **Client**

- Ba socket tương ứng với 3 stream dùng để truyền dữ liệu về màn hình, chuột và bàn phím.
- Các phương thức **set**, **get** cho các thuộc tính tương ứng được cài đặt như thông thường:
- Các phương thức **init**, **connect**, **release** thực hiện việc tạo sự liên kết với server phục vụ cho việc truyền nhận dữ liệu. Ta sẽ lần lượt tạo, kết nối và đóng 3 socket ứng với các nhiệm vụ đã trình bày (Cú pháp cụ thể cho một socket sẽ

được trình bày tại [2.3.1](#))

```
void Client::initClient() {
    this->initSocket(this->screenClientSocket);
    this->initSocket(this->mouseClientSocket);
    this->initSocket(this->keyboardClientSocket);
}

void Client::connectClientToServer() {
    this->connectSocketToServer(this->screenClientSocket, this->serverAddr, this->connectPars, this->pcIndex);
    this->connectSocketToServer(this->mouseClientSocket, this->serverAddr, this->connectPars, this->pcIndex);
    this->connectSocketToServer(this->keyboardClientSocket, this->serverAddr, this->connectPars, this->pcIndex);
}

void Client::releaseClient() {
    this->releaseSocket(this->screenClientSocket);
    this->releaseSocket(this->mouseClientSocket);
    this->releaseSocket(this->keyboardClientSocket);
}
```

Hình 2.2 – Source code: Method **init**, **connect**, **release**.

- **run**, **end** thực thi toàn bộ tiến trình truyền và nhận dữ liệu giữa client và server. Trong hàm run, tạo ra 3 biến ứng với màn hình, chuột, và bàn phím, sau đó lần lượt gọi phương thức thread của chúng để chạy 3 luồng truyền nhận độc lập (hoạt động của từng luồng sẽ được trình bày rõ tại [2.3.2](#)). Trong hàm end, ta lần lượt đóng các stream truyền dữ liệu.

```
void Client::run() {
    this->initClient();
    this->connectClientToServer();

    Mouse mouse;
    mouse.threadMouse(this->mouseClientSocket);

    Keyboard keyboard;
    keyboard.threadKeyboard(this->keyboardClientSocket);

    Image image;
    image.threadImage(this->screenClientSocket);
}

void Client::end() {
    releaseSocket(this->screenClientSocket);
    releaseSocket(this->mouseClientSocket);
    releaseSocket(this->keyboardClientSocket);
    cv::destroyAllWindows();
}
```

Hình 2.3 – Source code: Method **run**, **end**.

## 2.2.2 Server

Trong chương trình của Server, ta chấp nhận kết nối của client để nhận sự kiện về màn hình, chuột và bàn phím để xử lý. Ở đây chúng ta sẽ tạo 3 socket riêng để lắng nghe kết nối từ và tạo các luồng để:

- Gửi dữ liệu hình ảnh từ màn hình máy chủ tới máy khách.
- Nhận các sự kiện chuột từ máy khách và cập nhật trạng thái chuột trên máy chủ.
- Nhận các sự kiện bàn phím từ máy khách và cập nhật trạng thái bàn phím trên máy chủ.

## 2.3 Chi tiết mã nguồn

### 2.3.1 Socket

#### 2.3.1.1 Client

- *Khởi tạo Socket:*

- Khai báo một biến có kiểu `WSADATA` để bắt đầu sử dụng **Winsock** và kiểm tra đã khởi tạo thành công hay chưa.
- Khởi tạo một socket sử dụng địa chỉ IPv4 (`AF_INET`), và kiểu `SOCK_STREAM` dùng cho TCP. Kiểm tra việc khởi tạo socket.

```
void Client::initSocket(SOCKET& sock) {
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        std::cout << "Failed." << std::endl;
        WSACleanup();
    }

    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock == INVALID_SOCKET) {
        std::cout << "Failed to create client socket." << std::endl;
        WSACleanup();
    }
}
```

Hình 2.4 – Source code: Method **initSocket**.

- *Kết nối Client đến Server thông qua Socket:*

- Lấy địa chỉ ip và port, và thiết lập cấu trúc địa chỉ server (`SOCKADDR_IN`).
- Kết nối với server thông qua hàm `connect` và kiểm tra có kết nối được hay chưa.

```
void Client::connectSocketToServer(SOCKET& sock, SOCKADDR_IN& serverAddr,
    std::vector<std::pair<char*, int>> connectPars, int index) {
    char* ip = connectPars[index].first;
    int port = connectPars[index].second;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(port);

    if (inet_pton(AF_INET, ip, &serverAddr.sin_addr) != 1) {
        std::cout << "Failed to convert IP." << std::endl;
        closesocket(sock);
        WSACleanup();
    }

    if (connect(sock, (sockaddr*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) {
        std::cout << "Failed to connect server." << std::endl;
        closesocket(sock);
        WSACleanup();
    }
}
```

Hình 2.5 – Source code: Method **connectSocketToServer**.

- *Đóng Socket:*

```
void Client::releaseSocket(SOCKET& sock) {  
    closesocket(sock);  
    WSACleanup();  
}
```

Hình 2.6 – Source code: Method **releaseSocket**.

### 2.3.1.2 Server

- *Khởi tạo Winsock:*

- Sử dụng hàm `WSAStartup` để bắt đầu sử dụng thư viện Winsock.
- In ra thông báo lỗi nếu quá trình khởi tạo thất bại.

- *Tạo Socket:*

- Sử dụng hàm `socket` để tạo một socket với loại là **`SOCK_STREAM`** (TCP).
- Nếu việc tạo socket thất bại, đóng socket, in thông báo lỗi, và dọn dẹp Winsock.

- *Buộc socket:*

- Thiết lập cấu trúc địa chỉ máy chủ **`SOCKADDR_IN`** với địa chỉ IP và cổng của máy chủ.
- Sử dụng hàm `bind` để buộc socket với địa chỉ đã thiết lập.
- Nếu quá trình buộc thất bại, đóng socket, in thông báo lỗi, và dọn dẹp Winsock.

- *Lắng nghe kết nối đến:*

- Sử dụng hàm **`listen`** để đưa socket vào trạng thái lắng nghe kết nối đến.
- Nếu quá trình lắng nghe thất bại, đóng socket, in thông báo lỗi, và dọn dẹp Winsock.



```
// Init server
void initServer(SOCKET& serverSocket) {
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        std::cerr << "Failed to initialize Winsock" << std::endl;
    }

    serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket == INVALID_SOCKET) {
        std::cerr << "Failed to create server socket" << std::endl;
        closesocket(serverSocket);
        WSACleanup();
    }

    SOCKADDR_IN serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(PORT);

    if (bind(serverSocket, (const sockaddr*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) {
        std::cout << "Failed to bind" << std::endl;
        closesocket(serverSocket);
        WSACleanup();
    }

    if (listen(serverSocket, 5) == SOCKET_ERROR) {
        std::cout << "Failed to listen" << std::endl;
        closesocket(serverSocket);
        WSACleanup();
    }
}
```

Hình 2.7 – Source code: **initServer**

- *Đóng socket:*

```
// Release socket
void releaseSocket(SOCKET& socket) {
    closesocket(socket);
    WSACleanup();
}
```

Hình 2.8 – Source code: Release socket

## 2.3.2 Sự kiện

### 2.3.2.1 Màn hình

#### *Ý tưởng:*

- **Server:** Sử dụng luồng để thực hiện nhiệm vụ:
  - **SendImageData:** Gửi dữ liệu hình ảnh từ máy server sang máy client.
- **Client:** Sử dụng luồng để thực hiện nhiệm vụ:
  - **receiveImages:** Nhận dữ liệu từ server và hiển thị ảnh từ dữ liệu đó.

**Cài đặt hàm:**

- Định nghĩa class **Image** với các phương thức constructor, **receiveImages**, **threadImage**:

A screenshot of a code editor showing the source code for the `Image` class. The code is written in C++ and includes a constructor, two public methods, and a closing brace. The code is as follows:

```
class Image {  
public:  
    Image();  
  
    void receiveImages(SOCKET clientSocket);  
    void threadImage(SOCKET clientSocket);  
};
```

Hình 2.9 – Source code: Class **Image**

**•Server:****– cv::Mat CaptureScreen()**

- \* Hàm bắt đầu bằng việc lấy thông tin về màn hình sử dụng hàm **GetDC** và các hàm **GetSystemMetrics** để xác định kích thước và vị trí của màn hình ảo.
- \* Sau đó, nó tạo một bitmap tương thích với màn hình sử dụng **CreateCompatibleBitmap** và một bối cảnh tương thích với màn hình sử dụng **CreateCompatibleDC**.
- \* Sử dụng **BitBlt**, nó chụp ảnh từ màn hình vào bitmap đã tạo.
- \* Tiếp theo, nó xây dựng một cấu trúc **BITMAPINFOHEADER** để đọc dữ liệu hình ảnh và tạo một ma trận OpenCV với định dạng **CV\_8UC4** (RGBA).
- \* Sử dụng **GetDIBits**, dữ liệu ảnh được sao chép từ bitmap vào ma trận OpenCV.
- \* Cuối cùng, nó giải phóng các tài nguyên đã tạo bằng cách sử dụng các hàm **DeleteObject**, **DeleteDC**, và **ReleaseDC**.

```
// Get screen of controlled machine
cv::Mat CaptureScreen() {
    HDC hScreen = GetDC(NULL);
    int screenWidth = GetSystemMetrics(SM_CXVIRTUALSCREEN);
    int screenHeight = GetSystemMetrics(SM_CYVIRTUALSCREEN);
    int screenLeft = GetSystemMetrics(SM_XVIRTUALSCREEN);
    int screenTop = GetSystemMetrics(SM_YVIRTUALSCREEN);

    HBITMAP hBitmap = CreateCompatibleBitmap(hScreen, screenWidth, screenHeight);
    HDC hDC = CreateCompatibleDC(hScreen);
    SelectObject(hDC, hBitmap);
    BitBlt(hDC, 0, 0, screenWidth, screenHeight, hScreen, screenLeft, screenTop, SRCCOPY);

    BITMAPINFOHEADER bi;
    bi.biSize = sizeof(BITMAPINFOHEADER);
    bi.biWidth = screenWidth;
    bi.biHeight = -screenHeight;
    bi.biPlanes = 1;
    bi.biBitCount = 32;
    bi.biSizeImage = 0;
    bi.biCompression = 0;
    bi.biXPelsPerMeter = 0;
    bi.biYPelsPerMeter = 0;
    bi.biClrUsed = 0;
    bi.biClrImportant = 0;

    cv::Mat frame(screenHeight, screenWidth, CV_8UC4); // RGBA
    GetDIBits(hDC, hBitmap, 0, screenHeight, frame.data, (BITMAPINFO*)&bi, DIB_RGB_COLORS);
    cv::cvtColor(frame, frame, cv::COLOR_BGRA2BGR);

    DeleteObject(hBitmap);
    DeleteDC(hDC);
    ReleaseDC(NULL, hScreen);

    return frame;
}
```

Hình 2.10 – Source code: Function **CaptureScreen**

– **void SendImageData(SOCKET clientSocket)**

- \* Sử dụng hàm **CaptureScreen** để chụp ảnh từ màn hình máy tính.
- \* Giảm kích thước của ảnh bằng cách sử dụng hàm **cv::resize** với hệ số tỉ lệ là 75%.
- \* Sử dụng hàm **cv::imencode** để nén ảnh vào định dạng JPEG và lưu vào một vector buffer.
- \* Chia dữ liệu thành các gói có kích thước tối đa là **packetSize** và gửi chúng thông qua kết nối socket. Mỗi gói đầu tiên chứa kích thước thực của gói dữ liệu tiếp theo.
- \* Gửi một **marker** (-1) để đánh dấu kết thúc dữ liệu.
- \* Giải phóng bộ nhớ và tài nguyên được cấp phát, bao gồm việc giải phóng bộ nhớ của **frame**, làm trống vector **buffer**, và giảm kích thước của vector nếu cần thiết.

```

void SendImageData(SOCKET clientSocket) {
    const int packetSize = 256;
    while (1) {

        cv::Mat frame = CaptureScreen();

        cv::resize(frame, frame, cv::Size(), 0.75, 0.75);

        std::vector<uchar> buffer;

        cv::imencode(".jpg", frame, buffer);

        int size = static_cast<int>(buffer.size());

        int sentBytes = 0;
        while (sentBytes < size) {
            int remainingBytes = size - sentBytes;
            int bytesToSend = std::min(remainingBytes, packetSize);

            send(clientSocket, reinterpret_cast<char*>(&bytesToSend), sizeof(int), 0);
            send(clientSocket, reinterpret_cast<char*>(buffer.data()), bytesToSend, 0);

            sentBytes += bytesToSend;

            // Clear
            buffer = std::vector<uchar>(buffer.begin() + bytesToSend, buffer.end());
        }

        int endMarker = -1;
        send(clientSocket, reinterpret_cast<char*>(&endMarker), sizeof(int), 0);

        frame.release();
        buffer.clear();
        buffer.shrink_to_fit();

        Sleep(1);
    }
}

```

Hình 2.11 – Source code: Function **SendImageData**.

- **Client:** Tiến trình nhận dữ liệu và hiển thị ảnh:

– **void Image::receiveImages(SOCKET clientSocket)**

*Nhận kích thước mảnh dữ liệu:*

- \* Sử dụng hàm **recv** để nhận kích thước của mảnh dữ liệu tiếp theo.
- \* Sử dụng OpenCV (**cv::imdecode**) để giải mã mảnh dữ liệu thành một hình ảnh **cv::Mat**.
- \* Kiểm tra xem số byte nhận được có phải là kích thước của mảnh không. Nếu không, xuất thông báo lỗi và thoát khỏi vòng lặp.
- \* Loại bỏ hình ảnh và làm sạch dữ liệu đã nhận.

*Kiểm tra dấu hiệu kết thúc:*

- \* Nếu kích thước là -1, đây là dấu hiệu cho một hình ảnh hoàn chỉnh.
- \* Hiển thị hình ảnh trên cửa sổ với tên "**Virtual Machine Screen**" bằng cách sử dụng OpenCV **cv::imshow**.

*Nhận dữ liệu mảnh:*

- \* Nếu kích thước không phải là -1, đó là kích thước của mảnh dữ liệu thực tế.

- \* Tạo một vector **buffer** để nhận mảnh dữ liệu.
- \* Sử dụng **recv** để nhận mảnh dữ liệu và kiểm tra xem số byte nhận được có bằng với kích thước mảnh không.

*Nối dữ liệu mảnh vào vector:*

- \* Nối mảnh dữ liệu nhận được vào vector **receivedData**.

```
void Image::receiveImages(SOCKET clientSocket) {
    std::vector<uchar> receivedData;
    while (true) {
        int chunkSize;
        int receivedBytes = recv(clientSocket, reinterpret_cast<char*>(&chunkSize), sizeof(int), 0);

        if (receivedBytes != sizeof(int)) {
            std::cout << "Error receiving image size." << std::endl;
            break;
        }

        if (chunkSize == -1) {
            cv::Mat frame = cv::imdecode(receivedData, cv::IMREAD_COLOR);

            if (!frame.empty()) {
                cv::imshow("Virtual Machine Screen", frame);
                cv::setMouseCallback("Virtual Machine Screen", mouseEventControl, NULL);
                cv::Rect windowRect = cv::getWindowImageRect("Virtual Machine Screen");
                cv::waitKey(1);
                frame.release();
                receivedData.clear();
            }
        }
        else {
            std::vector<uchar> buffer(chunkSize);
            receivedBytes = recv(clientSocket, reinterpret_cast<char*>(buffer.data()), chunkSize, 0);

            std::cout << " ";

            if (receivedBytes != chunkSize) {
                std::cout << "Error receiving image data." << std::endl;
            }
            receivedData.insert(receivedData.end(), buffer.begin(), buffer.end());
        }
    }
}
```

Hình 2.12 – Source code: Function **receiveImages**

*Tiến trình nhận dữ liệu và hiển thị ảnh được chạy độc lập trong một luồng riêng:*

```
void Image::threadImage(SOCKET clientSocket) {
    std::thread receiveThread(&Image::receiveImages, this, clientSocket);
    receiveThread.join();
}
```

Hình 2.13 – Source code: **threadImage**

### 2.3.2.2 Chuột

#### Ý tưởng:

- **Server:** Sử dụng đồng thời hai luồng (thread) chạy đồng thời để thực hiện nhiệm vụ:



- **ReceiveMouseEvent**: Tiếp nhận thông tin sự kiện chuột (tọa độ, và sự kiện nhấn, thả hoặc cuộn) và lưu vào biến **mouseState**.
- **putMouse**: Lấy giá trị của **mouseState** và gửi đến hệ thống.
- **Client**: Sử dụng đồng thời hai luồng (thread) chạy đồng thời để thực hiện nhiệm vụ:
  - **mouseEventControl**: Bắt sự kiện chuột (chỉ lấy các sự kiện được diễn ra trong phạm vi window hiển thị màn hình của server) và lưu vào biến. **mouseState**.
  - **sendMouseInfomation**: Lấy giá trị của **mouseState** và gửi đến server.

### Cài đặt hàm:

- Định nghĩa một enum class **MouseEvent** để định nghĩa các sự kiện chuột (tránh xung đột tên):

```
enum class MouseEvent
{
    None,
    LeftButtonDown,
    LeftButtonUp,
    LeftDoubleClick,
    RightButtonDown,
    RightButtonUp,
    RightDoubleClick,
    MiddleButtonDown,
    MiddleButtonUp,
    ScrollUp,
    ScrollDown
};
```

Hình 2.14 – Souce code: Enum class **MouseEvent**

- Định nghĩa class **Mouse** với các thuộc tính **coordinate**, **event** tương ứng với tọa độ và sự kiện (nhấn, thả, cuộn) chuột và cài đặt các phương thức **constructor**, **set**, **get** tương ứng:

```
class Mouse {
private:
    POINT coordinate;
    MouseEvent event;
public:
    Mouse();
    Mouse(POINT coordinate, MouseEvent event);

    void setPosition(POINT coordinate);
    POINT getPosition();

    void setEvent(MouseEvent event);
    MouseEvent getEvent();
};

Mouse::Mouse() {
    this->coordinate = { 0,0};
    this->event = MouseEvent::None;
}

Mouse::Mouse(POINT coordinate, MouseEvent event) {
    this->coordinate = coordinate;
    this->event = event;
}

void Mouse::setPosition(POINT coordinate) {
    this->coordinate = coordinate;
}

POINT Mouse::getPosition() {
    return this->coordinate;
}

void Mouse::setEvent(MouseEvent event) {
    this->event = event;
}

MouseEvent Mouse::getEvent() {
    return this->event;
}
```

Hình 2.15 – Souce code: Class **Mouse****•Server:****– void ReceiveMouseEvent(SOCKET clientSocket)**

*Tiếp nhận thông tin sự kiện chuột:*

- \* Sử dụng một buffer để lưu dữ liệu nhận được từ client thông qua **clientSocket**.
- \* Nhận dữ liệu và kiểm tra có nhận được dữ liệu thành công hay không.

*Lưu thông tin sự kiện chuột biến **mouseState**:*

- \* Đọc dữ liệu từ buffer (thông qua hàm **memcpy**).
- \* Chuyển đổi dữ liệu nhận được thành các giá trị tọa độ, sự kiện chuột có kiểu dữ liệu tương ứng với các thuộc tính của biến **mouseState**.
- \* Lưu các dữ liệu trên vào biến **mouseState** thông qua các phương thức **setPosition**, **setEvent**.

```

void ReceiveMouseEvent(SOCKET clientSocket) {
    // While loop: accept and echo message back to client
    char buf[sizeof(int) * 3];
    while (true) {
        memset(buf, 0, sizeof(int) * 3);
        // Wait for client to send data
        int bytesReceived = recv(clientSocket, buf, sizeof(buf), 0);
        if (bytesReceived == SOCKET_ERROR) {
            std::cerr << "Error in receive mouse events. Quitting" << std::endl;
            mouseState.setPosition({ -1,-1 });
            mouseState.setEvent(MouseEvent::None);
            break;
        }

        if (bytesReceived == 0) {
            std::cout << "Client disconnected " << std::endl;
            mouseState.setPosition({ -1,-1 });
            mouseState.setEvent(MouseEvent::None);
            break;
        }

        MouseEvent mouseEvent = MouseEvent::None;
        int x = 0, y = 0, eventData = 0;
        memcpy(&x, buf, sizeof(int));
        memcpy(&y, buf + sizeof(int), sizeof(int));
        memcpy(&eventData, buf + 2 * sizeof(int), sizeof(int));
        std::cout << "1. x = " << x << " y = " << y << " event " << eventData << std::endl;

        // Convert received string to the corresponding
        if (eventData == 11) {
            mouseEvent = MouseEvent::LeftButtonDown;
        }
        else if (eventData == 10) {
            mouseEvent = MouseEvent::LeftButtonUp;
        }
        else if (eventData == 22) {
            mouseEvent = MouseEvent::LeftDoubleClick;
        }
        else if (eventData == 21) {
            mouseEvent = MouseEvent::RightButtonDown;
        }
        else if (eventData == 20) {
            mouseEvent = MouseEvent::RightButtonUp;
        }
        else if (eventData == 31) {
            mouseEvent = MouseEvent::MiddleButtonDown;
        }
        else if (eventData == 30) {
            mouseEvent = MouseEvent::MiddleButtonUp;
        }
        else if (eventData == 40) {
            mouseEvent = MouseEvent::ScrollUp;
        }
        else if (eventData == 41) {
            mouseEvent = MouseEvent::ScrollDown;
        }

        mouseState.setPosition({ x,y });
        mouseState.setEvent(mouseEvent);
    }
}

```

Hình 2.16 – Source code: Function **ReceiveMouseEvent**

– **void putMouse()**: Sử dụng 1 vòng lặp (có độ trễ 1ms) để liên tục thực hiện:

- \* Chuyển đổi thông tin chuột **mouseState** để set giá trị cho các thuộc tính của biến **input**.
- \* Gửi dữ liệu chuột đến hệ thống thông qua hàm **SendInput** và **setCursorPos**.

```

void putMouse() {
    // Simulate the mouse event
    int formerX = -1;
    int formerY = -1;
    MouseEvent formerEvent = MouseEvent::None;
    while (1) {
        int x = mouseState.getPosition().x;
        int y = mouseState.getPosition().y;
        MouseEvent mouseEvent = mouseState.getEvent();
        //Check for changing
        if (x != formerX || y != formerY || formerEvent != mouseEvent) {
            // when client disconnects or error in receive mouse events
            if (x == -1 || y == -1) return;
            formerX = x;
            formerY = y;
            formerEvent = mouseEvent;

            INPUT input{};
            input.type = INPUT_MOUSE;
            input.mi.dwFlags = MOUSEEVENTF_MOVE | MOUSEEVENTF_ABSOLUTE;
            input.mi.dx = x / 0.75;
            input.mi.dy = y / 0.75;

            switch (mouseEvent) {
                case MouseEvent::LeftButtonDown:
                    input.mi.dwFlags = MOUSEEVENTF_LEFTDOWN;
                    break;
                case MouseEvent::LeftButtonUp:
                    input.mi.dwFlags = MOUSEEVENTF_LEFTUP;
                    break;
                case MouseEvent::RightButtonDown:
                    input.mi.dwFlags = MOUSEEVENTF_RIGHTDOWN;
                    break;
                case MouseEvent::RightButtonUp:
                    input.mi.dwFlags = MOUSEEVENTF_RIGHTUP;
                    break;
                case MouseEvent::MiddleButtonDown:
                    input.mi.dwFlags = MOUSEEVENTF_MIDDLEDOWN;
                    break;
                case MouseEvent::MiddleButtonUp:
                    input.mi.dwFlags = MOUSEEVENTF_MIDDLEUP;
                    break;
                case MouseEvent::ScrollUp:
                    input.mi.dwFlags = MOUSEEVENTF_WHEEL;
                    input.mi.mouseData = WHEEL_DELTA;
                    break;
                case MouseEvent::ScrollDown:
                    input.mi.dwFlags = MOUSEEVENTF_WHEEL;
                    input.mi.mouseData = -WHEEL_DELTA;
                    break;
                default:
                    input.mi.dwFlags = MOUSEEVENTF_MOVE;
                    break;
            }

            SendInput(1, &input, sizeof(INPUT));
            SetCursorPos(input.mi.dx, input.mi.dy);

            Sleep(1);
        }
    }
}

```

Hình 2.17 – Source code: Function **putMouse**.



• **Client:**

*Tiến trình lấy sự kiện chuột:*

- **cv::setMouseCallback("Virtual Machine Screen", mouseEventControl, NULL)**

Hàm callback để liên tục bắt sự kiện chuột khi có sự kiện chuột xảy ra trong window hiển thị màn hình server. Hàm này chạy chung luồng với tiến trình hiện ảnh của màn hình server.

- **void mouseEventControl(int event, int x, int y, int flags, void\* userData)**

Sử dụng thông tin nhận được từ hàm callback ở trên để set giá trị cho các thuộc tính của biến **mouseState**.

```
void mouseEventControl(int event, int x, int y, int flags, void* userData) {
    if (event == cv::EVENT_MOUSEMOVE) {
        mouseState.setPosition({ x, y });
        mouseState.setEvent(MouseEvent::None);
    }
    else if (event == cv::EVENT_LBUTTONDOWN) {
        mouseState.setPosition({ x, y });
        mouseState.setEvent(MouseEvent::LeftButtonDown);
    }
    else if (event == cv::EVENT_LBUTTONUP) {
        mouseState.setPosition({ x, y });
        if (mouseState.getEvent() != MouseEvent::LeftButtonUp) mouseState.setEvent(MouseEvent::LeftButtonUp);
        else mouseState.setEvent(MouseEvent::None);
    }
    else if (event == cv::EVENT_LBUTTONDBLCLK) {
        mouseState.setPosition({ x, y });
        mouseState.setEvent(MouseEvent::LeftDoubleClick);
    }
    else if (event == cv::EVENT_RBUTTONDOWN) {
        mouseState.setPosition({ x, y });
        mouseState.setEvent(MouseEvent::RightButtonDown);
    }
    else if (event == cv::EVENT_RBUTTONUP) {
        mouseState.setPosition({ x, y });
        if (mouseState.getEvent() != MouseEvent::RightButtonUp) mouseState.setEvent(MouseEvent::RightButtonUp);
        else mouseState.setEvent(MouseEvent::None);
    }
    else if (event == cv::EVENT_RBUTTONDBLCLK) {
        mouseState.setPosition({ x, y });
        mouseState.setEvent(MouseEvent::RightDoubleClick);
    }
    else if (event == cv::EVENT_MBUTTONDOWN) {
        mouseState.setPosition({ x, y });
        mouseState.setEvent(MouseEvent::MiddleButtonDown);
    }
    else if (event == cv::EVENT_MBUTTONUP) {
        mouseState.setPosition({ x, y });
        if (mouseState.getEvent() != MouseEvent::MiddleButtonUp) mouseState.setEvent(MouseEvent::MiddleButtonUp);
        else mouseState.setEvent(MouseEvent::None);
    }
    else if (event == cv::EVENT_MOUSEWHEEL) {
        // Mouse wheel scroll
        int delta = cv::getMouseWheelDelta(flags);
        if (delta > 0) {
            // Scroll up
            mouseState.setPosition({ x, y });
            mouseState.setEvent(MouseEvent::ScrollUp);
        }
        else {
            // Scroll down
            mouseState.setPosition({ x, y });
            mouseState.setEvent(MouseEvent::ScrollDown);
        }
    }
}
```

Hình 2.18 – Source code: Function **mouseEventControl**

*Tiến trình gọi sự kiện chuột đến server:*

– **bool getBufferMouseInformation(char\* buf)**

Lưu các thông tin về vị trí và sự kiện chuột vào trong một buffer để chuẩn bị cho việc gửi dữ liệu đến server.

```
bool getBufferMouseInformation(char* buf) {
    // Convert mouse event to a string representation
    memset(buf, 0, 3 * sizeof(int));
    int eventData = 0;
    switch (mouseState.getEvent()) {
        case MouseEvent::LeftButtonDown:
            eventData = 11;
            break;
        case MouseEvent::LeftButtonUp:
            eventData = 10;
            break;
        case MouseEvent::LeftDoubleClick:
            eventData = 12;
            break;
        case MouseEvent::RightButtonDown:
            eventData = 21;
            break;
        case MouseEvent::RightButtonUp:
            eventData = 20;
            break;
        case MouseEvent::RightDoubleClick:
            eventData = 22;
            break;
        case MouseEvent::MiddleButtonDown:
            eventData = 31;
            break;
        case MouseEvent::MiddleButtonUp:
            eventData = 30;
            break;
        case MouseEvent::ScrollUp:
            eventData = 40;
            break;
        case MouseEvent::ScrollDown:
            eventData = 41;
            break;
        default:
            eventData = 0;
            break;
    }
    int x = mouseState.getPosition().x, y = mouseState.getPosition().y;
    memcpy(buf, &x, sizeof(int));
    memcpy(buf + sizeof(int), &y, sizeof(int));
    memcpy(buf + 2 * sizeof(int), &eventData, sizeof(int));
    return true;
}
```

Hình 2.19 – Source code: Function **getBufferMouseInformation**

– **void sendMouseInfomation(SOCKET sock)**

Kiểm tra tọa độ và sự kiện chuột, nếu có sự thay đổi thì tiến hành tạo buffer, lấy thông tin chuột lưu vào buffer và gửi đến server.

```
void sendMouseInfomation(SOCKET sock) {
    int formerX = -1;
    int formerY = -1;
    MouseEvent formerEvent = MouseEvent::None;
    std::cout << "In before before send func" << mouseState.getPosition().x << " " << mouseState.getPosition().y << "\n";
    while (1) {
        int x = mouseState.getPosition().x;
        int y = mouseState.getPosition().y;
        MouseEvent mouseEvent = mouseState.getEvent();
        if (x != formerX || y != formerY || formerEvent != mouseEvent) {
            formerX = x;
            formerY = y;
            formerEvent = mouseEvent;
            std::cerr << "In send func" << mouseState.getPosition().x << " " << mouseState.getPosition().y << "\n";
            char buf[3 * sizeof(int)];
            bool isVal = getBufferMouseInformation(buf);
            if (isVal)
                send(sock, buf, 3 * sizeof(int), 0);
        }
        Sleep(1);
    }
}
```

Hình 2.20 – Source code: Function **sendMouseInfomation**

*Tiến trình gửi thông tin chuột được chạy độc lập trong một luồng riêng biệt.*

```
void Mouse::threadMouse(SOCKET clientSocket) {
    std::thread sendMouseEventThread(sendMouseInfomation, clientSocket);
    sendMouseEventThread.detach();
}
```

Hình 2.21 – Source code: **threadMouse**

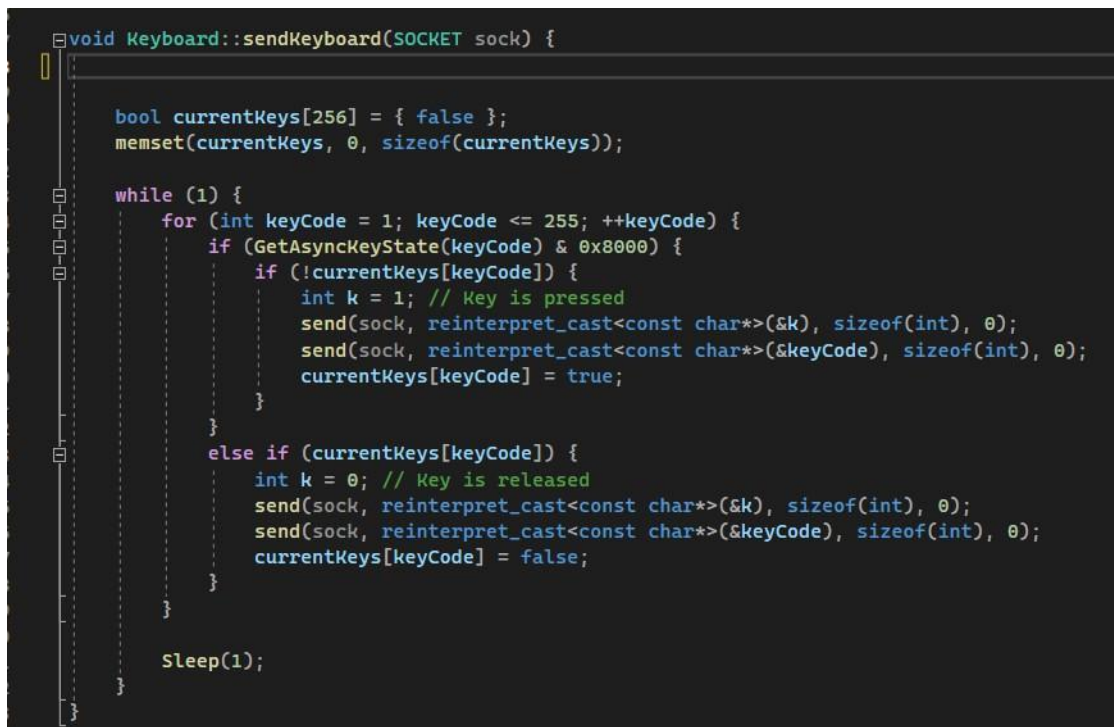
### 2.3.2.3 Bàn phím

Ở phần này em sẽ giới thiệu về sự kiện của bàn phím trong đồ án của nhóm em.

- *Client*

Các hàm được sử dụng trong client bao gồm:

\* **void sendKeyboard(SOCKET sock)**



```

void Keyboard::sendKeyboard(SOCKET sock) {

    bool currentKeys[256] = { false };
    memset(currentKeys, 0, sizeof(currentKeys));

    while (1) {
        for (int keyCode = 1; keyCode <= 255; ++keyCode) {
            if (GetAsyncKeyState(keyCode) & 0x8000) {
                if (!currentKeys[keyCode]) {
                    int k = 1; // Key is pressed
                    send(sock, reinterpret_cast<const char*>(&k), sizeof(int), 0);
                    send(sock, reinterpret_cast<const char*>(&keyCode), sizeof(int), 0);
                    currentKeys[keyCode] = true;
                }
            }
            else if (currentKeys[keyCode]) {
                int k = 0; // Key is released
                send(sock, reinterpret_cast<const char*>(&k), sizeof(int), 0);
                send(sock, reinterpret_cast<const char*>(&keyCode), sizeof(int), 0);
                currentKeys[keyCode] = false;
            }
        }

        Sleep(1);
    }
}
  
```

Hình 2.22 – source code của hàm **sendKeyboard**

Hàm **sendKeyboard** có chức năng gửi thông tin về các phím được nhấn và nhả ra từ bàn phím tới một **SOCKET** (socket) đã được thiết lập.

Cụ thể, hàm này có các bước thực hiện sau:

- Khởi tạo một mảng **currentKeys** gồm 256 phần tử, ban đầu tất cả các phần tử đều được đặt giá trị false. Mảng này sẽ lưu trạng thái của các phím, dùng để xác định xem một phím đã được nhấn hay đã được nhả ra.
- Trong vòng lặp vô hạn (**while(1)**), hàm sẽ duyệt qua từng mã phím từ 1 tới 255 để kiểm tra xem các phím đó có đang được nhấn hay không.
- Đối với mỗi mã phím (**keyCode**), hàm sẽ sử dụng hàm **GetAsyncKeyState** để kiểm tra xem phím có đang được nhấn không. Nếu phím đang được nhấn (**GetAsyncKeyState(keyCode) & 0x8000**), và trạng thái trước đó của phím là không được nhấn (**!currentKeys[keyCode]**), hàm sẽ thực hiện các bước sau:
  - a. Gửi một số nguyên 1 (**int k = 1**) thông qua socket sock, để chỉ định rằng một phím đã được nhấn.
  - b. Gửi mã phím (**keyCode**) thông qua socket sock, để truyền thông tin

về phím nào đã được nhấn.

**c.** Đánh dấu phím hiện tại (**currentKeys[keyCode]**) là đã được nhấn (**true**).

- Trong trường hợp phím không được nhấn (**keyCode**), tức là phím đã được nhấn trước đó nhưng không còn được nhấn nữa, hàm sẽ thực hiện các bước sau:

**a.** Gửi một số nguyên 0 (**int k = 0**) thông qua socket **sock**, để chỉ định rằng một phím đã được nhả ra (không còn được nhấn).

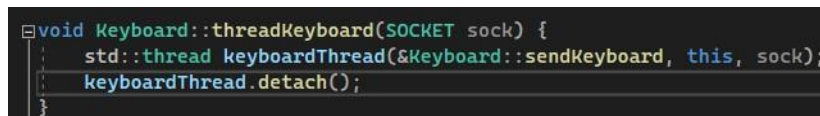
**b.** Gửi mã phím (**keyCode**) thông qua socket **sock**, để truyền thông tin về phím nào đã được nhả ra.

**c.** Đánh dấu phím hiện tại (**currentKeys[keyCode]**) là không được nhấn (**false**).

- Cuối cùng, sau khi kiểm tra tất cả các mã phím từ 1 tới 255, hàm sẽ dừng một khoảng thời gian ngắn (**Sleep(1)**) trước khi tiếp tục vòng lặp.

hàm **sendKeyboard** này sẽ liên tục gửi thông tin về trạng thái của các phím được nhấn và nhả ra từ bàn phím tới một socket đã được thiết lập.

\* **void threadKeyboard(SOCKET sock)**



```
void Keyboard::threadKeyboard(SOCKET sock) {  
    std::thread keyboardThread(&Keyboard::sendKeyboard, this, sock);  
    keyboardThread.detach();  
}
```

Hình 2.23 – source code của hàm **threadKeyboard**

Hàm **threadKeyboard** có chức năng tạo một luồng (**thread**) mới để thực thi hàm **sendKeyboard** trong lớp **Keyboard**.

Cụ thể, hàm này có các bước thực hiện sau:

- Tạo một đối tượng luồng (**keyboardThread**) từ lớp **std::thread**. Hàm **std::thread** nhận vào một con trỏ hàm thành viên và các đối số liên quan để tạo một luồng mới. Trong trường hợp này, con trỏ hàm thành viên được truyền vào là **&Keyboard::sendKeyboard**, tức là hàm **sendKeyboard** trong lớp **Keyboard**.
- Đối tượng luồng (**keyboardThread**) được khởi tạo với các đối số sau: **this**: Con trỏ đối tượng hiện tại (được sử dụng để truy cập vào thành viên và phương thức của lớp). **sock**: **Socket** được truyền vào từ đối số của hàm **threadKeyboard**.
- Gọi phương thức **detach()** trên đối tượng luồng (**keyboardThread**). Phương thức **detach()** được sử dụng để tách luồng mới tạo ra từ luồng chính. Điều này có nghĩa là luồng mới sẽ tiếp tục thực thi độc lập và



không cần phụ thuộc vào luồng chính. Trong trường hợp này, luồng **keyboardThread** sẽ thực thi hàm **sendKeyboard** trong lớp **Keyboard** mà không cần sự can thiệp của luồng gọi **threadKeyboard**.

Hàm **threadKeyboard** tạo một luồng mới và thực thi hàm **sendKeyboard** trong lớp **Keyboard** trên luồng đó. Sau đó, luồng mới được tách ra và tiếp tục thực thi độc lập với luồng gọi **threadKeyboard**. Điều này cho phép việc gửi thông tin từ bàn phím thông qua socket (**sock**) được thực hiện trong một luồng riêng biệt mà không làm chặn luồng chính.

### • Server

Các hàm được sử dụng trong server bao gồm:

#### \* **void ReceiveAndSendKey(SOCKET clientSocket)**

```
// Keyboard
// Receive data from client and show event keyboard in controlled machine
void ReceiveAndSendKey(SOCKET clientSocket)
{
    bool currentKeys[256] = { false };
    static bool previousKeys[256] = { false };
    memset(currentKeys, 0, sizeof(currentKeys));
    memset(previousKeys, 0, sizeof(previousKeys));
    while (true)
    {
        int key;
        int k;
        int bytesReceived = recv(clientSocket, reinterpret_cast<char*>(&k), sizeof(k), 0);
        int bytesRead = recv(clientSocket, reinterpret_cast<char*>(&key), sizeof(key), 0);
        currentKeys[key] = k;
        if (bytesReceived == SOCKET_ERROR)
        {
            std::cerr << "Failed to receive data." << std::endl;
            break;
        }
        if (bytesRead == SOCKET_ERROR)
        {
            std::cerr << "Failed to receive data." << std::endl;
            break;
        }

        // Show event keyboard
        INPUT input{};
        input.type = INPUT_KEYBOARD;
        input.ki.wVk = key;
        input.ki.wScan = MapVirtualKeyEx(key, MAPVK_VK_TO_VSC, GetKeyboardLayout(0));
        input.ki.dwFlags = 0;

        for (int i = 1; i <= 255; i++) {
            if (currentKeys[i] && !previousKeys[i]) {
                SendInput(1, &input, sizeof(INPUT));
                previousKeys[i] = true;
            }
            else if (!currentKeys[i] && previousKeys[i]) {
                input.ki.dwFlags |= KEYEVENTF_KEYUP;
                SendInput(1, &input, sizeof(INPUT));
                previousKeys[i] = false;
            }
        }
    }
}
```

Hình 2.24 – Source code: Function **ReceiveAndSendKey**

- Mảng **currentKeys** và **previousKeys** được khởi tạo với giá trị mặc định là false cho tất cả các phần tử. Mảng **currentKeys** lưu trữ trạng thái hiện tại của các phím, trong khi mảng **previousKeys** lưu trữ trạng thái trước đó của các phím.
- Hai lệnh **memset** được sử dụng để đặt tất cả các phần tử của **currentKeys** và **previousKeys** về 0. Điều này đảm bảo rằng các mảng có giá trị ban đầu là false.
- Một vòng lặp vô hạn (**while (true)**) được sử dụng để liên tục nhận và xử lý thông tin từ client.
- Hai lệnh **recv** được sử dụng để nhận dữ liệu từ socket. Đầu tiên, **recv** nhận thông tin về trạng thái của phím (k), sau đó **recv** nhận giá trị mã của phím (**key**). Dữ liệu được nhận được gán cho **currentKeys[key]** để cập nhật trạng thái hiện tại của phím.
- Hai câu điều kiện i được sử dụng để kiểm tra xem việc nhận dữ liệu có thành công hay không. Nếu có lỗi xảy ra trong quá trình nhận dữ liệu, thông báo lỗi sẽ được in ra màn hình và vòng lặp sẽ được thoát.
- Một đối tượng **INPUT** mới được khởi tạo, đại diện cho một sự kiện bàn phím. Các thuộc tính của đối tượng **INPUT** được thiết lập để phù hợp với

phím được nhận. `input.type` được đặt thành **INPUT\_KEYBOARD** để chỉ định rằng đây là một sự kiện bàn phím. `input.ki.wVk` được gán bằng giá trị mã phím, `input.ki.wScan` được gán bằng mã quét của phím, và `input.ki.dwFlags` được đặt thành 0 ban đầu.

- Một vòng lặp từ  $i = 1$  đến 255 được sử dụng để duyệt qua tất cả các phím. Trong vòng lặp này, các câu điều kiện if kiểm tra trạng thái của từng phím.
- Nếu `currentKeys[i]` là true và `previousKeys[i]` là false, tức là phím  $i$  đã được nhấn và trước đó không được nhấn, sự kiện bàn phím nhấn sẽ được gửi tới hệ thống bằng hàm **SendInput**. Sau đó, `previousKeys[i]` được cập nhật thành true.
- Ngược lại, nếu `currentKeys[i]` là false và `previousKeys[i]` là true, tức là phím  $i$  đã được nhả ra sau khi được nhấn trước đó, sự kiện bàn phím nhả ra sẽ được gửi tới hệ thống bằng cách thiết lập cờ **KEYEVENTF\_KEYUP** trong `input.ki.dwFlags` và gửi thông tin sự kiện phím bằng hàm **SendInput**. Sau đó, `previousKeys[i]` được cập nhật thành false.

## 2.3.3 Giao diện

### 2.3.3.1 Button

- Định nghĩa class **Button** với các thuộc tính **rect**, **text**, **clicked** tương ứng với hình chữ nhật, văn bản và sự kiện click và cài đặt phương thức **constructor**, **set**, **get** tương ứng:

```
class Button {
private:
    sf::RectangleShape rect;
    sf::Text text;
    bool clicked;
public:
    Button();
    Button(float w, float h, const sf::Color& cRect, const std::string& s, const sf::Font& font, const sf::Color& cText);

    // Set, Get method of Rectangle
    void setColorRect(const sf::Color& c);
    void setRect(float w, float h, const sf::Color& c);
    sf::RectangleShape getRect();

    // Set, Get method of Text
    void setString(const std::string& s);
    void setColorText(const sf::Color& c);
    void setPosText(float x, float y);
    void setSizeText(int sizeText);
    void setText(const std::string& s, const sf::Font& font, const sf::Color& c);
    sf::Text getText();

    // Set, Get method of Clicked
    void setClicked(bool clicked);
    bool getClicked();

    // Set method of button
    void setPosition(float posX, float posY);

    // Draw button
    void draw(sf::RenderWindow& window);
};
```

Hình 2.25 – Source code: Class **Button**

- **void Button::draw(sf::RenderWindow& window)**

- Sử dụng **window.draw(this->rect)** để vẽ hình chữ nhật của nút lên cửa sổ đồ họa.
- Sử dụng **window.draw(this->text)** để vẽ văn bản của nút lên cửa sổ đồ họa.

```
// Draw button
void Button::draw(sf::RenderWindow& window) {
    window.draw(this->rect);
    window.draw(this->text);
}
```

Hình 2.26 – Source code: Draw **Button**

*Đây là kết quả sau khi gọi Button:*



Hình 2.27 – Display button

### 2.3.3.2 Event

- Định nghĩa class **Event** với các thuộc tính **event**, tương ứng với sự kiện **sf::Event** trong SFML và cài đặt phương thức **constructor**, **set**, **get**, tương ứng:

```
// Constructor
Event::Event() {
}

// Set, Get method of sf::Event
void Event::setEvent(sf::Event& event) {
    this->event = event;
}

sf::Event Event::getEvent() {
    return this->event;
}

// Event Close
bool Event::Closed() {
    return (this->event.type == sf::Event::Closed);
}

class Event {
private:
    sf::Event event;
public:
    Event();

    // Set, Get method of sf::Event
    void setEvent(sf::Event& event);
    sf::Event getEvent();

    // Event Close
    bool Closed();

    // Event check click button
    bool Clicked(Button* button);
};
```

Hình 2.28 – Source code: Class **Event**

- **bool Event::Clicked(Button\* button)**

- Hàm bắt đầu bằng việc kiểm tra xem sự kiện chuột có phải là lúc nhấn chuột **sf::Event::MouseButtonPressed** không.
- Nếu là sự kiện nhấn chuột, hàm tiếp tục kiểm tra xem tọa độ của chuột có nằm trong hình chữ nhật của nút không.
- Nếu tọa độ chuột nằm trong hình chữ nhật của nút, trạng thái "đã nhấn" của nút được chuyển đảo (toggled).
- Hàm trả về true nếu sự kiện được xác nhận là việc nhấn chuột và tọa độ nằm trong hình chữ nhật của nút và ngược lại trả về false.

A screenshot of a code editor showing the implementation of the `Event::Clicked` function. The code is in C++ and uses the SFML library. It checks if the event type is `sf::Event::MouseButtonPressed`. If so, it checks if the mouse button's coordinates are within the button's global bounds. If both conditions are met, it toggles the button's `clicked` state and returns `true`. Otherwise, it returns `false`.

```
// Event check click button
bool Event::Clicked(Button* button) {
    if (this->event.type == sf::Event::MouseButtonPressed) {
        if (button->getRect().getGlobalBounds().contains((float)event.mouseButton.x, (float)event.mouseButton.y)) {
            button->setClicked(!button->getClicked());
            return true;
        }
    }
    return false;
}
```

Hình 2.29 – Source code: **Event::Clicked**

### 2.3.3.3 Slide 1

- Định nghĩa các hàm, các thuộc tính và chức năng của các nút tương ứng trong cửa sổ đầu tiên của giao diện.



```

class Slidel {
private:
    // Position of objects in graphics
    float posX;
    float posY;
    float padding;

    // Header
    Button* header;

    // Button
    Button* buttonServer;
    std::vector<Button*> buttonItems;
    Button* buttonConnected;

public:
    Slidel();
    ~Slidel();

    // Set, Get method of PosX
    void setPosX(float posX);
    float getPosX();

    // Set, Get method of PosY
    void setPosY(float posY);
    float getPosY();

    // Set, Get method of Padding
    void setPadding(float padding);
    float getPadding();

    // Set, Get method of Header
    void setHeader(Button* header);
    Button* getHeader();

    // Set, Get method of buttonServer
    void setButtonServer(Button* buttonServer);
    Button* getButtonServer();

    // Set, Get method of buttonItems
    void setButtonItems(std::vector<Button*> buttonItems);
    std::vector<Button*> getButtonItems();

    // Set, Get method of buttonConnected
    void setButtonConnected(Button* buttonConnected);
    Button* getButtonConnected();

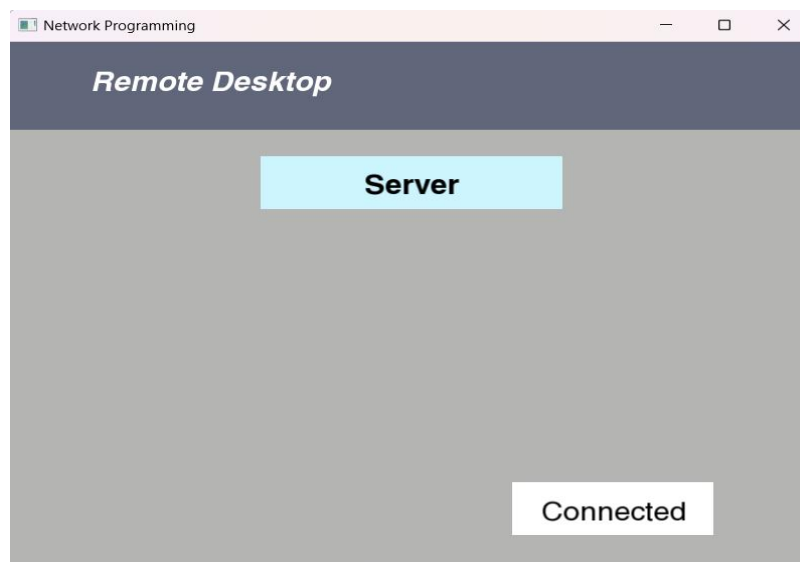
    // Run
    void run(sf::RenderWindow& window, Client client);
};

```

Hình 2.30 – Source code: Class **Slide 1**

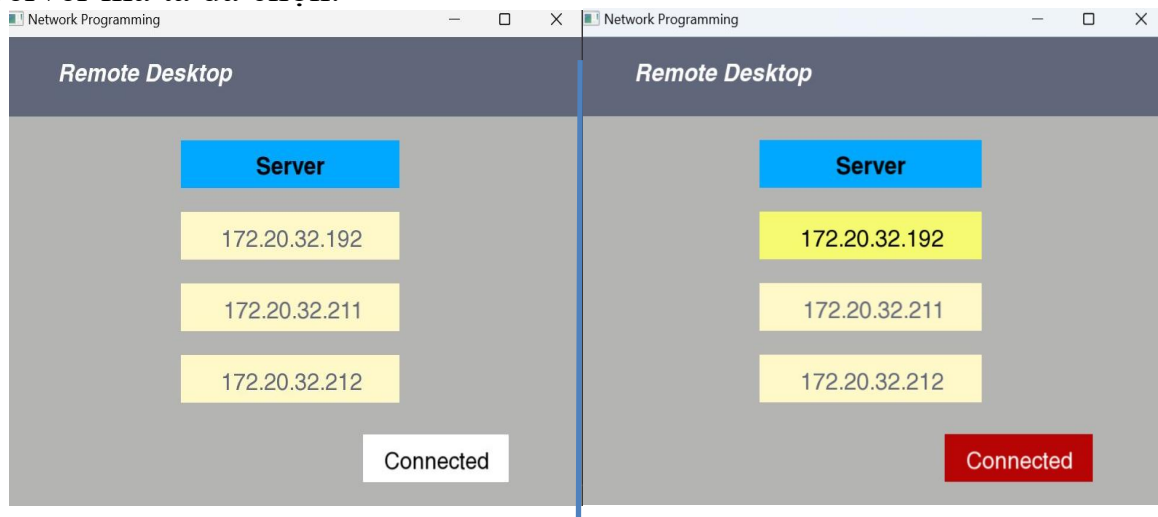
• **void run(sf::RenderWindow& window, Client client)**

Trong phương thức này, các phần tử giao diện như nút (Button) và tiêu đề (Header) được khởi tạo và vẽ lên cửa sổ. Nếu người dùng tương tác với các phần tử giao diện, ví dụ như nhấp chuột vào một nút, các sự kiện liên quan sẽ được xử lý.



Hình 2.31 – Giao diện slide 1 - Trạng thái off

- Lúc này trên màn hình sẽ có nút là nút “**Server**” và nút “**Connected**”.
- Nếu người dùng nhấp chuột vào nút “**Server**”, thì lúc này cửa sổ sẽ hiện thị ra các nút “**Computer**”, số nút “**Compute**” sẽ tương ứng với số máy Server mà chúng ta cung cấp. Khi ta nhấn nút “**Server**” lại lần nữa thì các nút “**Compute**” sẽ bị ẩn đi.
- Khi nhấp chuột vào một trong các nút “**buttonItems**” (được lưu trong mảng **buttonItems**), trong trường hợp này khi ta nhấn vào nút **Computer** thì lúc này nút “**Computer**” tương ứng sẽ được hiển thị là đang chọn (ô đó sáng màu lên). Lúc này nút “**Connected**” sẽ được đưa vào trạng thái có thể chọn (ô chuyển sang màu đỏ). Ta nhấn chuột vào “**Connected**” thì client sẽ gửi thông báo tới Server và kết nối với Server mà ta đã chọn.

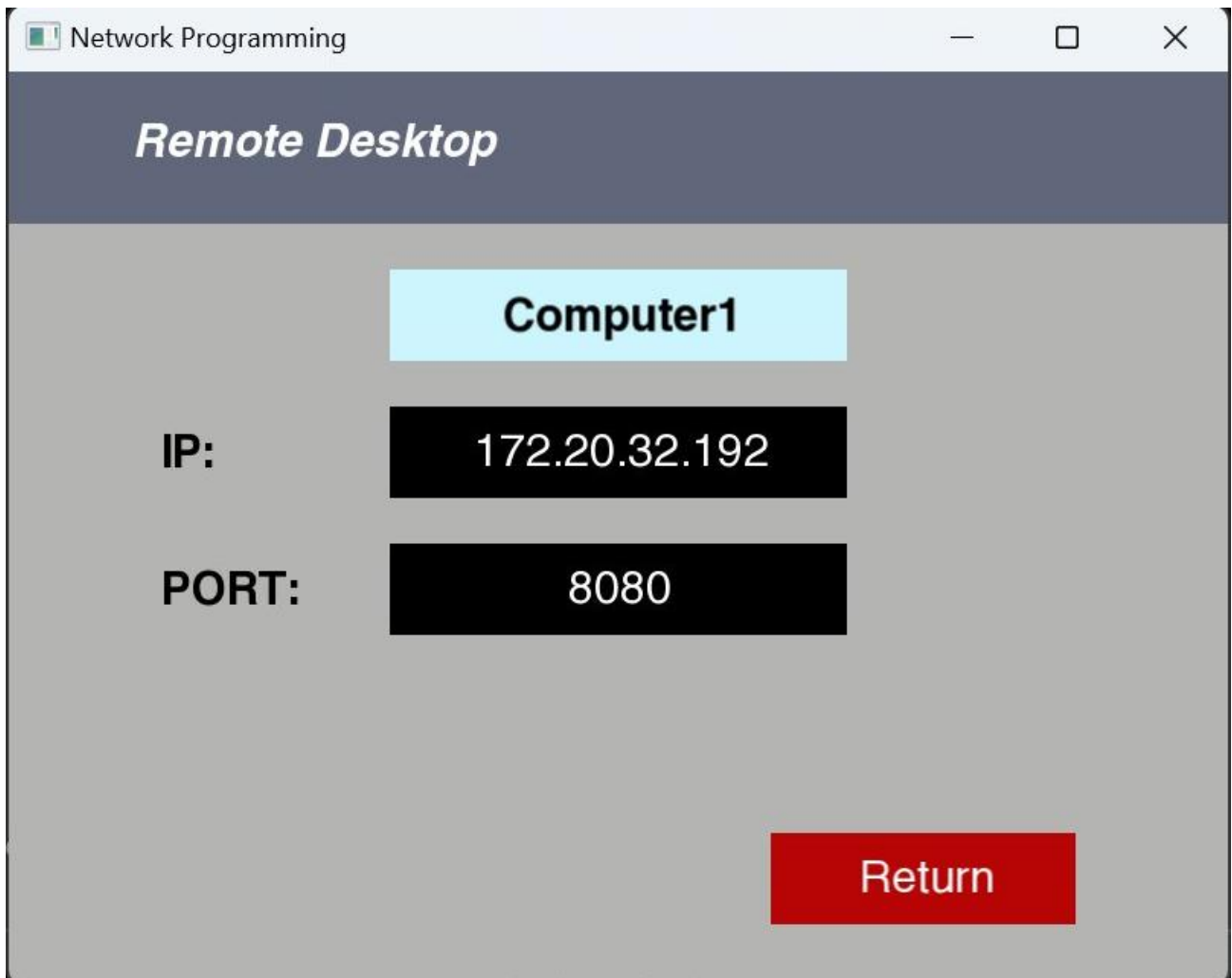


Hình 2.32 – Giao diện slide 1 - Trạng thái on

### 2.3.3.4 Slide 2

Phần giao diện hiển thị sau khi đã kết nối thành công với server. Màn hình hiển thị gồm:

- Header: Remote Desktop
- Body
  - Thông tin server: Tên máy server, địa chỉ IP, port.
  - Nút Return: Trở lại trang ban đầu kể kết nối với máy khác.



Hình 2.33 – Giao diện slide2

### ***Cài đặt mã nguồn***

Lớp Slide2 gồm:

- Thuộc tính:
  - Vị trí: **posX, posY, padding**.
  - Thành phần của slide 2: **buttonNameComputer, buttonIp, textIp buttonPort, textPort buttonReturn**.
  - Thông tin port, ip: **infoServer**.
- Phương thức:
  - Các phương thức **set, get** tương ứng với các thuộc tính trên.
  - Phương thức **run**: Thực thi việc xây dựng và hiển thị giao diện:
    - \* Lấy thông tin về máy chủ thông qua **getnameinfo**.
    - \* Xây dựng và hiển thị các thành phần: header, tên máy, địa chỉ IP, port và nút Return.
    - \* Canh và bắt sự kiện click vào nút Return, nếu nút bị click sẽ trở lại trang ban đầu để tiếp tục các thao tác kết nối với máy khác.

```

class Slide2 {
private:
    // Position of objects in graphics
    float posX;
    float posY;
    float padding;

    // Header
    Button* header;

    // Button
    Button* buttonNameComputer;
    Button* buttonIp;
    Button* buttonPort;
    Button* buttonReturn;

    // Ip + port information
    std::pair<std::string, int> infoServer;

    // Text ip + port
    sf::Text textIp;
    sf::Text textPort;
public:
    Slide2();
    ~Slide2();

    // Set, Get method of PosX
    void setPosX(float posX);
    float getPosX();

    // Set, Get method of PosY
    void setPosY(float posY);
    float getPosY();

    // Set, Get method of PosY
    void setPosY(float posY);
    float getPosY();

    // Set, Get method of Padding
    void setPadding(float padding);
    float getPadding();

    // Set, Get method of Header
    void setHeader(Button* header);
    Button* getHeader();

    // Set, Get method of buttonNameComputer
    void setButtonNameComputer(Button* buttonNameComputer);
    Button* getButtonNameComputer();

    // Set, Get method of buttonIp;
    void setButtonIp(Button* buttonIp);
    Button* getButtonIp();

    // Set, Get method of buttonPort
    void setButtonPort(Button* buttonPort);
    Button* getButtonPort();

    // Set, Get method of buttonReturn
    void setButtonReturn(Button* buttonReturn);
    Button* getButtonReturn();

    // Set, Get method of information server
    void setInfoServer(std::pair<std::string, int> infoServer);
    std::pair<std::string, int> getInfoServer();

    // Set, Get method of text ip + port
    void setTextIp(sf::Text textIp);
    sf::Text getTextIp();
    void setTextPort(sf::Text textPort);
    sf::Text getTextPort();

    // Run
    void run(sf::RenderWindow& window, Client client);
};

```

Hình 2.34 – Cấu trúc class Slide2

## Chương 3

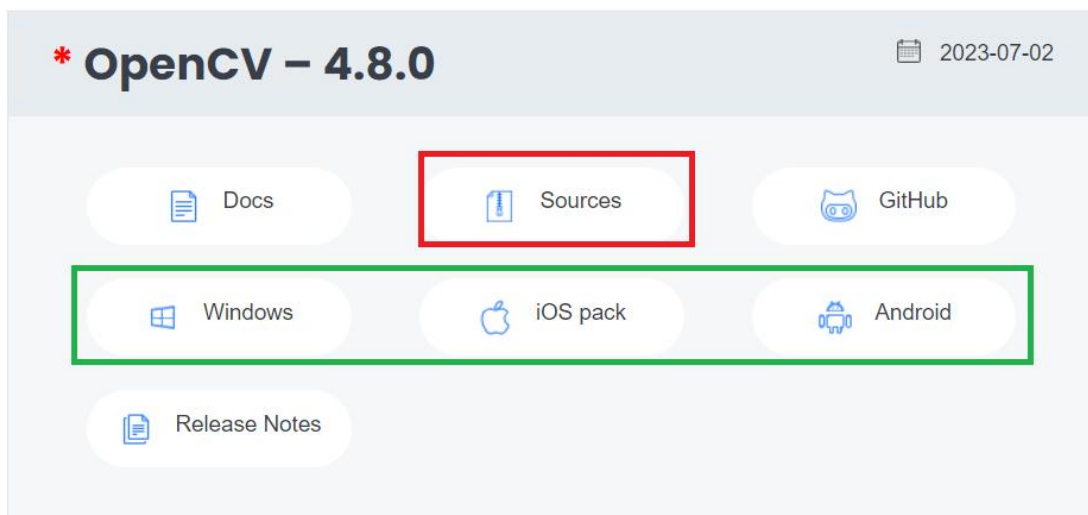
# Hướng dẫn sử dụng

### 3.1 Cài đặt thư viện

Để có thể sử dụng được chương trình, ta cần cài đặt môi trường của hai thư viện gồm **OpenCV2** và **SFML**.

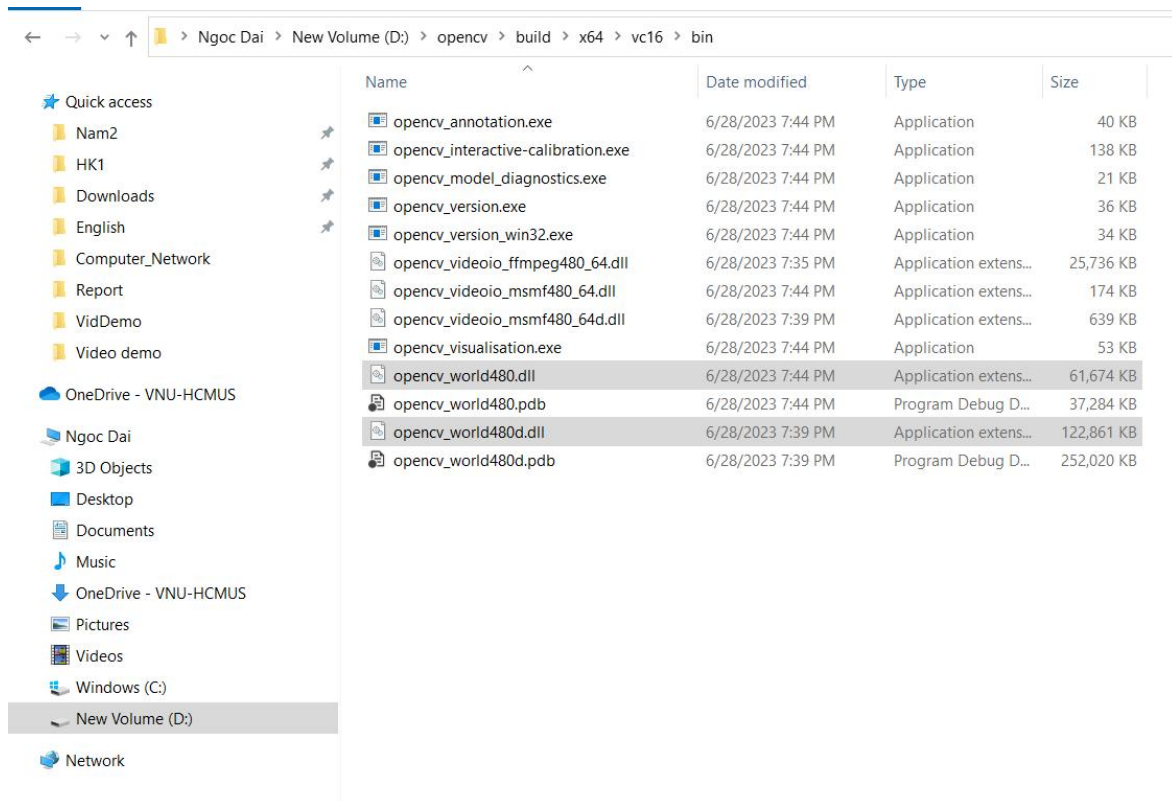
#### 3.1.1 OpenCV2

- Tải sources của thư viện OpenCV2 tại <https://opencv.org/releases/>. Chọn phiên bản tải phù hợp với máy và tải về. Thực hiện các bước chạy file thực thi và giải nén như thông thường.



Hình 3.1 – Hướng dẫn cài đặt OpenCV2

- Truy cập vào thư mục **bin** trong folder vừa giải nén xong. Thông thường, đường dẫn có dạng: `..\opencv\build\x64\vc16\bin`.

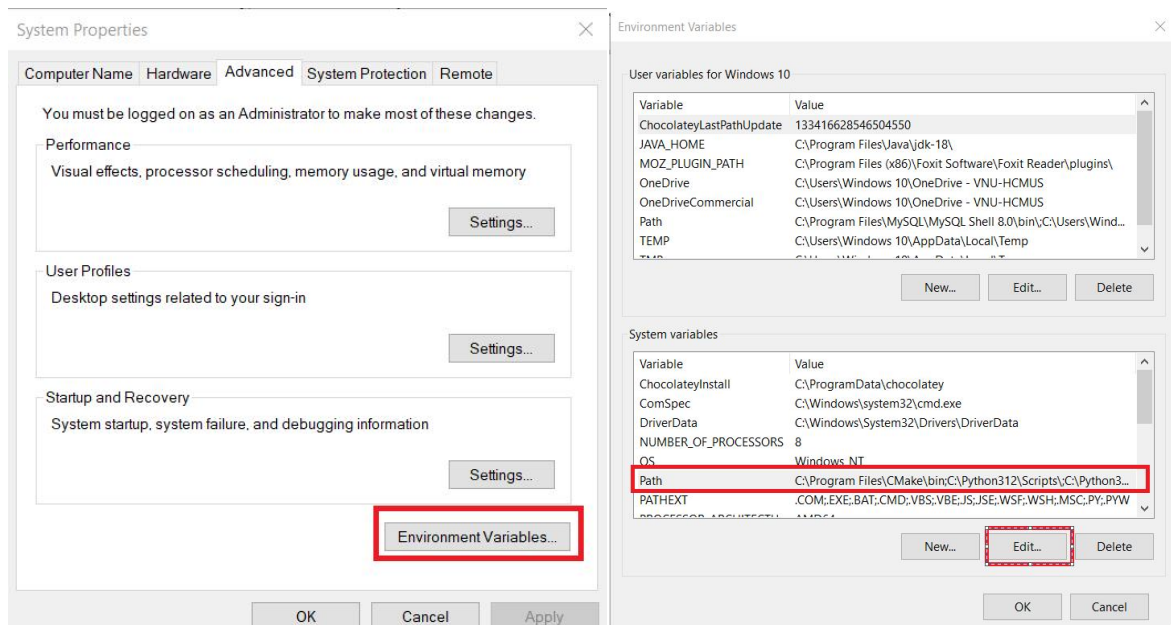


Hình 3.2 – Hướng dẫn cài đặt OpenCV2

- Cài đặt thư mục này vào máy bằng cách:

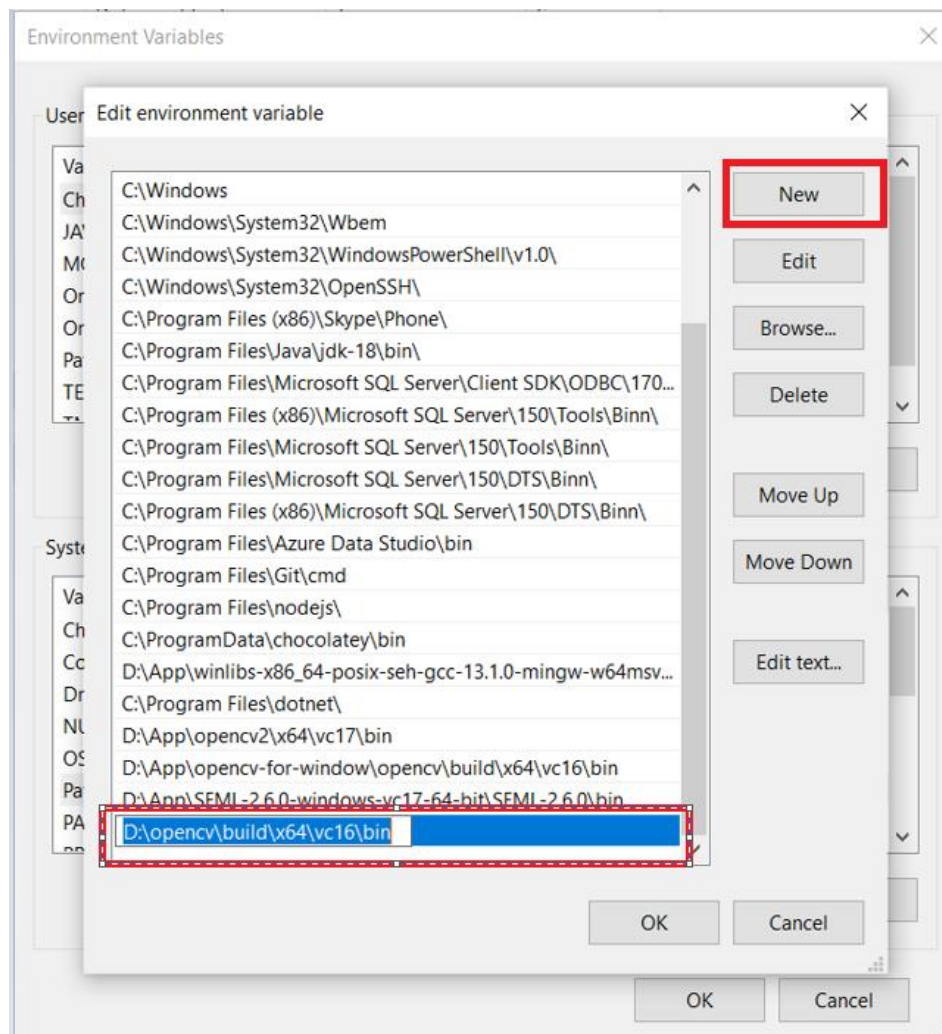
➤ **Cách 1:** Thêm đường dẫn file này vào **Environment variable** (hướng dẫn chi tiết có thể tham khảo tại

<https://www.computerhope.com/issues/ch000549.htm>)



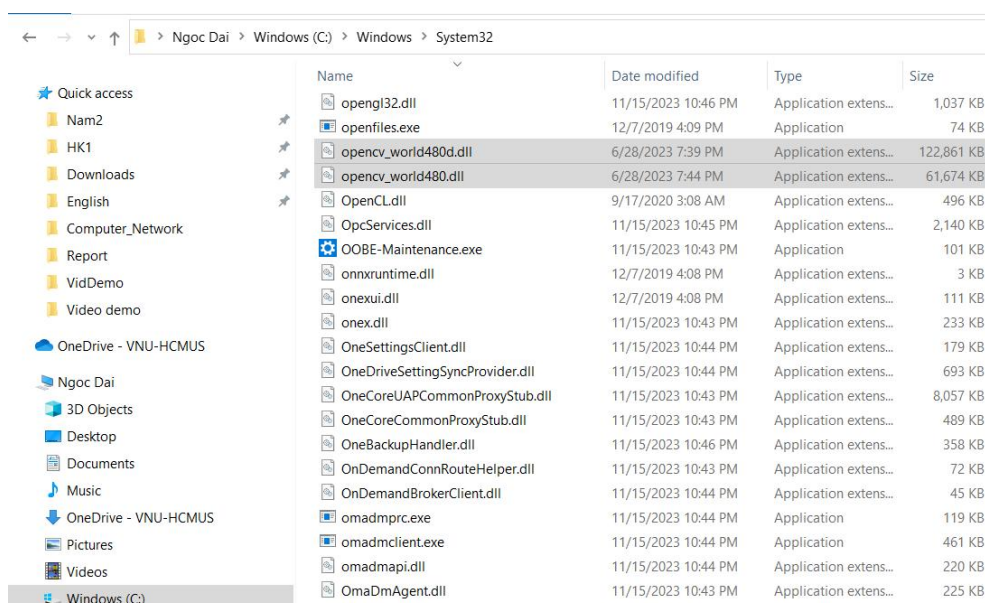
Hình 3.3 – Hướng dẫn cài đặt OpenCV2





Hình 3.4 – Hướng dẫn cài đặt OpenCV2

- **Cách 2:** Thêm các file **opencv\_world480.dll** và **opencv\_world480d.dll** trong thư mục bin vào trong folder System32 (đường dẫn thông thường **C:\Windows\System32**).



Hình 3.5 – Hướng dẫn cài đặt OpenCV2

### 3.1.2 SFML

- Tải sources của thư viện SFML tại <https://www.sfml-dev.org/download/sfml/2.6.1/>. Chọn phiên bản tải phù hợp với máy và tải về (trong hướng dẫn này, sử dụng phiên bản Visual C++17 (2022) - 64bit). Thực hiện các bước chạy file thực thi và giải nén như thông thường.




















Hình 3.6 – Hướng dẫn cài đặt SFML

- Truy cập vào thư mục **bin** trong folder vừa giải nén xong. Thông thường, đường dẫn có dạng `..\SFML-2.6.1-windows-vc17-64-bit\SFML-2.6.1\bin`.
- Cài đặt thư mục này vào máy bằng cách thêm vào folder System32 hoặc thêm path vào trong Environment variable tương tự cách cài đặt OpenCV2.

### 3.1.3 Lưu ý về cài đặt môi trường

- Các file font chữ **.ttf** và file thực thi **.exe** phải đặt chung trong một folder.
- Trong trường hợp không cài đặt được môi trường, có thể đặt file thực thi **.exe** chung với các file môi trường **.dll** trong cùng một folder và chạy file thực thi như thông thường (file môi trường **.dll** có kèm theo trong folder source).



Name	Date modified	Type	Size
 Client.exe	12/19/2023 12:08 AM	Application	91 KB
 Server.exe	12/17/2023 9:37 AM	Application	25 KB
 openal32.dll	10/14/2023 9:55 AM	Application extens...	654 KB
 opencv_videoio_ffmpeg480_64.dll	6/28/2023 7:35 PM	Application extens...	25,736 KB
 opencv_videoio_msmf480_64.dll	6/28/2023 7:44 PM	Application extens...	174 KB
 opencv_videoio_msmf480_64d.dll	6/28/2023 7:39 PM	Application extens...	639 KB
 opencv_world480.dll	6/28/2023 7:44 PM	Application extens...	61,674 KB
 sfml-audio-2.dll	10/14/2023 9:55 AM	Application extens...	1,159 KB
 sfml-audio-d-2.dll	10/14/2023 9:55 AM	Application extens...	1,741 KB
 sfml-graphics-2.dll	10/14/2023 9:55 AM	Application extens...	871 KB
 sfml-network-2.dll	10/14/2023 9:55 AM	Application extens...	125 KB
 sfml-system-2.dll	10/14/2023 9:55 AM	Application extens...	49 KB
 sfml-window-2.dll	10/14/2023 9:55 AM	Application extens...	139 KB
 SwanseaBold-D0ox.ttf	12/8/2023 4:16 PM	TrueType font file	52 KB
 SwanseaBoldItalic-p3Dv.ttf	12/8/2023 4:16 PM	TrueType font file	51 KB
 SwanseaItalic-AwqD.ttf	12/8/2023 4:16 PM	TrueType font file	53 KB
 Swansea-q3pd.ttf	12/8/2023 4:16 PM	TrueType font file	53 KB

Hình 3.7 – Lưu ý

## 3.2 Sử dụng chương trình

### 3.2.1 Lấy địa chỉ IP của server

Có nhiều cách khác nhau để lấy địa chỉ IP của máy. Ta có thể sử dụng cách sau:

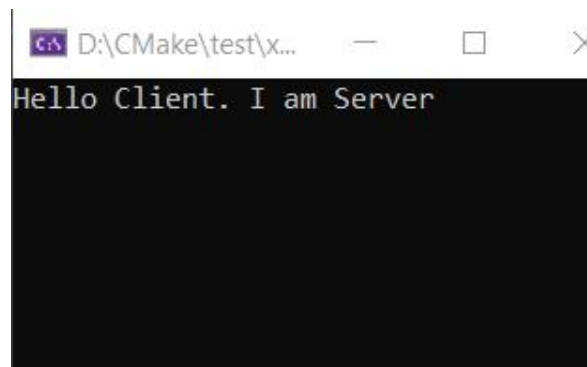
- Mở **cmd** ( Window + R, **cmd**).
- Sử dụng lệnh **ipconfig**, tìm và lấy địa chỉ IP của máy:  
Ví dụ: IPv4 Address. . . . . : 192.168.1.11

### 3.2.2 Tại server

- Mở file **server.exe**  
Cấp quyền truy cập nếu file bị hệ thống bảo vệ của máy tính chặn.

Name	Date modified	Type	Size
Server.exe	12/17/2023 9:37 AM	Application	25 KB
opengl32.dll	10/14/2023 9:55 AM	Application extens...	654 KB
opencv_videoio_ffmpeg480_64.dll	6/28/2023 7:35 PM	Application extens...	25,736 KB
opencv_videoio_msmf480_64.dll	6/28/2023 7:44 PM	Application extens...	174 KB
opencv_videoio_msmf480_64d.dll	6/28/2023 7:39 PM	Application extens...	639 KB
opencv_world480.dll	6/28/2023 7:44 PM	Application extens...	61,674 KB
sfml-audio-2.dll	10/14/2023 9:55 AM	Application extens...	1,159 KB
sfml-audio-d-2.dll	10/14/2023 9:55 AM	Application extens...	1,741 KB
sfml-graphics-2.dll	10/14/2023 9:55 AM	Application extens...	871 KB
sfml-network-2.dll	10/14/2023 9:55 AM	Application extens...	125 KB
sfml-system-2.dll	10/14/2023 9:55 AM	Application extens...	49 KB
sfml-window-2.dll	10/14/2023 9:55 AM	Application extens...	139 KB
SwanseaBold-D0ox.ttf	12/8/2023 4:16 PM	TrueType font file	52 KB
SwanseaBoldItalic-p3Dv.ttf	12/8/2023 4:16 PM	TrueType font file	51 KB
SwanseaItalic-AwqD.ttf	12/8/2023 4:16 PM	TrueType font file	53 KB
Swansea-q3pd.ttf	12/8/2023 4:16 PM	TrueType font file	53 KB

Hình 3.8 – Hướng dẫn sử dụng chương trình tại Server.



Hình 3.9 – Màn hình console tại Server.

- Để ngắt kết nối ta chỉ cần tắt màn hình console.

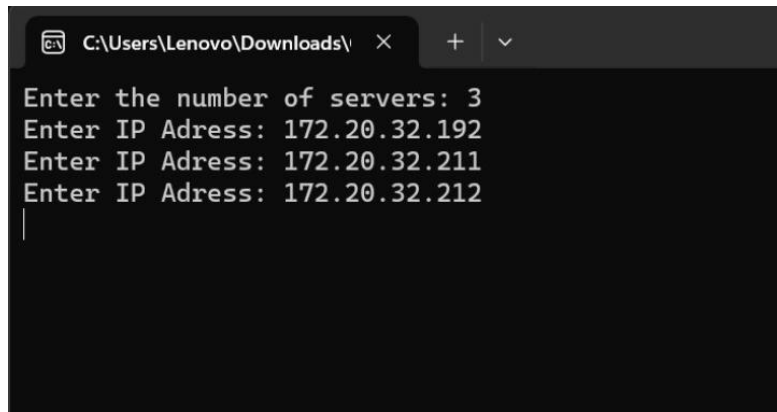
### 3.2.3 Tại Client

- Mở file Client.exe

Name	Date modified	Type	Size
Client.exe	12/19/2023 12:08 AM	Application	91 KB
opengl32.dll	10/14/2023 9:55 AM	Application extens...	654 KB
opencv_videoio_ffmpeg480_64.dll	6/28/2023 7:35 PM	Application extens...	25,736 KB
opencv_videoio_msmf480_64.dll	6/28/2023 7:44 PM	Application extens...	174 KB
opencv_videoio_msmf480_64d.dll	6/28/2023 7:39 PM	Application extens...	639 KB
opencv_world480.dll	6/28/2023 7:44 PM	Application extens...	61,674 KB
sfml-audio-2.dll	10/14/2023 9:55 AM	Application extens...	1,159 KB
sfml-audio-d-2.dll	10/14/2023 9:55 AM	Application extens...	1,741 KB
sfml-graphics-2.dll	10/14/2023 9:55 AM	Application extens...	871 KB
sfml-network-2.dll	10/14/2023 9:55 AM	Application extens...	125 KB
sfml-system-2.dll	10/14/2023 9:55 AM	Application extens...	49 KB
sfml-window-2.dll	10/14/2023 9:55 AM	Application extens...	139 KB
SwanseaBold-D0ox.ttf	12/8/2023 4:16 PM	TrueType font file	52 KB
SwanseaBoldItalic-p3Dv.ttf	12/8/2023 4:16 PM	TrueType font file	51 KB
SwanseaItalic-AwqD.ttf	12/8/2023 4:16 PM	TrueType font file	53 KB
Swansea-q3pd.ttf	12/8/2023 4:16 PM	TrueType font file	53 KB

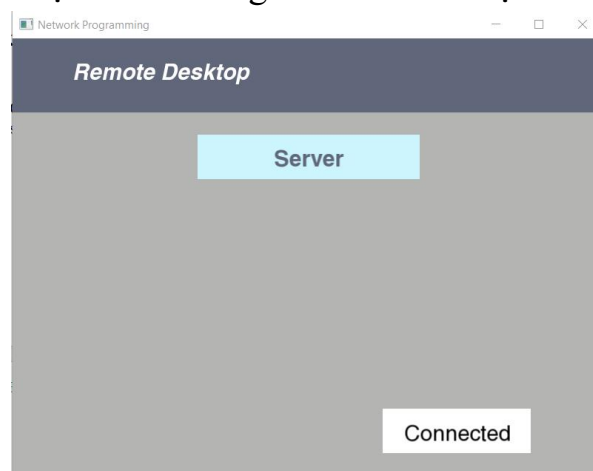
Hình 3.10 – Hướng dẫn sử dụng chương trình tại Client

- Tại màn hình console, ta lần lượt nhập số lượng máy chủ và lần lượt nhập địa chỉ IP tương ứng.



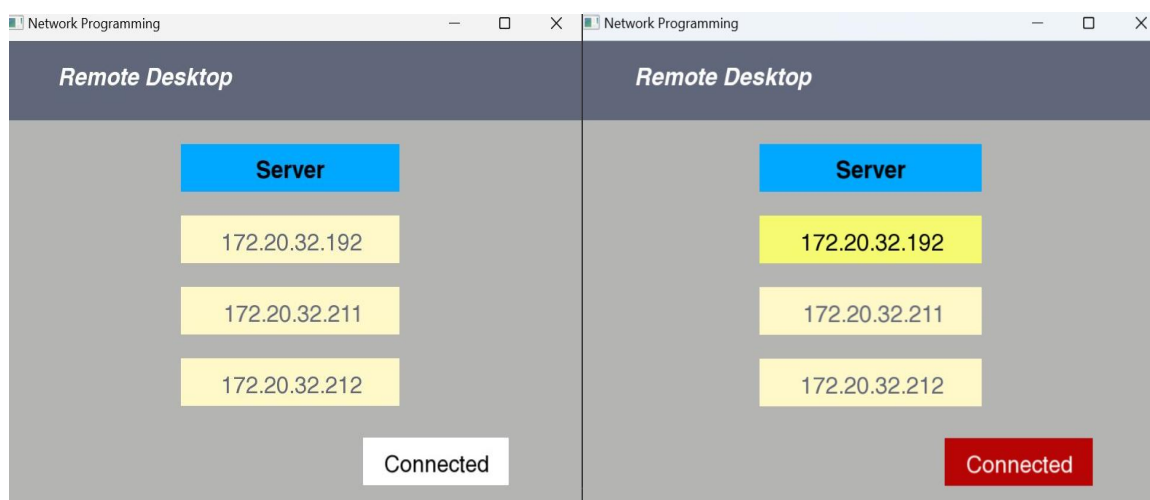
Hình 3.11 – Hướng dẫn sử dụng chương trình tại Client

- Sau đó, cửa sổ giao diện của chương trình sẽ hiển thị.



Hình 3.12 – Hướng dẫn sử dụng chương trình tại Client

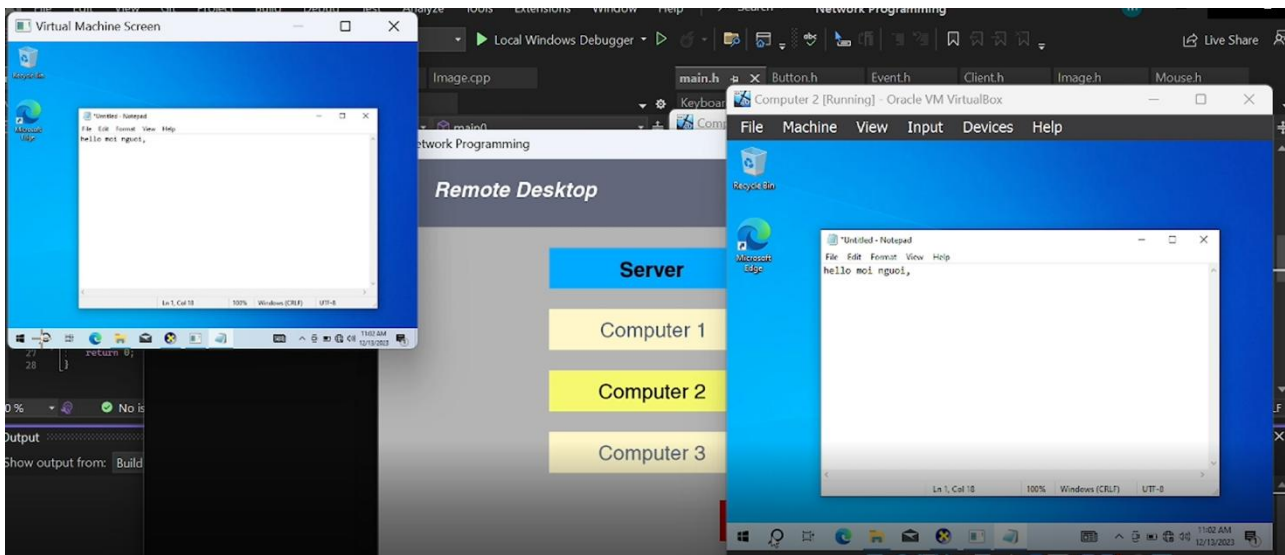
- Click vào nút **Server** để hiển thị ra các nút tương ứng với các server mà ta đã nhập địa chỉ IP trước đó. Chọn máy muốn kết nối, và click vào nút **Connected** để kết nối.



Hình 3.13 – Hướng dẫn sử dụng chương trình tại Client

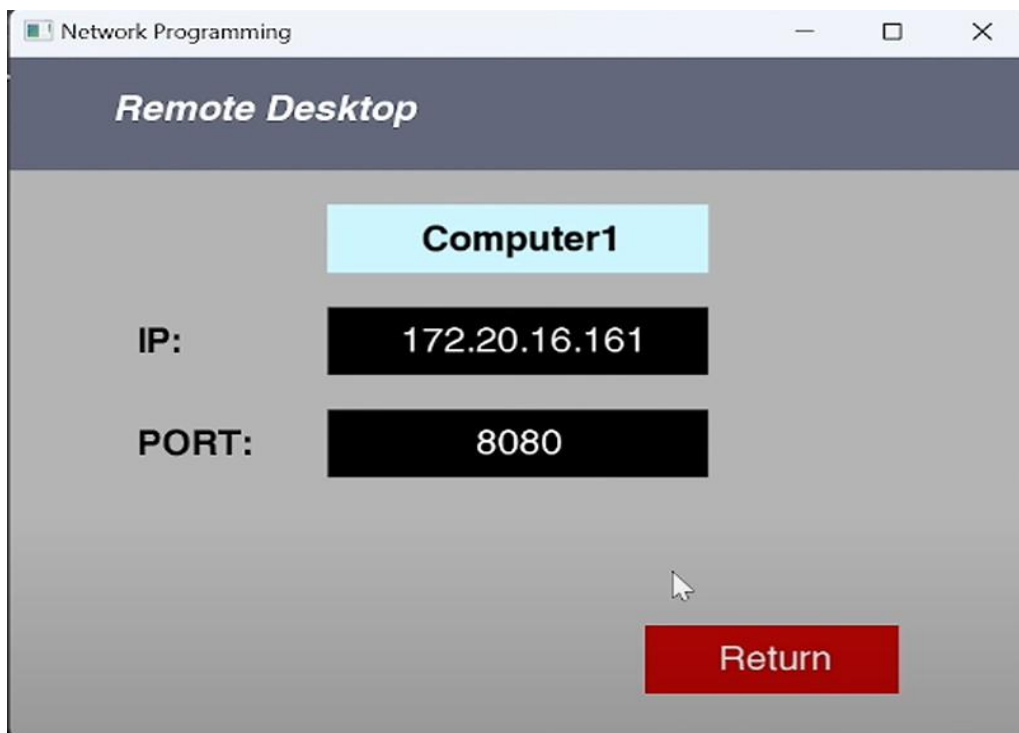
- Kết nối thành công, màn hình của server sẽ được hiển thị tại client và ta có thể

thực hiện điều khiển các thao tác chuột, bàn phím thông qua cửa sổ này.



Hình 3.14 – Hướng dẫn sử dụng chương trình tại Client

- Sau khi ngắt kết nối, cửa sổ điều khiển sẽ hiển thị, trên cửa sổ có các thông tin về server vừa kết nối: Tên máy, địa chỉ IP, Port kết nối.
- Để trở lại trang ban đầu, và kết nối với các máy khác, ta click vào **Return**



Hình 3.15 – Hướng dẫn sử dụng chương trình tại Client

# Tài liệu tham khảo

- [1] Xử Lý Ảnh Với OpenCV Trong C++ Cho Người Mới Bắt Đầu
- [2] How can I encode and decode the depth image with opencv? I use the following code to get a depth image in webots, but fail. Thanks a lot
- [3] sf::RectangleShape Class Reference
- [4] Text and fonts
- [5] Text at center of rect
- [6] [MFC] Http request with winsock2.h
- [7] How can I take a screenshot in a windows application?
- [8] Xử Lý Sự Kiện Chuột và Phím với Borland Graphics Interface - BGI
- [9] GetAsyncKeyState function (winuser.h)
- [10] Cannot found any data type used by SendInput() function