TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

Võ Trung Hoàng Hưng - 21120011
Nguyễn Phúc Tân - 21120028
Hồ Lê Minh Quân - 21120314

ĐIỀU KHIỂN MÁY TÍNH TỪ XA

ĐỒ ÁN MÔN HỌC: MẠNG MÁY TÍNH
CHƯƠNG TRÌNH CỬ NHÂN TÀI NĂNG

GIÁO VIÊN HƯỚNG DẪN
Thầy Đỗ Hoàng Cường
Cô Huỳnh Thụy Bảo Trân
Cô Chung Thùy Linh

Tp. Hồ Chí Minh, tháng 05/2023

Lời cảm ơn

Chúng em xin trân trọng cảm ơn các thầy cô:

- Thầy Đỗ Hoàng Cường
- Cô Huỳnh Thụy Bảo Trân
- Cô Chung Thùy Linh

đã tận tình hỗ trợ chúng em qua bài giảng trên lớp, tài liệu, source code demo, và trả lời các thắc mắc của chúng em trong quá trình thực hiện đồ án này.

Mục lục

Danh sách hình

Danh sách bảng

Chương 1

Giới thiệu

1.1 Tổng quan

Ứng dụng cho phép người dùng (sử dụng máy client) điều khiển một máy tính từ xa (máy server) trong cùng mạng LAN với máy Server. Ứng dụng sử dụng **socket**, giao thức **TCP** ở tầng Transport, và được lập trình bằng ngôn ngữ C++. Dưới đây là các chức năng của ứng dụng:

- Kết nối đến máy Server thông qua địa chỉ IP và Port number.
- Liệt kê thông tin các tiến trình (processes) đang chạy trên máy Server
- Liệt kê thông tin các ứng dụng (applications) đang chạy trên máy Server
- Chụp màn hình máy Server
- Bắt phím nhấn trên máy Server
- Duyệt cây thư mục trong máy Server

1.2 Về socket

Socket là là một giao diện lập trình ứng dụng (API) cho việc giao tiếp mạng giữa các ứng dụng mạng. Socket cung cấp cơ chế gửi và nhận dữ liệu, thiết lập kết nối và quản lý các thông tin liên quan đến mạng như địa chỉ IP và Port number.

Ứng dụng sử dụng socket hướng kết nối: dựa trên giao thức TCP, việc truyền dữ liệu chỉ thực hiện giữa 2 quá trình đã thiết lập kết nối. Giao thức này đảm bảo dữ liệu được truyền đến nơi nhận một cách đáng tin cậy, đúng thứ tự nhờ vào cơ chế quản lý luồng lưu thông trên mạng và cơ chế chống tắc nghẽn. Đồng thời, mỗi thông điệp gửi phải có xác nhận trả về và các gói tin chuyển đi tuần tự.

1.3 Công cụ

1.3.1 Microsoft Foundation Classes

MFC là một bộ công cụ phát triển ứng dụng của Microsoft dành cho ngôn ngữ lập trình C++. Nó cung cấp một tập hợp các lớp và thành phần để phát triển các ứng dụng Windows.

Lớp CSocket từ thư viện **afxsock.h** của MFC đóng vai trò là socket sử dụng trong quá trình lập trình ứng dụng này. Bên cạnh đó, ứng dụng còn sử dụng thư viện **afxwin.h** của MFC để hỗ trợ việc tạo giao diện.

1.3.2 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.

1.4 Các link

1.4.1 Link video demo

`https://youtu.be/bAyCufM2_0o`

1.4.2 Link mã nguồn

`https://github.com/minhquanBr2/HCMUS-remote-controller-with-socket.git`

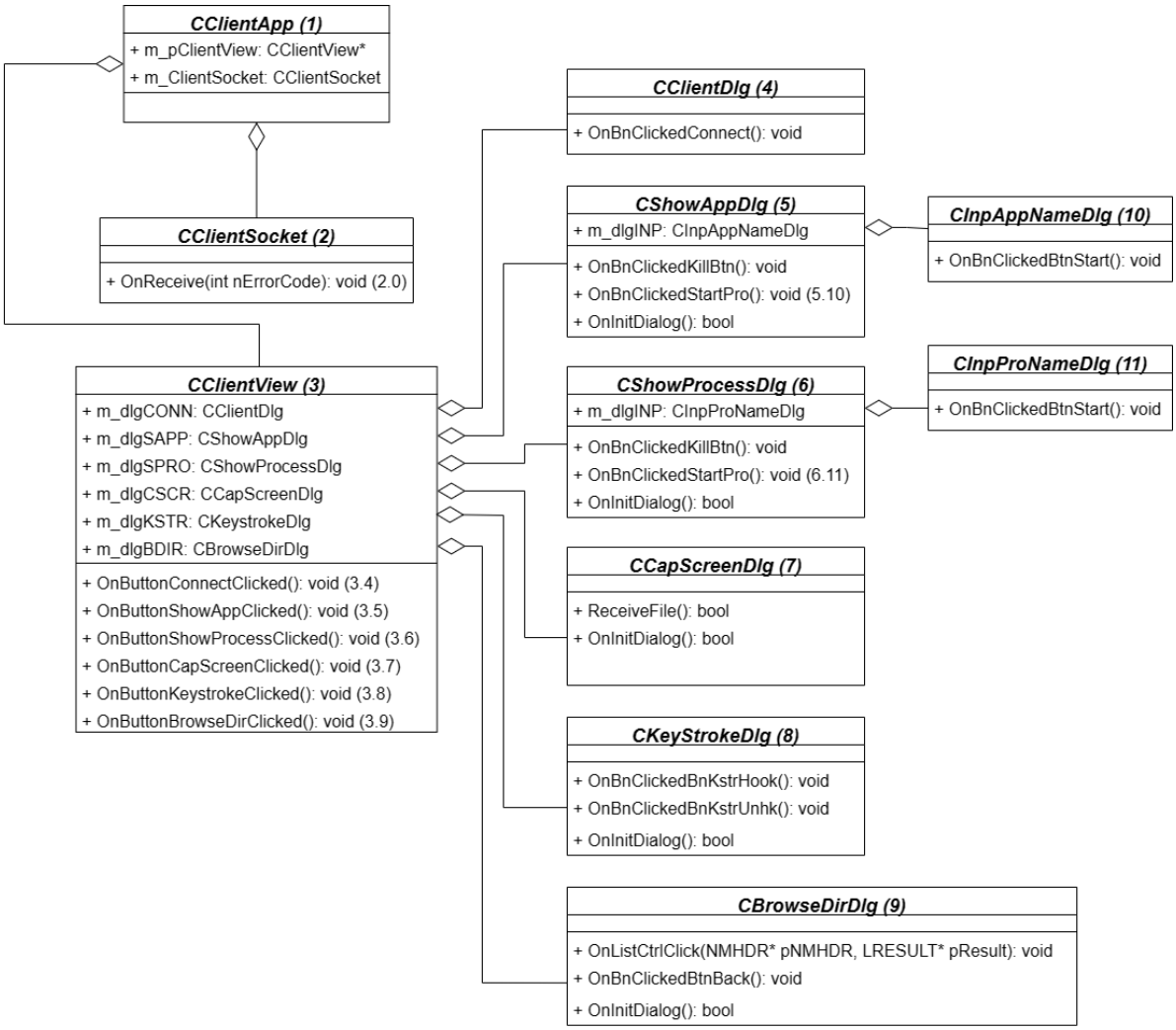
Chương 2

Tổ chức mã nguồn

2.1 Tổng quan

Trên thực tế, ứng dụng điều khiển từ xa này gồm 2 ứng dụng riêng biệt dành cho 2 máy khác nhau: client và server. App Server gần như không có giao diện, vì chỉ có vai trò gửi nhận và thực hiện các yêu cầu mà client gửi đến. Ngược lại, app Client được tổ chức với giao diện gồm các nút nhằm thực hiện các chức năng giao tiếp khác nhau với server tùy nhu cầu của người dùng.

2.2 App Client



Hình 2.1: Sơ đồ lớp của ứng dụng cho client

Trong sơ đồ bên trên, các lớp dùng để cài đặt giao diện cho app (bao gồm các dialog)

được vẽ với các dấu mũi tên thể hiện quan hệ HAS-A (tức là một lớp có biến thành phần là đối tượng của lớp khác). **CClientApp** và **CClientView** là các lớp điều khiển giao diện, **CClientSocket** là lớp dùng để cài đặt socket, các lớp còn lại từ **(4)** đến **(11)** cài đặt các thao tác trên các dialog ứng với các công việc khác nhau. Cụ thể:

- **(4)** - `CClientDlg`: Dialog quản lý việc nhập thông tin địa chỉ IP và port của server.
- **(5)** - `CShowAppDlg`: Dialog quản lý các tác vụ liên quan đến các ứng dụng của server.
- **(6)** - `CShowProcessDlg`: Dialog quản lý các tác vụ liên quan đến các tiến trình của server.
- **(7)** - `CCapScreenDlg`: Dialog quản lý tác vụ chụp ảnh màn hình server.
- **(8)** - `CKeystrokeDlg`: Dialog quản lý tác vụ bắt phím nhấn của server.
- **(9)** - `CBrowseDirDlg`: Dialog quản lý tác vụ duyệt cây thư mục của server.
- **(10)** - `CInpAppNameDlg`: Dialog quản lý tác vụ nhập tên một ứng dụng và khởi động ứng dụng đó ở phía server.
- **(11)** - `CInpProNameDlg`: Dialog quản lý tác vụ nhập tên một tiến trình và khởi động tiến trình đó ở phía server.

Một số phương thức khi được gọi cũng sẽ làm xuất hiện các dialog liên quan. Cụ thể:

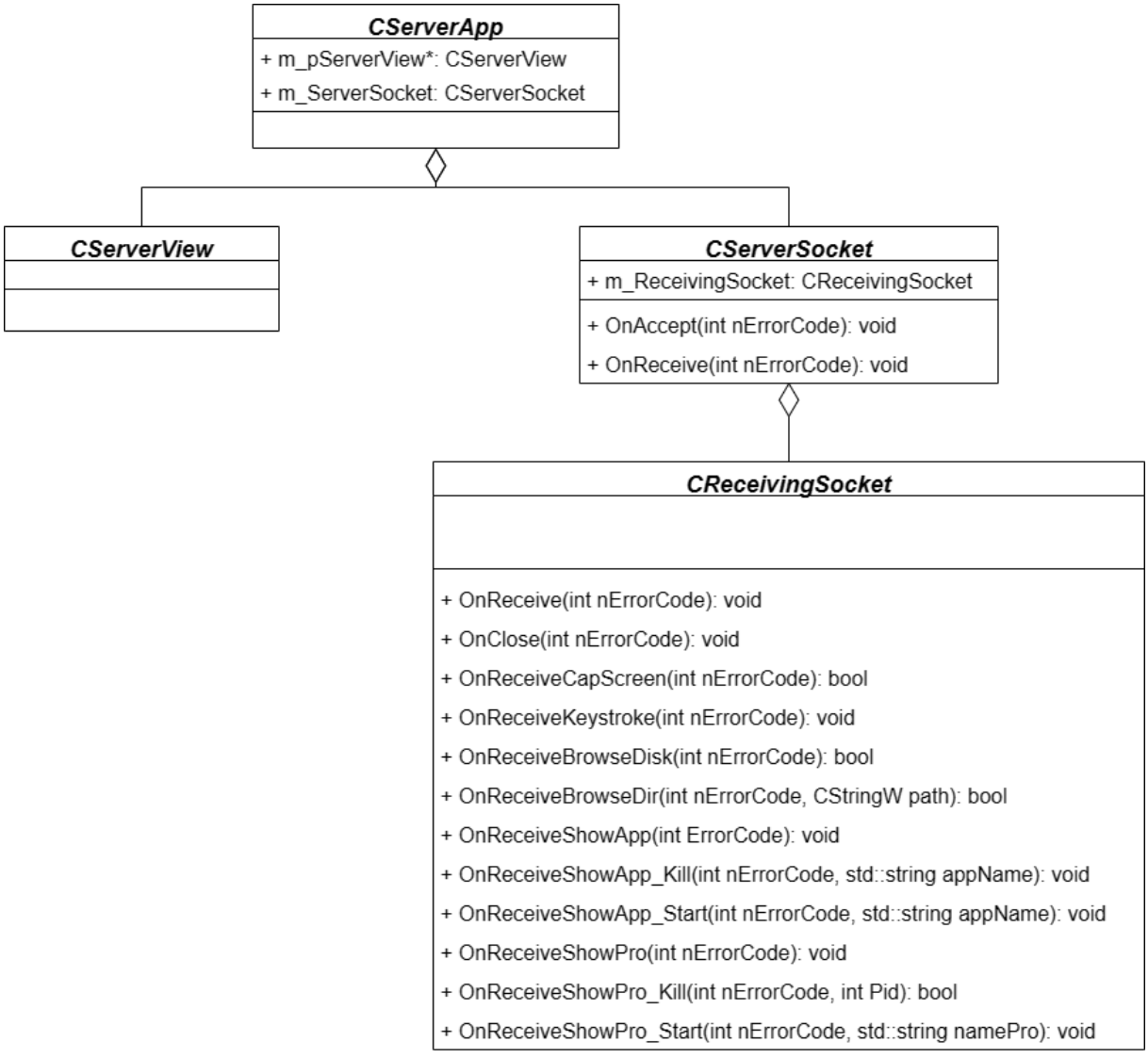
- **(3.4)** - `OnButtonConnectClicked` gọi dialog **(4)**.
- **(3.5)** - `OnButtonShowAppClicked` gọi dialog **(5)**.
- **(3.6)** - `OnButtonShowProcessClicked` gọi dialog **(6)**.
- **(3.7)** - `OnButtonCapScreenClicked` gọi dialog **(7)**.
- **(3.8)** - `OnButtonKeystrokeClicked` gọi dialog **(8)**
- **(3.9)** - `OnButtonBrowseDirClicked` gọi dialog **(9)**.
- **(5.10)** - `OnBnClickedStartApp` gọi dialog **(10)**.
- **(6.11)** - `OnBnClickedStartPro` gọi dialog **(11)**.

2.3 App Server

Bên dưới là sơ đồ mô tả các lớp bên trong ứng dụng dành cho server. Giống như app Client, app Server cũng có các lớp **CServerApp** và **CServerView** là cho cài đặt giao diện, **CServerSocket** dùng để cài đặt socket. Đặc biệt có thêm lớp **CReceivingSocket** là một socket riêng để giao tiếp với app Client vừa kết nối, bao gồm gửi nhận dữ liệu từ app Client.

Khi socket của server nhận thông điệp từ client thông qua phương thức `OnReceive`, server sẽ tùy nội dung thông điệp mà thực hiện công việc tương ứng bằng cách gọi các phương thức bắt đầu bằng tiền tố `OnReceive`. Một số thông điệp ứng với các công việc là:

- `REQ_SAPP`: liệt kê các ứng dụng (show app).
- `REQ_SAPP_START`: khởi động một ứng dụng.
- `REQ_SAPP_KILL`: tắt một ứng dụng.
- `REQ_SPRO`: liệt kê các tiến trình (show process).
- `REQ_SPRO_START`: khởi động một tiến trình.
- `REQ_SPRO_KILL`: tắt một tiến trình.
- `REQ_CSCR`: chụp màn hình (capture screen).
- `REQ_KSTR`: bắt phím nhấn (keystroke).
- `REQ_BDIR`: duyệt cây thư mục (browse directory).



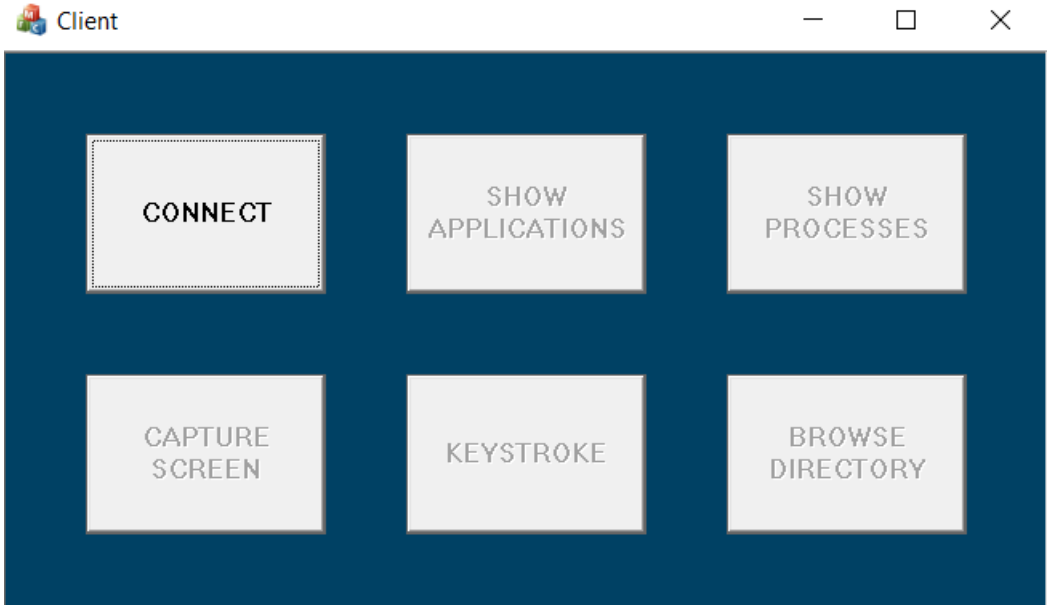
Hình 2.2: Sơ đồ lớp của ứng dụng cho server

Chương 3

Các chức năng của phần mềm

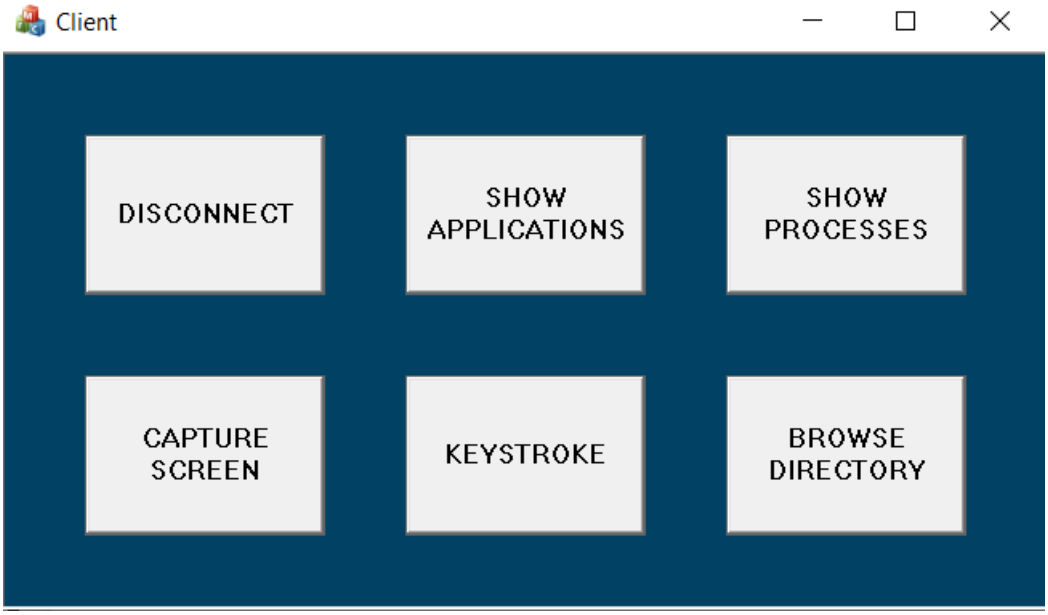
3.1 Mô tả giao diện app Client

Giao diện ban đầu của app Client được gồm 6 nút nhằm thực hiện các chức năng giao tiếp với Server khác nhau tùy nhu cầu của người dùng (Hình 3.1).



Hình 3.1: Giao diện mặc định của app Client khi chưa được kết nối

Mặc định khi khởi động app, các nút không phải nút **CONNECT** bị vô hiệu hóa, và hộp thoại **Connect** sẽ hiện ra để người dùng nhập thông tin địa chỉ IP và port của app Server mà họ muốn kết nối tới. Sau khi kết nối thành công, nút **CONNECT** đổi tên thành **DISCONNECT**, các nút khác mới có thể được sử dụng (Hình 3.2).

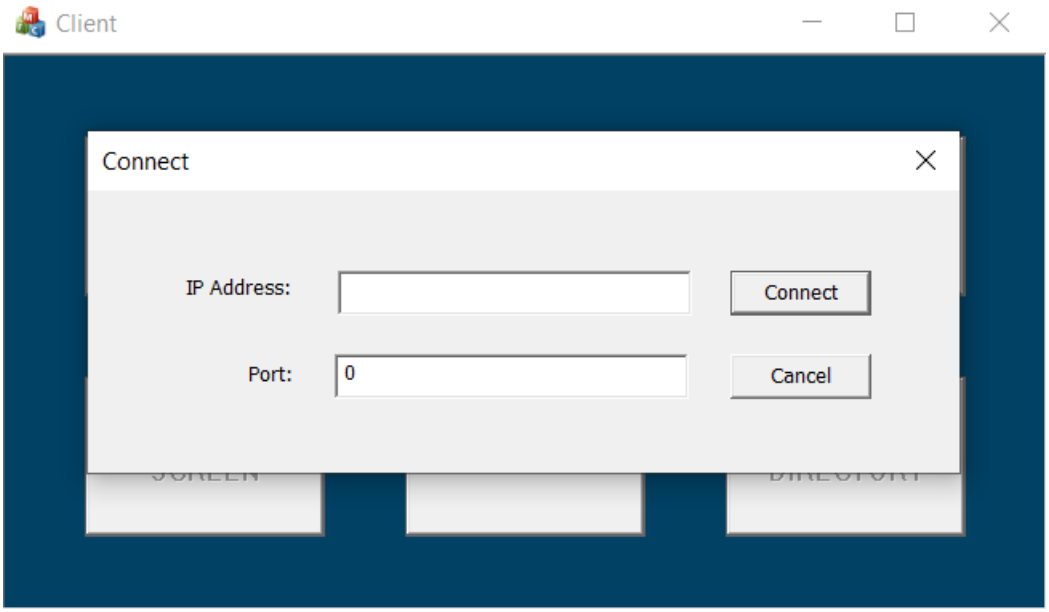


Hình 3.2: Giao diện của app Client khi đã được kết nối

Sau khi thực hiện xong một chức năng, người dùng có thể tắt hộp thoại điều khiển chức năng đó và bấm nút khác để thực hiện chức năng khác.

3.2 Kết nối đến Server (nút CONNECT)

3.2.1 Hướng dẫn sử dụng



Hình 3.3: Hộp thoại Connect yêu cầu nhập địa chỉ IP và port của máy Server để kết nối

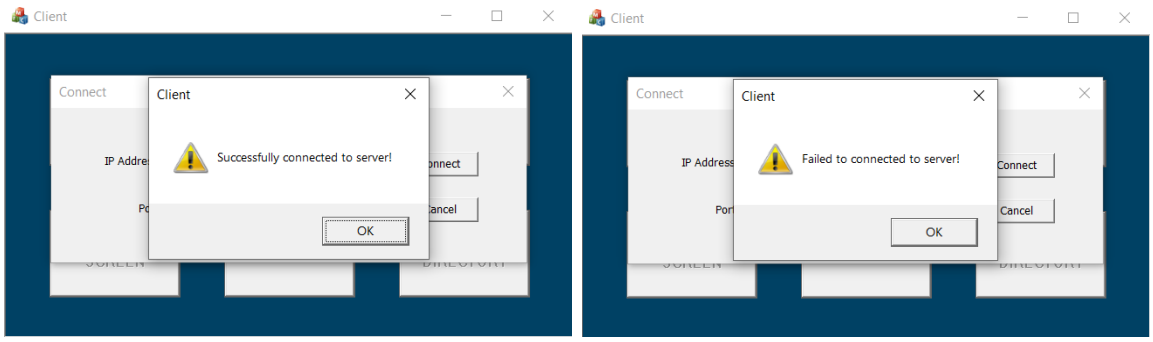
Hộp thoại **Connect** có thể xuất hiện mặc định cùng lúc khởi động chương trình, hoặc khi người dùng nhấn vào nút **CONNECT** (Hình 3.3).

Người dùng nhập địa chỉ IP và port của app Server vào rồi nhấn **Connect** để kết nối. Nếu kết nối thất bại, có thể thử lại vô hạn lần cho đến khi kết nối thành công. Luôn

có hộp thoại xuất hiện để thông báo kết nối thành công hoặc thất bại sau mỗi lần người dùng thử (Hình 3.4).

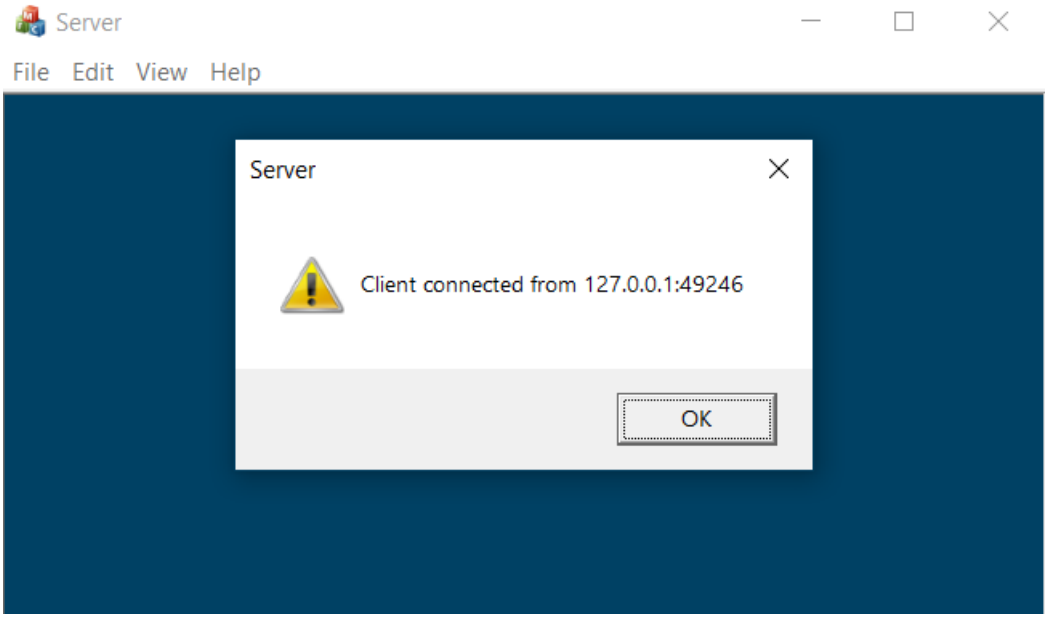
Lưu ý về địa chỉ IP khi kết nối:

- Kết nối 2 ứng dụng trên 2 máy khác nhau trong cùng mạng: người dùng phía Client cần biết trước IP của máy Server. IP của máy Server có thể lấy ra bằng lệnh `ipconfig` của **Command Prompt**. Port là 6666 vì đã được khởi tạo trong mã nguồn.
- Kết nối 2 ứng dụng trên cùng 1 máy: ngoài cách trên, còn có thể nhập địa chỉ loopback **127.0.0.1**. Đây là địa chỉ IP đặc biệt được dùng để tham chiếu đến chính máy tính đang sử dụng. Khi gửi dữ liệu đến địa chỉ loopback, dữ liệu sẽ được gửi và xử lý trên cùng một máy tính mà không cần thông qua mạng vật lý.



Hình 3.4: Hộp thoại Connect sau khi kết nối thành công/thất bại

Sau khi kết nối thành công, phía app Server cũng sẽ hiển thị thông báo rằng Client đã kết nối thành công. Trong thông báo có thông tin về địa chỉ IP và port của app Client (Hình 3.5).



Hình 3.5: Hộp thoại phía app Server thông báo kết nối thành công

3.2.2 Một số hàm đã được cài đặt

App Server

1. **Phương thức tạo socket**

```
m_ServerSocket.Create(6666);
```

`m_ServerSocket` là một member variable của lớp **CServerSocket**. Hàm này tạo một socket có port là 6666 (mặc định được MFC chọn ngẫu nhiên) và được quản lý bởi biến `m_ListCtrl`. Từ đó, socket sẽ lắng nghe hoạt động của client trên tất cả các card mạng.

2. **Phương thức lắng nghe các yêu cầu kết nối**

```
m_ServerSocket.Listen();
```

3. **Phương thức chấp nhận một yêu cầu kết nối từ Client**

```
m_ServerSocket.Accept(m_ReceivingSocket);
```

Ở đây `m_ReceivingSocket` là một socket mới, được tạo ra để xử lý giao tiếp với máy Client, đại diện cho kết nối đã được chấp nhận. Nó khác với `m_ServerSocket` là socket có vai trò lắng nghe và chờ các kết nối đến.

App Client

App Client làm việc với biến `m_ClientSocket` được quản lý bởi lớp CClientDlg (lớp quản lý các đối tượng thuộc hộp thoại Connect). Khi nút CONNECT được nhấn, phương thức `OnBnClickedConnect()` được kích hoạt, bên trong có gọi những phương thức sau.

1. **Phương thức tạo socket**

```
m_ClientSocket.Create(6666);
```

Chú ý cần kiểm tra xem socket đã được tạo hay chưa trước mỗi lần gọi hàm Connect() để tránh gặp lỗi.

2. **Phương thức kết nối đến Server**

```
m_ClientSocket.Connect(m_strIPAddress, m_iPort);
```

Trong đó `m_strIPAddress` là địa chỉ IP của Server còn `m_iPort` là port của socket lắng nghe kết nối trên Server.

Ngoài ra một số phương thức của CSocket thường xuyên được sử dụng xuyên suốt ứng dụng.

‘

1. **Phương thức truyền/nhận dữ liệu**

```
virtual int CSocket::Send/Receive(const void* lpBuf, int
    nBufLen, int nFlags = 0);
```

- `lpBuf`: vùng đệm chứa dữ liệu.

- `nBuffLen`: kích thước của vùng đệm tính theo byte.
- `nFlag`: cách nhận/truyền dữ liệu, sử dụng giá trị mặc định là 0.
- Giá trị trả về là số byte nhận/truyền được.

2. **Phương thức lấy tên socket**

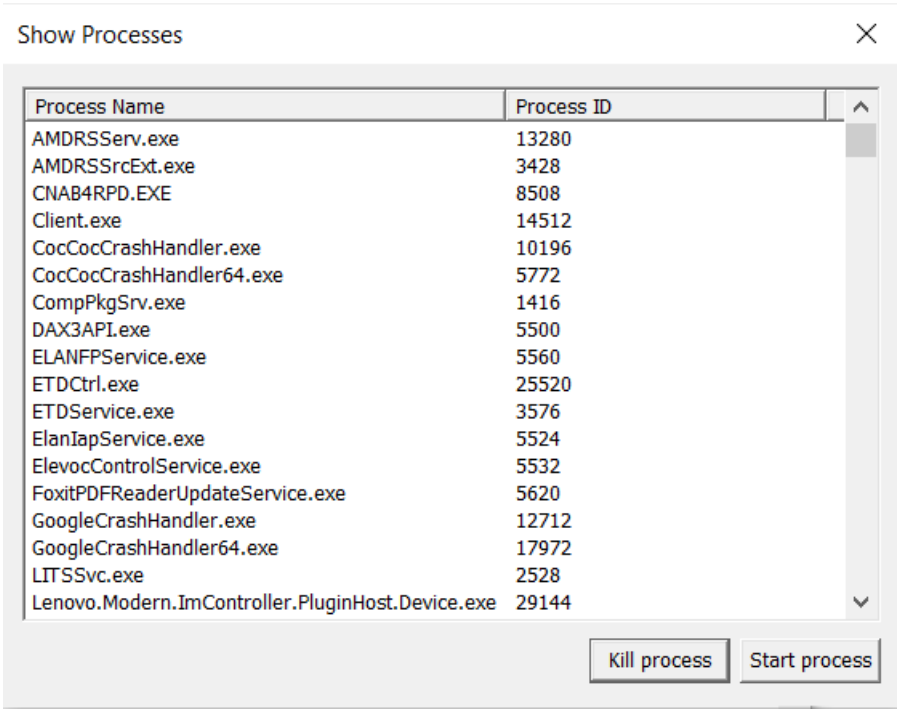
```
BOOL CAsyncSocket::GetSockName(SOCKADDR* lpSockAddr, int
    * lpSockAddrLen);
```

- `lpSockAddr`: con trỏ đến địa chỉ của socket.
- `lpSockAddrLen`: con trỏ đến độ dài địa chỉ của socket.
- Phương thức này được gọi mỗi khi người dùng ấn các nút trên màn hình app để kiểm tra tình trạng kết nối của socket.

3.3 List thông tin các tiến trình đang chạy trên Server (nút SHOW PROCESSES)

3.3.1 Hướng dẫn sử dụng

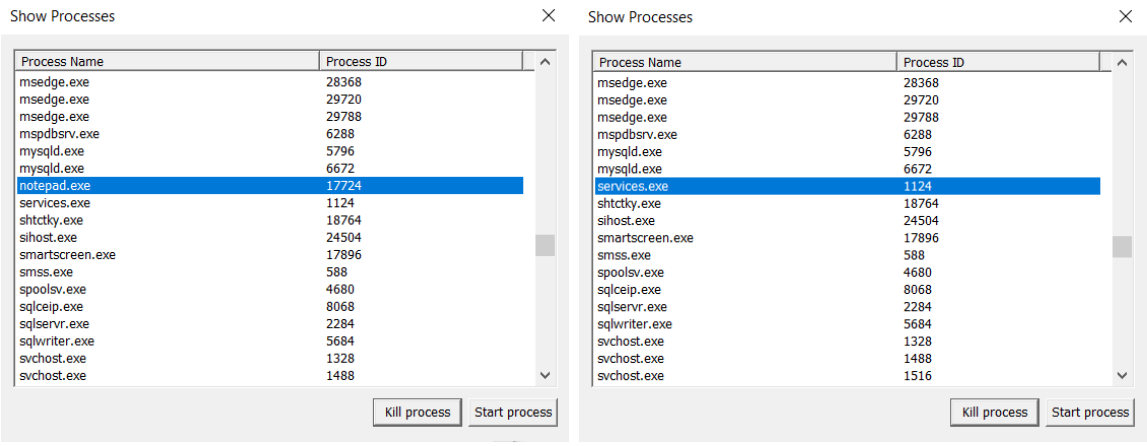
Người dùng phía Client nhấn nút SHOW PROCESSES. Hộp thoại Show Processes sẽ hiện ra và liệt kê các processes đang chạy trên máy Server (Hình 3.6).



Hình 3.6: Thông tin các process

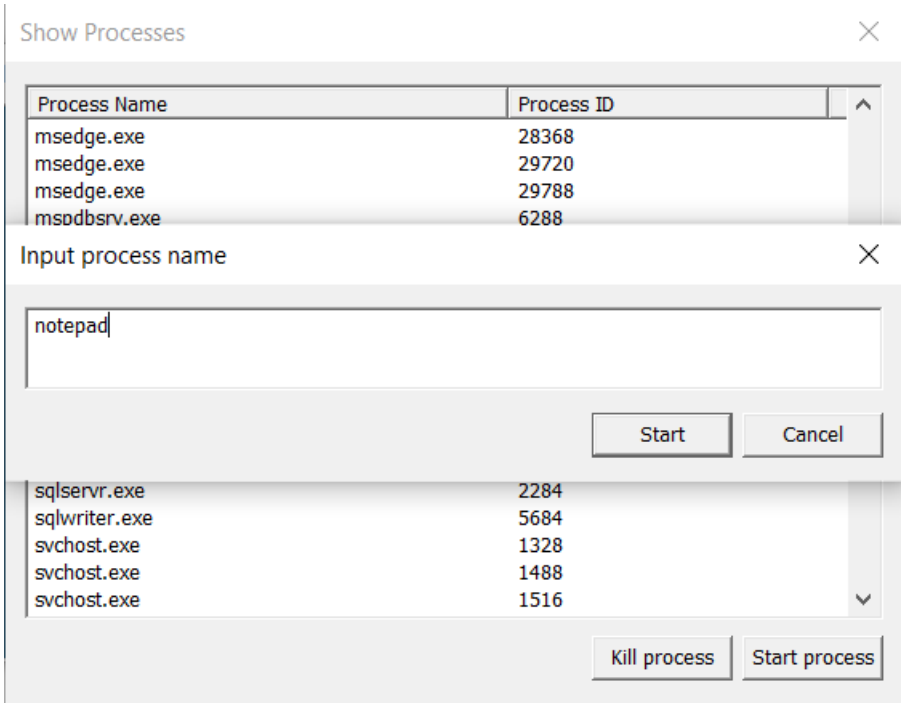
Bên cạnh đó có 2 chức năng đi kèm:

- **Tắt một process trên máy Server**: Nhấp chuột trái vào process muốn xóa trên danh sách, nhấn nút Kill process (Hình 3.7).



Hình 3.7: Tắt một process

- **Khởi động một process trên máy Server**: Nhấn nút Start process, nhập tên process muốn khởi động vào hộp thoại hiện ra, nhấn nút Start (Hình 3.8).



Hình 3.8: Khởi động một process

3.3.2 Một số hàm đã được cài đặt

1. **Phương thức gửi yêu cầu thực hiện chức năng liệt kê ứng dụng (Client)**

```
1 void CClientView::OnButtonShowProcessClicked()
```

Phương thức này gửi yêu cầu liệt kê danh sách các tiến trình tới máy Server (sử dụng phương thức `Send()`), sau đó nhận danh sách đó từ máy Server (sử dụng phương thức `Receive()`). Sau đó gọi hộp thoại để in ra danh sách tiến trình.

2. **Phương thức hiển thị danh sách ứng dụng lên hộp thoại (Client)**

```
bool CShowProcessDlg::OnInitDialog()
```

Phương thức này lấy danh sách tiến trình lấy được từ Server hiển thị lên hộp thoại trên máy Client.

3. **Phương thức liệt kê các ứng dụng trên máy Server và gửi về máy Client (Server)**

```
void CReceivingSocket::OnReceiveShowPro(int nErrorCode)
```

Sử dụng các hàm `Process32First` và `Process32Next` để trích xuất thông tin của các processes đang chạy trên máy Server(các hàm này thuộc `Windows Tool Help Library` - một phần của `Windows API` - thư viện `TLHelp32`). Sau đó gửi danh sách tiến trình lấy được về máy Client (sử dụng phương thức `Send()`).

4. **Phương thức gửi yêu cầu thực hiện chức năng dừng một ứng dụng (Client)**

```
void CShowProcessDlg::OnBnClickedKillBtn()
```

Phương thức lấy thông tin `Port ID` của tiến trình đang được chọn trên danh sách, gửi kèm lệnh yêu cầu dừng tiến trình này về máy Server (dùng phương thức `Send()`).

5. **Phương thức dừng tiến trình (Server)**

```
void CReceivingSocket::OnReceiveShowPro_Kill(int
    nErrorCode, int Pid)
```

Nhận thông tin `Port ID` của tiến trình cần dừng (tham số **int Pid**). Gọi hàm `TerminateProcess()` của Windows API để dừng tiến trình.

6. **Phương thức gửi yêu cầu thực hiện chức năng khởi động một tiến trình (Client)**

```
void CInpProNameDlg::OnBnClickedBtnStart()
```

Phương thức gửi tên tiến trình (người dùng nhập) và lệnh yêu cầu bật tiến trình đến máy Server (dùng phương thức `Send()`).

7. **Phương thức bắt đầu một tiến trình (Server)**

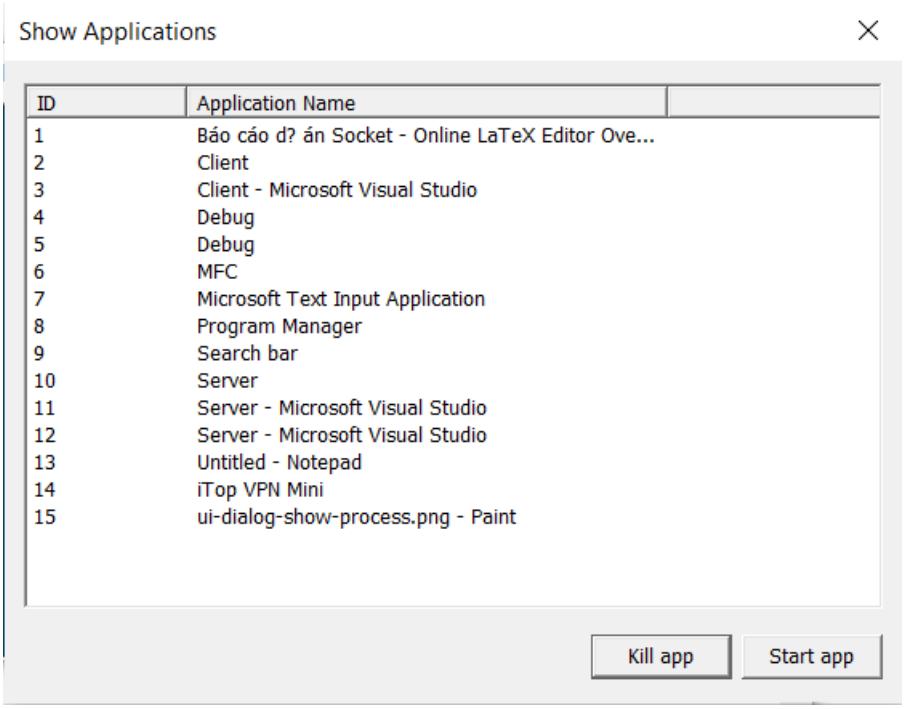
```
void CReceivingSocket::OnReceiveShowPro_Start(int
    nErrorCode, std::string namePro)
```

Nhận thông tin tên của tiến trình cần khởi động (tham số **std::string namePro**). Gọi hàm CreateProcessA() của Windows API để khởi động tiến trình.

3.4 List thông tin các ứng dụng đang chạy trên Server (nút SHOW APPLICATIONS)

3.4.1 Hướng dẫn sử dụng

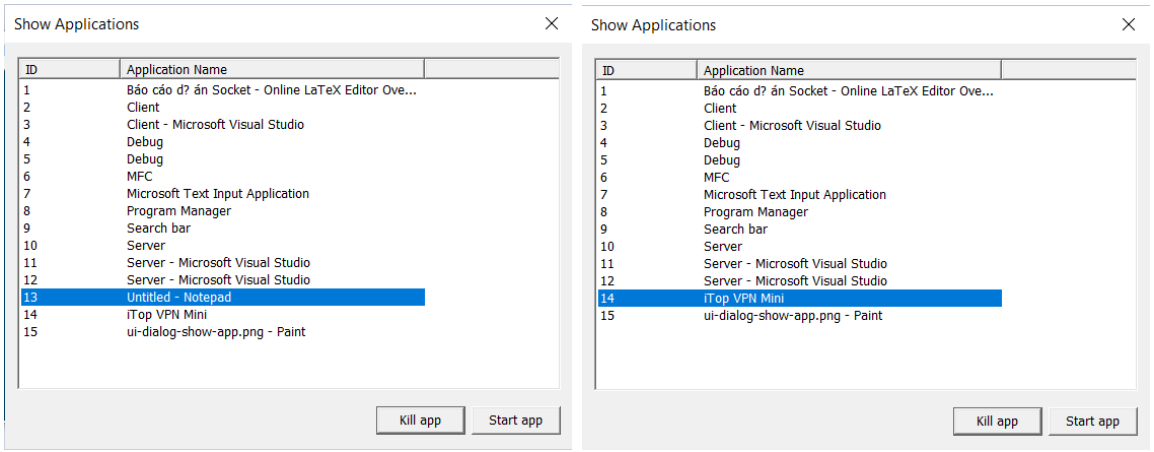
Người dùng phía Client nhấn nút SHOW APPLICATIONS. Hộp thoại Show Applications sẽ hiện ra và liệt kê các ứng dụng đang chạy trên máy Server (Hình 3.9).



Hình 3.9: Thông tin các ứng dụng

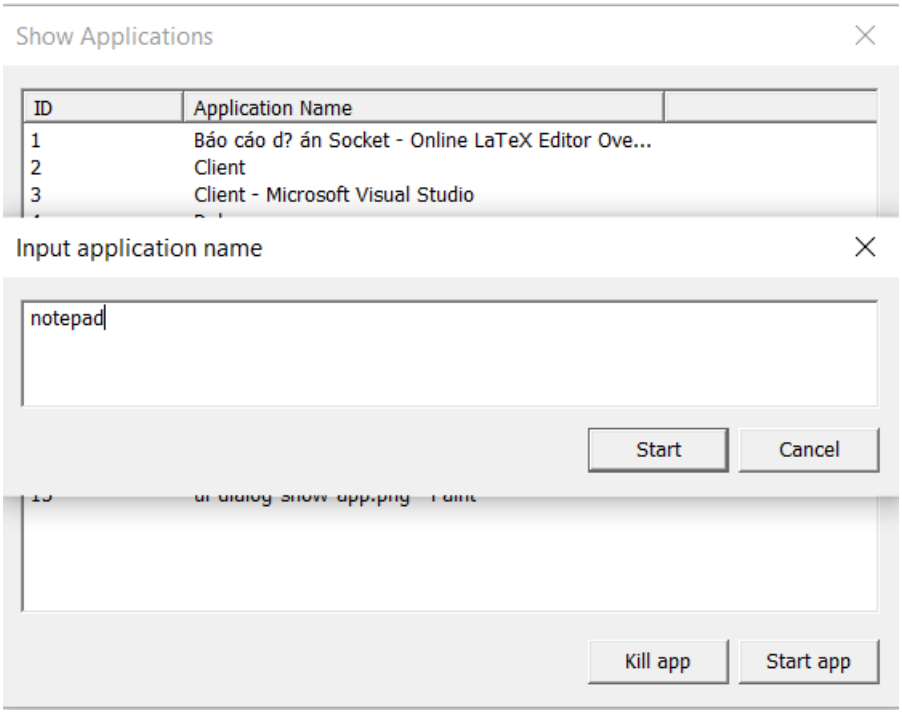
Bên cạnh đó có 2 chức năng đi kèm:

- **Tắt một ứng dụng trên máy Server**: Nhấp chuột trái vào ứng dụng muốn xóa trên danh sách, nhấn nút Kill app (Hình 3.10).



Hình 3.10: Tắt một ứng dụng

- **Khởi động một ứng dụng trên máy Server**: Nhấn nút Start application, nhập tên ứng dụng muốn khởi động vào hộp thoại hiện ra, nhấn nút Start (Hình 3.11).



Hình 3.11: Khởi động một ứng dụng

3.4.2 Một số hàm đã được cài đặt

1. **Phương thức gửi yêu cầu thực hiện chức năng liệt kê ứng dụng (Client)**

```
void CClientView::OnButtonShowAppClicked()
```

Phương thức này gửi yêu cầu liệt kê danh sách các ứng dụng đang chạy tới máy Server (sử dụng phương thức `Send()`), sau đó nhận danh sách đó từ máy Server (sử dụng phương thức `Receive()`). Sau đó gọi hộp thoại để in ra danh sách ứng dụng.

2. **Phương thức hiển thị danh sách ứng dụng lên hộp thoại (Client)**

```
bool CShowAppDlg::OnInitDialog()
```

Phương thức này lấy danh sách ứng dụng nhận được từ Server hiển thị lên hộp thoại trên máy Client.

3. **Phương thức liệt kê các ứng dụng trên máy Server và gửi về máy Client (Server)**

```
void CReceivingSocket::OnReceiveShowApp(int ErrorCode)
```

Sử dụng hàm `GetWindowTextW()` của `Windows API` để trích xuất tên của các ứng dụng đang chạy trên máy Server. Sau đó gửi danh sách tên các ứng dụng này về máy Client (sử dụng phương thức `Send()`).

4. **Phương thức gửi yêu cầu thực hiện chức năng dừng một ứng dụng (Client)**

```
void CShowAppDlg::OnBnClickedKillBtn()
```

Phương thức lấy thông tin tên của ứng dụng đang được chọn trên danh sách, gửi kèm lệnh yêu cầu dừng ứng dụng này về máy Server(dùng phương thức `Send()`).

5. **Phương thức dừng ứng dụng (Server)**

```
void CReceivingSocket::OnReceiveShowApp_Kill(int
    nErrorCode, std::string appName)
```

Nhận thông tin tên của ứng dụng cần dừng (tham số **std::string appName**). Gọi hàm `TerminateApplication()` của Windows API để dừng ứng dụng.

6. **Phương thức gửi yêu cầu thực hiện chức năng khởi động một ứng dụng (Client)**

```
void CInpAppNameDlg::OnBnClickedBtnStart()
```

Phương thức gửi tên ứng dụng(người dùng nhập) và lệnh yêu cầu bật tiến trình đến máy Server(sử dụng phương thức `Send()`).

7. **Phương thức bắt đầu một ứng dụng (Server)**

```
void CReceivingSocket::OnReceiveShowApp_Start(int
    nErrorCode, std::string appName)
```

Nhận thông tin tên của ứng dụng cần khởi động (tham số **std::string appName**). Gọi hàm CreateProcessA() của Windows API để khởi động ứng dụng.

3.5 Chụp màn hình Server (nút CAPTURE SCREEN)

3.5.1 Hướng dẫn sử dụng

Để sử dụng chức năng này, trước hết người dùng phía Client cần nhấn nút CAPTURE SCREEN. Hộp thoại Capture Screen sẽ hiện ra, trên đó hiển thị ảnh chụp toàn màn hình của máy Server (Hình 3.12).

Hình 3.12: Chụp màn hình máy server

3.5.2 Một số hàm đã được cài đặt

1. **Phương thức gửi yêu cầu thực hiện chức năng chụp màn hình (Client)**

```
void CClientView::OnButtonCapScreenClicked()
```

Phương thức này được gọi khi người dùng phía Client nhấn nút CAPTURE SCREEN. Trước hết, ta kiểm tra xem kết nối còn nguyên hay bị mất do ảnh hưởng từ việc thực hiện các chức năng trước đó, nếu mất kết nối thì kết nối lại. Sau đó, `m_ClientSocket` gửi thông điệp `REQ_CSCR` (request cap screen) bằng phương thức `Send()` yêu cầu thực hiện công việc, rồi lần lượt nhờ biến dialog `m_dlgCSCR` gọi các phương thức `ReceiveFile()` và `DoModal()` để nhận file ảnh và hiển thị file ảnh đó trong hộp thoại.

2. **Hàm tự chụp màn hình (Server)**

```
int ScreenCapture();
```

Hàm này có nhiệm vụ chụp ảnh màn hình và lưu vào file bitmap `screenshot.bmp`. Đầu tiên, ta lấy thông tin về chiều rộng và chiều cao của màn hình. Sau đó, ta tạo ra một device context (cấu trúc dữ liệu chứa thông tin và các chức năng để tương tác với các thiết bị) có kiểu dữ liệu là `HDC` tương thích với màn hình này, rồi tiếp tục tạo một biến `HBITMAP` có kích thước tương thích với `HDC` đã tạo, dùng để lưu trữ hình ảnh được chụp màn hình. Khi việc khởi tạo được hoàn tất, hàm `BitBlt` được sử dụng để sao chép từng byte pixel của màn hình vào biến bitmap. Biến bitmap sau đó được gán cho các thông tin phù hợp với định dạng file bitmap, bao gồm `InfoHeader`, `FileHeader`, pixel array, và kết thúc với việc xuất hình ảnh ra file bitmap `screenshot.bmp`.

3. **Phương thức gửi ảnh từ Server sang Client**

```
BOOL CReceivingSocket::OnReceiveCapScreen(int nErrorCode
    );
```

Phương thức này được gọi khi socket phía Server nhận được thông điệp `REQ_CSCR` từ Client. Nó khởi đầu bằng việc gọi hàm `ScreenCapture()` để chụp ảnh màn hình và xuất ra file ảnh bitmap. Sau khi có ảnh, ta chia nhỏ ảnh thành từng đoạn có chiều dài 4096 bytes và lần lượt gửi từng đoạn sang Client.

4. **Phương thức nhận ảnh (Client)**

```
BOOL CCapScreenDlg::ReceiveFile();
```

Phương thức này được gọi khi ngay sau khi thông điệp `REQ_CSCR` được gửi đi. Nó trước hết lấy thông tin về kích thước của file từ header, và vì nhận dữ liệu của file ảnh theo từng đoạn, nó sẽ liên tục nhận dữ liệu đến khi nào tổng số byte nhận được bằng với kích thước file.

5. **Phương thức hiển thị hộp thoại bao gồm hiển thị ảnh (Client)**

```
BOOL CCapScreenDlg::OnInitDialog();
```

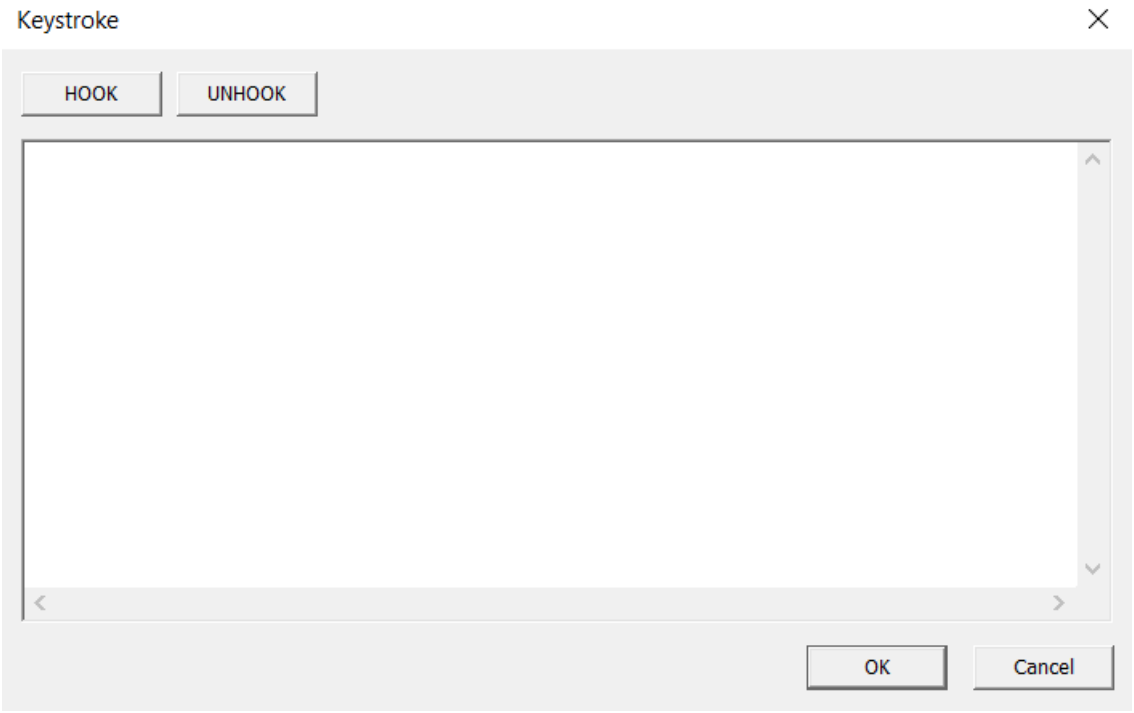
Phương thức `DoModal()` được gọi ngay sau khi ta gọi phương thức `ReceiveFile()`, yêu cầu khởi tạo và hiển thị hộp thoại. Khi đó, phương thức `OnInitDialog()` sẽ được ngầm gọi để chuẩn bị các thông tin trước khi hiển thị hộp thoại. Nó trước

hết lấy đường dẫn hình ảnh, lưu ảnh vào biến kiểu `CImage`, scale ảnh và vẽ lên cho phù hợp với kích thước khung hiển thị bằng phương thức `StretchBlt()`, rồi cuối cùng gán ảnh vào khung hiển thị bằng phương thức `SetBitmap()`. Khi hộp thoại hiện ra, vì các thao tác trên đều đã thực hiện xong, kết quả là hình ảnh sẽ được hiển thị đầy đủ trong khung ảnh của hộp thoại. Chức năng hoàn tất.

3.6 Bắt phím nhấn (nút KEYSTROKE)

3.6.1 Hướng dẫn sử dụng

Để sử dụng chức năng này, trước hết người dùng phía Client cần nhấn nút KEYSTROKE. Hộp thoại Keystroke sẽ hiện ra, với 2 nút HOOK và UNHOOK, cùng một khung text để chưa thông tin các phím nhấn ghi nhận được từ phía Server (Hình 3.13).



Hình 3.13: Hộp thoại Keystroke với các nút HOOK/UNHOOK và khung text hiển thị tên các phím bắt được từ server

Khi nhấn nút HOOK, các phím nhấn phía Server sẽ được ghi nhận lại, và hiển thị trong khung text thành từng dòng, mỗi phím một dòng (Hình 3.14).

Hình 3.14: Danh sách các phím bắt được từ Server sau một khoảng thời gian

Khi nhấn nút UNHOOK, các phím nhấn phía Server sẽ dừng được ghi nhận, khi đó khung text sẽ không hiển thị thêm kí tự nào, cho đến khi người dùng phía Client lại bấm nút HOOK để ghi nhận các phím nhấn.

3.6.2 Một số hàm đã được cài đặt

1. **Phương thức gửi yêu cầu thực hiện chức năng bắt phím nhấn (Client)**

```
void CClientView::OnButtonKeystrokeClicked()
```

   Phương thức này được gọi khi người dùng phía Client nhấn nút KEYSTROKE. Trước hết, ta kiểm tra xem kết nối còn nguyên hay bị mất do ảnh hưởng từ việc thực hiện các chức năng trước đó, nếu mất kết nối thì kết nối lại.Sau đó, `m_ClientSocket` gửi thông điệp `REQ_KSTR` (request keystroke) bằng phương thức `Send()` yêu cầu thực hiện công việc, rồi nhờ biến dialog `m_dlgCSCR` gọi phương thức `DoModal()` để hiển thị hộp thoại.

2. **Phương thức cho phép nhận phím nhấn (Client)**

```
void CKeystrokeDlg::OnBnClickedBnKstrHook();
```

   Phương thức này dùng phương thức `Send()` để gửi thông điệp `REQ_KSTR_HOOK` đến Server.

3. **Phương thức chặn nhận phím nhấn (Client)**

```
void CKeystrokeDlg::OnBnClickedBnKstrUnhk();
```

   Phương thức này dùng phương thức `Send()` để gửi thông điệp `REQ_KSTR_UNHOOK` đến Server.

4. **Phương thức điều khiển việc gửi tên phím nhấn (Server)**

```
void CReceivingSocket::OnReceiveKeystroke(int nErrorCode
    ){
     if (m_hook == NULL && m_strMsg == "REQ_KSTR_HOOK")
         m_hook = SetWindowsHookEx(WH_KEYBOARD_LL,
    KeyboardProc, NULL, 0);
     else if (m_hook != NULL && m_strMsg == "
    REQ_KSTR_UNHOOK")    {
         UnhookWindowsHookEx(m_hook);
         m_hook = NULL;
     }
}
```

Bên trong lớp `CReceivingSocket` có một biến kiểu `HHOOK` là `m_hook` được khởi tạo giá trị NULL. Ở đây, `HHOOK` là một kiểu dữ liệu đại diện cho một con trỏ tới một hook procedure trong Windows. Trong Windows, một hook là một cơ chế cho phép ứng dụng theo dõi và can thiệp vào sự kiện xảy ra trong hệ thống, chẳng hạn như sự kiện bàn phím, chuột, hoặc hành vi hệ thống. Khi một hook được đặt, hook procedure sẽ được gọi mỗi khi sự kiện tương ứng xảy ra, cho phép ứng dụng thực hiện các xử lý tùy chỉnh.

Phương thức này thông qua cấu trúc điều khiển if-else sẽ thực hiện các nhiệm vụ khác nhau tùy giá trị của thông điệp nhận được và giá trị của `m_hook`.

- Khi `m_hook` chưa được đặt và thông điệp yêu cầu đặt hook (`REQ_KSTR_HOOK`), hàm `SetWindowsHookEx()` sẽ được gọi, trả về một HHOOK, cho phép ứng dụng lưu trữ và quản lý hook đã đăng ký. `KeyboardProc` chính là hàm được gọi mỗi khi sự kiện một phím được nhấn xảy ra.
- Khi `m_hook` đã được đặt và thông điệp yêu cầu gỡ bỏ hook procedure (`REQ_KSTR_UNHOOK`) hàm `UnhookWindowsHookEx()` sẽ được gọi để gỡ hook đã đăng ký và gán nó thành NULL.

5. **Hàm cài đặt hook procedure cho sự kiện nhấn phím (Server)**

```
LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam,
    LPARAM lParam){
     ...
         CString keystrokeName = GetKeystrokeName(
    pKeyboardHookStruct->vkCode);
         if (keystrokeName != " ") {
             ((CServerApp*)AfxGetApp())->m_ServerSocket.
    m_ReceivingSocket.Send(keystrokeName.GetBuffer(
    keystrokeName.GetLength()), keystrokeName.GetLength
    ());
         }
     ...
     return CallNextHookEx(NULL, nCode, wParam, lParam);
}
```

Hàm này lấy thông tin các phím được nhấn dưới dạng virtual key code (số nguyên), sau đó đưa cho hàm `GetKeystrokeName()` chuyển thành tên phím ở dạng chuỗi kí tự và gửi thông tin này về Client. Sau khi kết thúc, hàm lại gọi hàm `CallNextHookEx` để chuyển tiếp thông điệp hook của bàn phím tới hook procedure tiếp theo trong chuỗi hook sau khi xử lý thông điệp.

6. **Hàm chuyển giá trị phím thành tên phím (Server)**

```
CString GetKeystrokeName(int vkCode);
```

Hàm này chuyển giá trị các phím ở dạy virtual key code thành số nguyên. Một số ví dụ:

- `VK_CAPITAL` ⟶ "[CAPS_LOCK]"
- `VK_RETURN` ⟶ "[ENTER]"
- `VK_LSHIFT` / `VK_RSHIFT` ⟶ "[SHIFT]"

7. **Phương thức hiển thị hộp thoại kết quả (Client)**

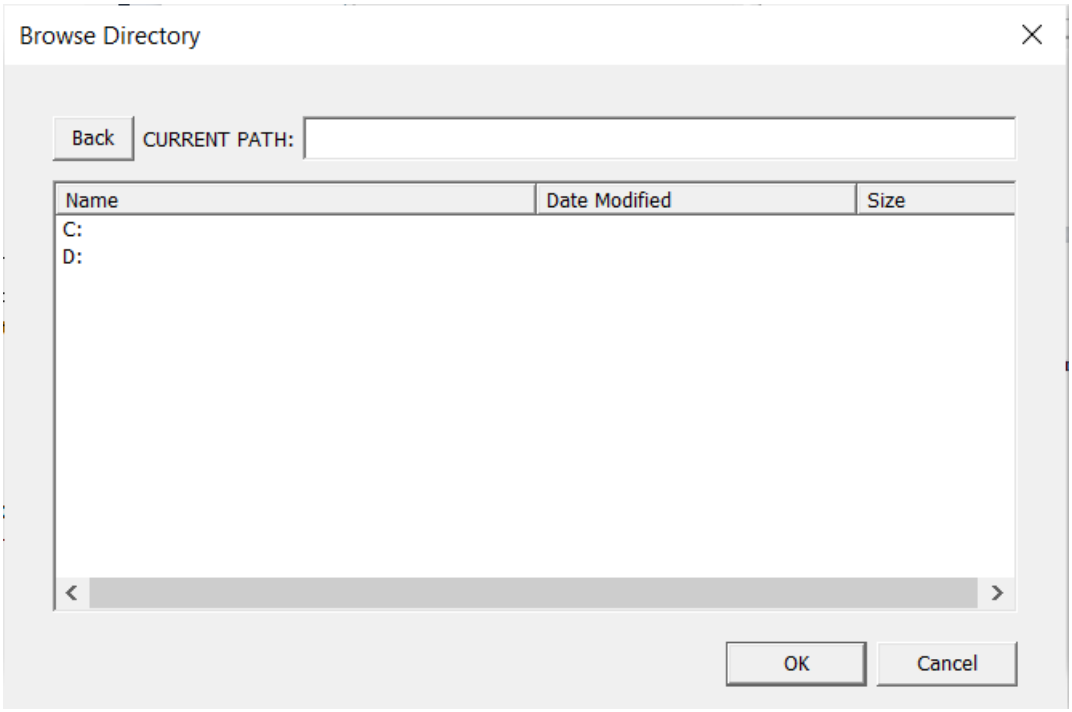
```
void CClientSocket::OnReceive(int nErrorCode);
```

Dây thực chất là phương thức nhận tất cả các thông điệp mà Server gửi đến Client. Tuy nhiên, khi biến `m_isHooked` có giá trị TRUE, tức là hook đã được đặt, tên phím sẽ tự động được thêm vào biến chứa kết quả và được in ra màn hình trong khung text thông qua phương thức `CWnd::SetWindowText()`.

3.7 Duyệt cây thư mục (nút BROWSE DIRECTORY)

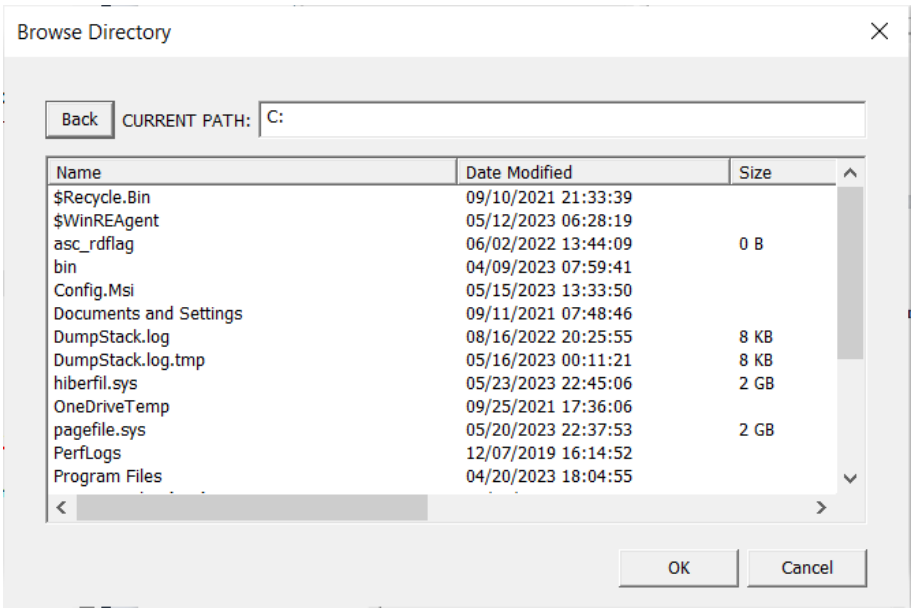
3.7.1 Hướng dẫn sử dụng

Để sử dụng chức năng này, trước hết người dùng phía Client cần nhấn nút BROWSE DIRECTORY. Hộp thoại Browse Directory sẽ hiện ra, với bảng thông tin các ổ đĩa, thư mục, tập tin gồm các thuộc tính: tên, thời gian chỉnh sửa gần nhất và kích thước (Hình 3.15).



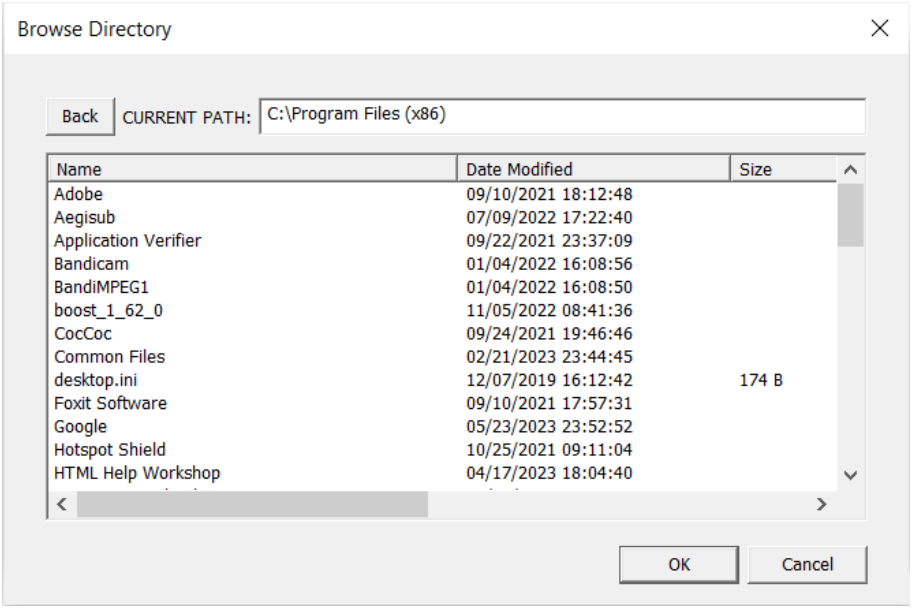
Hình 3.15: Hộp thoại Browse Directory với thông tin các ổ đĩa trong máy Server

Khi hộp thoại Browse Directory hiện ra, đầu tiên trong bản thông tin sẽ hiển thị tên các ổ đĩa trong máy Server. Khi người dùng double click vào tên một ổ đĩa, các thư mục trong ổ đĩa đó lần lượt hiện ra theo thứ tự bảng chữ cái (Hình 3.16).



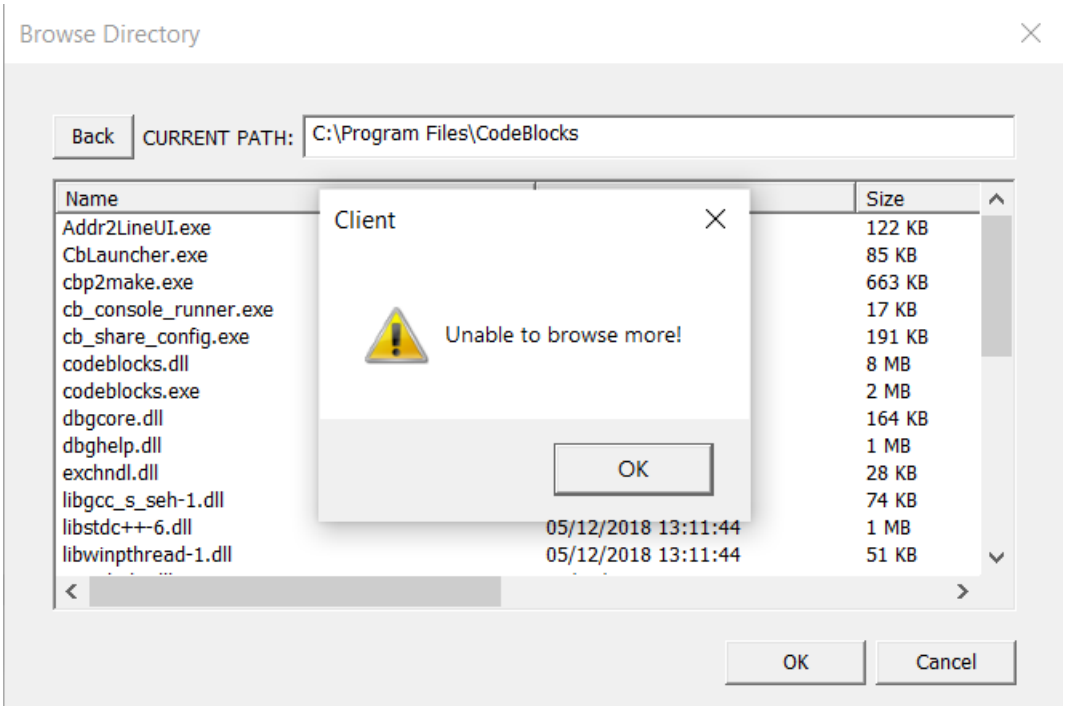
Hình 3.16: Thông tin các thư mục nằm trong ổ đĩa C:

Tương tự, khi người dùng double click tên một thư mục, các thư mục con của thư mục đó cũng hiện ra theo thứ tự bảng chữ cái (Hình 3.17).



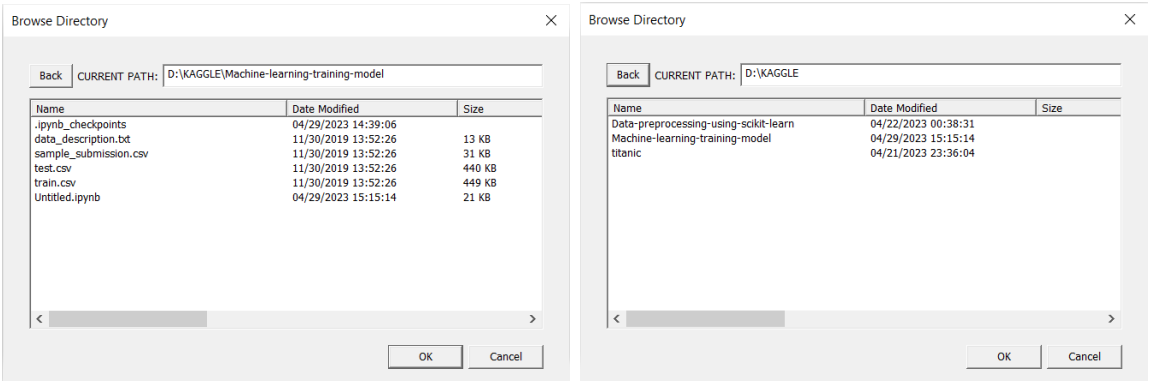
Hình 3.17: Thông tin các thư mục nằm trong thư mục C:/Program Files (x86)

Tuy nhiên, khi người dùng double click tên một tập tin, vì ta không thể duyệt thêm được nữa nên phần mềm sẽ hiển thị hộp thoại thông báo **Unable to browse more!** (Hình 3.18).



Hình 3.18: Vì các file không thể duyệt thêm nên khi tên các file được click, ứng dụng báo lỗi.

Ngoài ra, khi người dùng click nút BACK, có thể quay trở lại thư mục mẹ của thư mục hiện thời (Hình 3.19).



Hình 3.19: Sau khi nhấn nút **BACK**, danh sách thư mục trở về thư mục mẹ của nó

Lưu ý, mỗi khi thực hiện các thao tác vừa được mô tả như trên, khung text CURRENT PATH cũng sẽ thay đổi nội dung là đường dẫn hiện đang được duyệt.

3.7.2 Một số hàm đã được cài đặt

1. **Phương thức gửi yêu cầu thực hiện chức năng duyệt cây thư mục (Client)**

```
1 void OnButtonBrowseDirClicked();
```

Phương thức này được gọi khi người dùng phía Client nhấn nút BROWSE DIRECTORY. Trước hết, ta kiểm tra xem kết nối còn nguyên hay bị mất do ảnh hưởng từ việc thực hiện các chức năng trước đó, nếu mất kết nối thì kết nối lại. Sau

đó, `m_ClientSocket` gửi thông điệp `REQ_BDIR` (request browse directory) bằng phương thức `Send()` yêu cầu Server lấy thông tin các ổ đĩa và trả về cho Client. Client sau đó nhờ biến dialog `m_dlgBDIR` gọi phương thức `ReceiveBrowseDisk()` để nhận dữ liệu và phương thức `DoModal()` để hiển thị hộp thoại chứa các thông tin này.

2. **Hàm duyệt thông tin các ổ đĩa (Server)**

```
void BrowseDisk(std::vector<CStringW>& diskList)
```

Hàm này sử dụng biến `szDrive` để lưu thông tin các ổ đĩa ở dạng string thông qua hàm `GetLogicalDriveStrings()` của thư viện `Windows.h`. Thông tin các ổ đĩa được lưu trong vector `diskList` để sau đó được truy cập và gửi về Client.

3. **Phương thức gửi thông tin các ổ đĩa từ Server về Client**

```
BOOL CReceivingSocket::OnReceiveBrowseDisk(int
    nErrorCode);
```

Phương thức này ban đầu lấy ra danh sách tên ổ đĩa nhờ gọi hàm `BrowseDisk()` đã mô tả bên trên, lưu vào một chuỗi kí tự `msg`, sau đó sử dụng phương thức `Send()` của **CSocket** để gửi chuỗi này sang phía Client.

4. **Phương thức nhận thông tin các ổ đĩa từ Server (Client)**

```
BOOL CBrowseDirDlg::ReceiveBrowseDisk(std::vector<
    CStringA>& msgArr);
```

Phương thức này được gọi ngay sau khi thông điệp `REQ_BDIR` được gửi đi. Nó sử dụng phương thức `Receive()` của **CSocket** để nhận dữ liệu, sau đó cũng lưu thông tin các ổ đĩa vào một dãy các chuỗi kí tự `msgArr`.

5. **Phương thức hiển thị hộp thoại kết quả với tên các ổ đĩa (Client)**

```
BOOL CBrowseDirDlg::OnInitDialog();
```

Phương thức `DoModal()` được gọi ngay sau khi ta gọi phương thức `ReceiveBrowseDisk()`, yêu cầu khởi tạo và hiển thị hộp thoại. Khi đó, phương thức `OnInitDialog()` sẽ được ngầm gọi để chuẩn bị các thông tin trước khi hiển thị hộp thoại. Nó chia dữ liệu lưu trong `msgArr` ra thành từng chuỗi ứng với tên từng ổ đĩa, sau đó tạo bảng với các cột **Name**, **Date Modified**, **Size** để chuẩn bị cho việc hiển thị thông tin các thư mục.

6. **Phương thức gửi yêu cầu duyệt các thư mục (Client)**

```
void CBrowseDirDlg::OnListCtrlClick(NMHDR* pNMHDR,
    LRESULT* pResult);
```

Phương thức này thực chất là các hành động ứng với sự kiện tên một thư mục được double click vào. Sau khi double click, tên các thư mục được trích xuất và gửi sang Server bằng phương thức `Send` kèm với header `REQ_BDIR`. Server nhận được và trả về dữ liệu các thư mục con, tập tin trong thư mục đó, ta bắt lấy bằng hàm `ReceiveBrowseDir()`. Chuỗi dài dữ liệu được về sẽ được tách thành từng thông

tin nhỏ là tên, thời gian chỉnh sửa và kích thước của từng thư mục con/tập tin, và được trình bày lên bảng (quản lý bởi biến). Ngoài ra, nội dung khung text **CURRENT PATH** cũng tự thay đổi thành thư mục đang được duyệt.

7. **Hàm duyệt thư mục (Server)**

```
BOOL BrowseDir(CStringW lpPath, std::vector<CStringW>&
    dirList);
```

Hàm này sử dụng vector `dirList` để lưu thông tin các ổ đĩa với kiểu dữ liệu **CStringW** thông qua các hàm `FindFirstFile()` và `FindNextFile()` cung cấp bởi thư viện `kernel32.dll`. Các hàm này sử dụng tha số `lpPath` được cung cấp. Vì các thư mục có thể tồn tại các ký tự Unicode nên ta sử dụng kiểu dữ liệu **CStringW** thay vì **CString** thông thường để lưu dữ liệu cho chính xác.

8. **Phương thức gửi thông tin các thư mục, tập tin từ Server về Client)**

```
void CReceivingSocket::OnReceiveBrowseDir(int nErrorCode
    , CStringW path);
```

Phương thức này đầu tiên gọi hàm `BrowseDir()` với tham số `path` để có thông tin các thư mục, tập tin và gộp lại thành 1 chuỗi để gửi một lần (dùng phương tức `Send()`). Lưu ý trước khi gửi, các chuỗi phải được chuyển từ dạng wide characters (**CStringW**) về multibyte characters (**CString**, UTTF-8 encoding) để các kí tự giữ đúng ý nghĩa khi được truyền sang phía Client.

9. **Phương thức nhận thông tin các thư mục, tập tin từ Server (Client)**

```
BOOL CBrowseDirDlg::ReceiveBrowseDir(std::vector<
    CStringA>& msgArr);
```

Phương thức này được gọi bên trong `OnListCtrlClick()` ngay sau khi thông điệp `REQ_BDIR` được gửi đi. Nó sử dụng phương thức `Receive()` của **CSocket** để nhận dữ liệu, sau đó cũng thông tin các thư mục, tập tin vào một dãy các chuỗi kí tự `msgArr`.