**Đại Học Quốc Gia Thành Phố Hồ Chí Minh**

**Trường Đại Học Khoa Học Tự Nhiên**



Khoa: Công Nghệ Thông Tin

Môn: Mạng Máy Tính

HK I - Năm Học 2024-2025

**BÁO CÁO ĐỒ ÁN THỰC HÀNH**

CHƯƠNG TRÌNH ĐIỀU KHIỂN MÁY TÍNH TỪ XA

**Nhóm 5**

|  |  |
| --- | --- |
| Lâm Trọng Nghĩa | 22810209 |
| Nguyễn Hải Âu | 24880003 |
| Nguyễn Hồ Quỳnh Ngân | 24880042 |
| Đặng Thị Phương Nguyên | 22880106 |
| Nguyễn Diệu Thiện | 22810215 |

# 

# Nội Dung

[**Nội Dung 2**](#_y9q2zs23hesz)

[**Tổng Quan 3**](#_hu550lqb0kst)

[1. Giới Thiệu Nhóm 3](#_6h5jkq8rt1x7)

[2. Giới Thiệu Chương Trình 3](#_l9vlfqlf0847)

[3. Video Demo Chương Trình 4](#_ycdkozumt9af)

[4. Yêu Cầu Của Chương Trình 4](#_xf68rayojhip)

[5. Các Thư Viện Sử Dụng 5](#_3lbdg87rjfh9)

[**Chức Năng Chương Trình 6**](#_v1gcfrn6pcih)

[1. Chức Năng Kết Nối và Giao Tiếp 6](#_box5ewayb8ah)

[2. Chức Năng Quản Lý Ứng Dụng (Application) 18](#_40hut0nnq0qo)

[3. Chức Năng Quản Lý Process 33](#_welh4nv9dzs4)

[4. Chức Năng Keystroke 48](#_oqk6mfyzfeca)

[4. Chức Năng Screenshot 52](#_bop2ypal0k)

[5. Chức Năng Quản Lý File 55](#_cn5njppdqn1i)

[5. Chức Năng Shutdown và Reset 70](#_u255pjs8rijw)

# Tổng Quan

## 1. Giới Thiệu Nhóm

Nhóm bao gồm 5 thành viên như sau:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nhóm** | **MSSV** | **Họ** | **Tên** | **Địa chỉ thư điện tử** |
| 5 | 22810209 | Lâm Trọng | Nghĩa | 22810209@student.hcmus.edu.vn |
| 24880003 | Nguyễn Hải | Âu | 24880003@student.hcmus.edu.vn |
| 24880042 | Nguyễn Hồ Quỳnh | Ngân | 24880042@student.hcmus.edu.vn |
| 22880106 | Đặng Thị Phương | Nguyên | 22880106@student.hcmus.edu.vn |
| 22810215 | Nguyễn Diệu | Thiện | 22810215@student.hcmus.edu.vn |

Vai trò và đóng góp của các thành viên trong đồ án:

|  |  |  |  |
| --- | --- | --- | --- |
| **Thành Viên** | **MSSV** | **Vai Trò** | **Đóng Góp (%)** |
| Lâm Trọng Nghĩa | 22810209 | - Kiểm thử  - Báo cáo (Files) | 20 |
| Nguyễn Hải Âu | 24880003 | - Kiểm thử  - Báo cáo (Shutdown/Reset) | 20 |
| Nguyễn Hồ Quỳnh Ngân | 24880042 | - Kiểm thử  - Báo cáo (Keystroke, Screenshot)  - Làm video | 20 |
| Đặng Thị Phương Nguyên | 22880106 | - Lập trình  - Báo cáo (Connection, Tổng Quan, trình bày) | 20 |
| Nguyễn Diệu Thiện | 22810215 | - Lập trình  - Báo cáo (Apps, Process) | 20 |

## 2. Giới Thiệu Chương Trình

Chương trình điều khiển từ xa (remote control) được thiết kế nhằm hỗ trợ việc quản lý và điều khiển máy tính từ xa một cách hiệu quả. Chương trình được phát triển trên ngôn ngữ Python. Hệ thống bao gồm hai thành phần chính:

1. Trình **SERVER**: chạy trên máy bị điều khiển
2. Trình **CLIENT**: dùng để điều khiển máy đang chạy trình SERVER

Nhóm đã phát triển được các tính năng bao gồm:

* Quản lý các ứng dụng (application) đang chạy trên máy SERVER (liệt kê, khởi động, hoặc dừng).
* Quản lý các processes trên máy SERVER (liệt kê, khởi động, hoặc dừng).
* Thực hiện tắt nguồn (shutdown) hoặc khởi động lại (reset) máy SERVER.
* Xem màn hình hiện thời của máy SERVER (chụp màn hình và lưu lại).
* Thực hiện khóa bàn phím và bắt phím nhấn (keylogger) trên máy SERVER.
* Xóa hoặc sao chép file từ máy SERVER.

Với các chức năng đa dạng trên, chương trình này là công cụ hữu ích cho việc quản lý từ xa, đặc biệt trong các môi trường cần giám sát và điều hành nhiều máy tính cùng lúc.

## 3. Video Demo Chương Trình

Link của video demo của nhóm có thể coi ở đây:

https://youtu.be/ERiBeSsTZ4Q

## 4. Yêu Cầu Của Chương Trình

Để chạy được chương trình, các yêu cầu cần thiết là:

- Cài đặt Python3, pip3

- Cài đặt các thư viện có trong file Requirements.txt

- Máy bị điều khiển (server) chỉ hỗ trợ hệ điều hành Windows. Máy điều khiển (client) và máy bị điều khiển (server) trong cùng một mạng LAN.

Hướng dẫn cách khởi động chương trình:

* Cài đặt các requirements ở terminal (chỉ làm ở *lần chạy đầu tiên*)

|  |
| --- |
| pip3 install -r requirements.txt |

* Đặt folder Server trên máy bị điều khiển. Chạy file `main.py`

|  |
| --- |
| cd path/to/folder/Server python3 main.py |

* Đặt folder Client trên máy điều khiển. Chạy file `main.py`

|  |
| --- |
| cd path/to/folder/Client python3 main.py |

* Nhập địa chỉ IP của máy bị điều khiển để kết nối. Port mặc định Server mở là 8080 (có thể cấu hình).

## 5. Các Thư Viện Sử Dụng

| **Thư viện** | **Mục đích** | **Chức năng cụ thể** |
| --- | --- | --- |
| socket | Kết nối mạng | - Tạo và quản lý kết nối TCP  - Truyền nhận dữ liệu giữa client-server  - Xử lý timeout và đóng kết nối |
| json | Xử lý dữ liệu | - Chuyển đổi dữ liệu thành JSON  - Format dữ liệu gửi/nhận |
| threading | Xử lý đa luồng | - Tạo thread riêng cho server  - Xử lý nhiều kết nối đồng thời |
| PyQt5 | Giao diện người dùng | - Tạo cửa sổ và dialog  - Hiển thị thông báo và cảnh báo  - Xử lý sự kiện người dùng  - Tạo bảng và hiển thị dữ liệu |
| os | Thao tác hệ thống | - Xử lý file và thư mục  - Thực thi lệnh hệ thống  - Quản lý path |
| subprocess | Xử lý process | - Chạy và quản lý process  - Thực thi lệnh shell  - Lấy output từ command |
| PIL (Pillow) | Xử lý ảnh | - Chụp màn hình  - Xử lý và lưu ảnh |
| pynput | Ghi nhận phím | - Theo dõi keyboard input  - Keylogger  - Xử lý keyboard events |
| logging | Ghi log | - Log thông tin hoạt động  - Log lỗi và cảnh báo  - Debug và troubleshoot |
| datetime | Xử lý thời gian | - Tính toán và ghi nhận thời gian  - Format thời gian |
| typing | Type hints | - Khai báo kiểu dữ liệu  - Kiểm tra typing |
| time | Xử lý thời gian | - Thực hiện delay và timeout  - Ghi nhận thời gian |
| sys | Xử lý lệnh hệ thống | - Xử lý command line |
| collections | Cấu trúc dữ liệu | - Sử dụng cho buffer  - Quản lý dữ liệu |
| platform | Thông tin hệ thống | - Kiểm tra hệ điều hành  - Thực hiện lệnh của hệ điều hành |
| signal | Xử lý signal | - Xử lý giao tiếp giữa system  - Tắt máy tính |

# Chức Năng Chương Trình

## 1. Chức Năng Kết Nối và Giao Tiếp

### a. Tổng Quan Về Cơ Chế

**1. Các Class Chính**

| **Class** | **Thiết bị** | **Chức năng chính** |
| --- | --- | --- |
| RemoteControlServer | Server | - Quản lý hoạt động của server  - Xử lý kết nối từ client (chỉ dùng TCP)  - Điều phối các lệnh và phản hồi |
| NetworkManager | Client | - Quản lý kết nối tới server (chỉ dùng TCP)  - Xử lý gửi/nhận dữ liệu  - Quản lý các socket connection |

**2. Các Phương Thức của Server**

| **Phương thức** | **Loại** | **Chức năng** |
| --- | --- | --- |
| \_\_init\_\_(host, port) | Khởi tạo | - Khởi tạo server với host và port  - Thiết lập môi trường |
| main() | Kết nối | - Tạo và quản lý main/basic socket  - Lắng nghe kết nối từ client |
| handle\_connection(conn, addr, is\_main) | Xử lý kết nối | - Xử lý từng kết nối client  - Phân luồng xử lý main/basic socket |
| toggle\_server(): | Xử lý kết nối | - Kiểm soát vòng đời của server  - Quản lý tài nguyên hệ thống  - Kiểm tra thread kết nối |
| cleanup() | Đóng kết nối | - Xoá tài nguyên  - Đóng các socket |
| shutdown() | Đóng kết nối | - Tắt server  - Giải phóng tài nguyên |

**3. Các Phương Thức của Client**

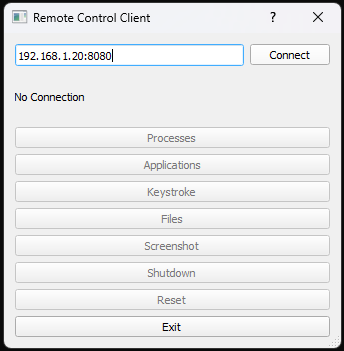
| **Phương thức** | **Loại** | **Chức năng** |
| --- | --- | --- |
| \_\_init\_\_() | Khởi tạo | - Khởi tạo socket và biến môi trường  - Thiết lập timeout (300s) và buffer (4096 bytes, có thể thay đổi tùy nhu cầu của chức năng) |
| connect(host, port) | Kết nối | - Tạo kết nối tới server  - Thiết lập main và basic socket |
| send\_message(msg) | Gửi/Nhận | - Gửi lệnh tới server  - Tự động chọn socket phù hợp |
| \_send\_main\_message(msg) | Gửi/Nhận | - Gửi lệnh qua main socket (dùng cho process, app) |
| \_send\_basic\_message(msg) | Gửi/Nhận | - Gửi lệnh qua basic socket (dùng cho file, keylog) |
| receive\_file(filename) | File | - Nhận và lưu file từ server |
| disconnect() | Đóng kết nối | - Ngắt kết nối với server  - Gửi lệnh quit |
| cleanup() | Đóng kết nối | - Dọn dẹp tài nguyên  - Đóng các socket |

### b. Giao Diện

Ở phía Server:

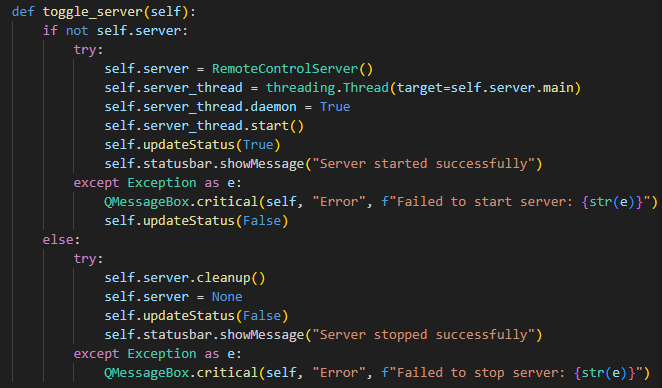
|  |  |
| --- | --- |
| *Trạng thái chưa khởi động server* | *Trạng thái đã khởi động server* |
|  |  |

Ở phía Client:



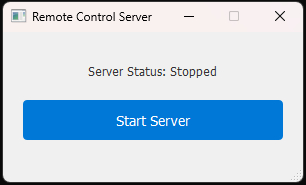
### c. Cách Hoạt Động Chính

#### Chức Năng 1. Khởi động Server

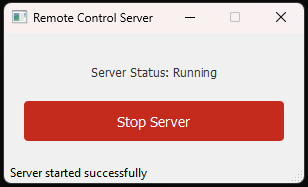
1. Khi người dùng click nút *Start Server*, hàm *toggle\_server* được gọi và khởi động server. Hàm *toggle\_server()* sẽ thực hiện các lệnh sau:

* Kiểm tra nếu server chưa chạy (if not self.server)
* Tạo instance mới của RemoteControlServer
* Tạo thread mới để chạy server.main()
* Set thread là daemon (sẽ tự động kết thúc khi chương trình chính kết thúc)
* Khởi động thread
* Cập nhật UI (chuyển nút Start Server thành Stop Server)

*Trạng thái chưa khởi động server:*

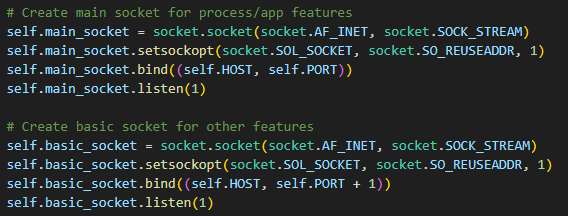


*Trạng thái đã khởi động server:*



1. Server tạo và bind hai socket:

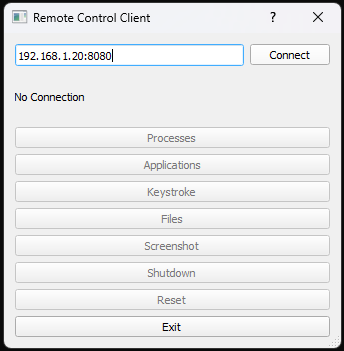
* Main socket trên port 8080
* Basic socket trên port 8081 (8080 + 1)

1. Cả hai socket đều được cấu hình để lắng nghe kết nối, sử dụng protocol TCP (*socket.SOCK\_STREAM*):

#### Chức Năng 2. Kết nối từ Client

1. Khi người dùng nhập địa chỉ và click nút *Connect*, client thực hiện hàm *connect()* trong class *NetworkManager*:

*Trạng thái chưa kết nối:*

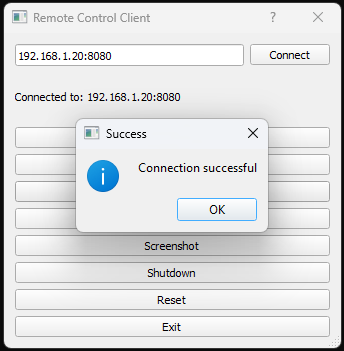


Hàm *connect()* thực hiện các lệnh chính sau:



* Tạo kết nối main socket đến port chính
* Tạo kết nối basic socket đến port + 1
* UI được cập nhật (enable các nút chức năng, hiển thị thông báo trạng thái kết nối)

*Trạng thái kết nối thành công:*

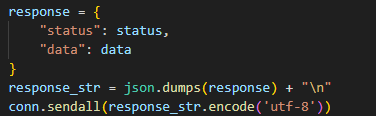


#### Chức Năng 3. Thiết lập giao tiếp và đóng kết nối

1. **Định dạng message**

1.1. Main socket (Process/App commands):

* Format: command//action//parameters
* Ví dụ: process//list, app//kill//123
* Phản hồi dạng JSON với status (trạng thái thành công của kết quả) và data (dữ liệu)



1.2. Basic socket:

* Format đơn giản hơn: command//parameters
* Ví dụ: capture, key//hook
* Phản hồi dạng binary hoặc text đơn giản

1. **Xử lý lỗi và timeout**

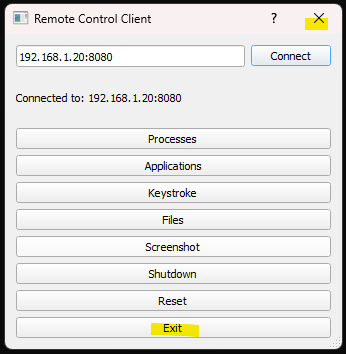
* Timeout mặc định: 300 giây
* Buffer size: 4096 bytes (mặc định, có thể thay đổi theo nhu cầu của từng chức năng)
* Tự động đóng kết nối khi gặp lỗi
* Logging chi tiết các lỗi phát sinh để hỗ trợ debug

#### Chức Năng 4. Ngắt kết nối

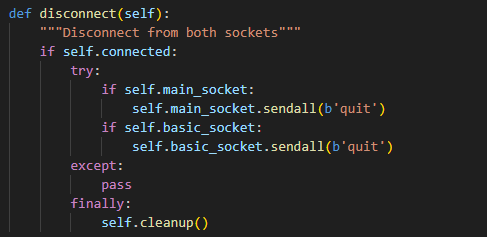
1. **Ngắt kết nối từ Client**

1.1. Khi người dùng đóng ứng dụng hoặc click Exit.

*Yêu cầu ngắt kết nối bằng cách nhấn X ở góc phải hoặc nhấn Exit:*



Hàm *disconnect()* được gọi:

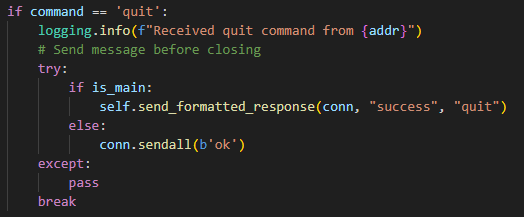


1.2. Các bước thực hiện:

* Gửi lệnh 'quit' đến cả hai socket
* Đóng các socket
* Xóa trạng thái kết nối đã thiết lập

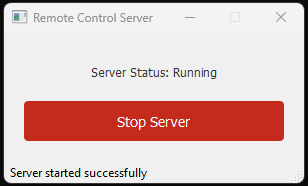
1. **Ngắt kết nối từ Server**

2.1. Nếu server nhận lệnh *quit* từ client thì sẽ phản hồi xác nhận lại cho client:

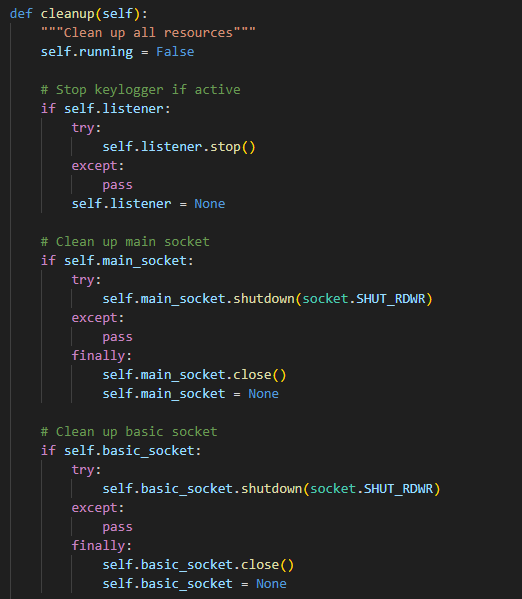


2.2. Server sẽ dừng bằng cách chạy hàm *cleanup* sau khi nhận lệnh *quit* hoặc sau khi người dùng bấm *Stop Server* hoặc tắt ứng dụng server.

*Nhấn Stop Server ở server để ngắt kết nối:*



Hàm *cleanup()* thực hiện các lệnh chính như sau:



- Dừng keylogger nếu đang chạy

- Đóng tất cả các socket

- Giải phóng tài nguyên

## 2. Chức Năng Quản Lý Ứng Dụng (Application)

### a. Tổng Quan Về Cơ Chế

**1. Các Phương Thức của Client**

Nằm trong Class *Dialong\_app(QDialog, Ui\_dialog\_app)* trong file main.py.

| **Phương thức** | **Chức năng** |
| --- | --- |
| def setup\_connections(self) | Các nút thực hiện các mục lớn (Application, Process,…) |
| def app(self): | Kiểm tra kết nối và mở cửa sổ làm việc cho mục Application |
| class Dialog\_app(QDialog, Ui\_dialog\_app) | Hiển thị cửa sổ làm việc cho mục Application |
| def view\_apps(self) | Thực hiện request list các applications trên Server và hiển thị các list applications trả về từ phía Server trong cửa sổ làm việc của mục Application |
| def send\_message(self, msg: str) -> str | Chọn socket phù hợp (với application thì chọn port 8080) |
| def \_send\_main\_message(self, msg: str) -> str | Gửi lệnh đến Server qua socket (có port 8080) |
| def \_receive\_main\_response(self) -> str | Nhận dữ liệu truyền về từ Server đến Client |
| def parse\_server\_response(self, response: str) -> tuple[bool, any] | Kiểm tra dữ liệu truyền về từ Server đến Client |
| def start\_app(self) | Mở cửa sổ làm việc cho mục nhập tên application |
| class Dialog\_start(QDialog, Ui\_dialog\_start) | Hiển thị cửa sổ làm việc cho mục nhập tên application |
| def start(self) | Thực hiện request start application trên Server và refresh các list applications trong cửa sổ làm việc của mục Application |
| def kill\_app(self) | Mở cửa sổ làm việc cho mục nhập id application |
| class Dialog\_kill(QDialog, Ui\_dialog\_kill) | Hiển thị cửa sổ làm việc cho mục nhập id application |
| def kill(self) | Thực hiện request stop application trên Server và refresh các list applications trong cửa sổ làm việc của mục Application |

**2. Các Phương Thức của Server**

Nằm trong def *handle\_app\_commands(self, conn: socket.socket, command: str, args: list)* trong file ps.py.

| **Phương thức** | **Chức năng** |
| --- | --- |
| def handle\_connection(self, conn: socket.socket, addr: str, is\_main: bool) | Xử lý các request từ Client để gọi hàm thực hiện các chức năng tương ứng |
| def handle\_app\_commands(self, conn: socket.socket, command: str, args: list) | Thực hiện một trong các thao tác (list, start, stop) applications ở phía Server mà người dùng truyền lệnh thực hiện |
| def list\_apps(self): | Thực hiện list các applications đang chạy ở Server |
| def send\_formatted\_response(self, conn: socket.socket, status: str, data: any = None) | Thực hiện format lại response và trả response đã được format về cho Client |

### b. Giao Diện

*Hiển thị các ứng dụng đang chạy ở Server:*

A screenshot of a computer

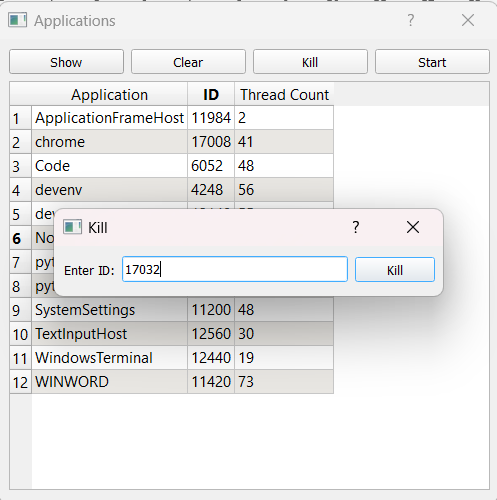
Description automatically generated

*Bật (start) một ứng dụng nào đó ở Server:*

A screenshot of a computer program

Description automatically generated

*Tắt (kill) một ứng dụng theo ID ở Server:*



### c. Cách Hoạt Động Chính

#### Chức Năng 1. Khởi động Applications

1. Chọn nút bấm **Applications** ở màn hình chính, hàm def setup\_connections(self) thực hiện lệnh self.btn\_app.clicked.connect(self.app), tức khi chọn Applications thì btn\_app được chỉ thị nhấn chọn và sau khi được nhấn chọn sẽ thực hiện hàm def app(self).

A screenshot of a remote control

Description automatically generated

1. Hàm def app(self) được thực hiện: Gọi hàm check\_connection() để kiểm tra hiện trạng kết nối giữa máy Client và Server, nếu kết nối không ổn định thì trả về kết quả báo lỗi bên phía Client, nếu kết nối ổn định thì thực hiện Dialog\_app(self.network, self) với tham số truyền vào là đối tượng quản lý kết nối mạng giữa Client và Server đang kết nối với nhau.
2. Class Dialog\_app(QDialog, Ui\_dialog\_app) được thực hiện với hàm chạy chính là def \_\_init\_\_(self, network: NetworkManager, parent=None) bao gồm:

* self.setupUi(self): Khai báo các đối tượng (nút nhấn, layout,..) giao diện người dùng từ lớp Ui\_dialog\_app trong file dialog\_ui.py
* self.network = network: Lưu đối tượng network
* self.setup\_ui(): Thiết lập giao diện hiện sau khi chọn nút Applications
* self.connect\_signals(): Khai báo các nút nếu được nhấp chọn sẽ thực hiện các hàm tương ứng để thực hiện list hoặc start hoặc stop các applications đang chạy ở Server

A screenshot of a computer

Description automatically generated

#### Chức Năng 2. Hiện (list) các ứng dụng đang chạy ở Server

1. Chọn nút **Show**, hàm def connect\_signals(self) thực hiện lệnh self.btn\_show.clicked.connect(self.view\_apps), tức sau khi chọn Show nút btn\_show được chỉ thị nhấp chọn và sau khi được nhấp chọn sẽ thực hiện hàm def view\_apps(self).

A screenshot of a computer

Description automatically generated

1. Hàm def view\_apps(self) ở phía Client được thực hiện:

* Thiết lập con trỏ chuột xoay vòng khi đang chờ response từ Server (QtWidgets.QApplication.setOverrideCursor(Qt.WaitCursor))
* Thiết lập thời gian chờ response là 10 giây để tránh tình trạng client ui bị đứng máy quá lâu (timer = QTimer() và timer.singleShot(10000, lambda: QtWidgets.QApplication.restoreOverrideCursor())
* Request lên server và chờ nhận response từ server về việc thực hiện chức năng List applications đang chạy ở Server bằng cách gọi hàm send\_message('app//list') với biến truyền vào “app//list” để nhận diện đang gọi đến server thực hiện chức năng list các applications.
* Hàm def send\_message(self, msg: str) -> str thực hiện với msg được truyền vào là 'app//list' : Kiểm tra kết nối giữa Client và Server, nếu kết nối không ổn định thì báo lỗi, nếu kết nối ổn định thì sẽ phân chia tùy theo chức năng mà sẽ có port thực hiện khác nhau (như process và app sẽ thực hiện ở port 8080 ở hàm def \_send\_main\_message(self, msg: str) -> str
* Hàm def \_send\_main\_message(self, msg: str) -> str được thực hiện với msg được truyền vào là 'app//list': Kiểm tra kết nối của socket (main\_socket), nếu kết nối không còn thì báo lỗi, nếu còn kết nối thì thiết lập thời gian để request và nhận response từ phía server là 300 (giây) (self.main\_socket.sendall(msg.encode("utf-8"))). Sau khi thiết lập thời gian thực hiện thì gửi yêu cầu List các applications đang chạy trên Server bằng socket (self.main\_socket.sendall(msg.encode("utf-8")) với msg được truyền vào là 'app//list') (Phần Server thực hiện sẽ được đề cập ở phần bên dưới) và nhận các đoạn dữ liệu trả về từ Server qua việc thực hiện hàm def \_receive\_main\_response(self) -> str (với các đoạn dữ liệu trả về có size là 4096). Hàm def \_receive\_main\_response(self) -> str thực hiện vòng lặp while cho đến khi hết các đoạn dữ liệu (có size 4096) được trả về từ Server, mỗi lần thực hiện vòng lặp thì các đoạn dữ liệu trả về được nối lại với nhau, sau đó hàm trả về dữ liệu hoàn chỉnh và nhận kết quả trả về ở đích đến cuối cùng là response của hàm def view\_apps(self) (response = self.network.send\_message('app//list')). Sau khi quay trở lại hàm def view\_apps(self) và nhận response từ Server, thực hiện hàm def parse\_server\_response(self, response: str) -> tuple[bool, any] (với tham số truyền vào là response của Server) để kiểm tra format response trả về và hàm trả về với 2 giá trị (keyword) là status và data.
* Sau khi nhận kết quả trả về từ hàm def parse\_server\_response(self, response: str) ở hàm def view\_apps(self) thì gán vào 2 biến success và data (success, data = self.network.parse\_server\_response(response)). Kiểm tra nếu thời gian đã được thiết lập để chờ response trước đó còn hoạt động thì dừng thời gian và cho con trỏ chuột quay về trạng thái ban đầu (if timer.isActive(): / timer.stop() / QtWidgets.QApplication.restoreOverrideCursor()) ; Kiểm tra trạng thái thành công (dựa vào biến success), kiểu dữ liệu của data (data phải có dạng dict), và ‘app’ phải có trong data. Nếu lỗi thì thông báo đến người dùng.
* Khai báo biến apps và gán giá trị là các biến trong data có từ khóa là ‘app’.
* In các giá trị (name, pid, threads) trong biến app và hiện ở các hàng với các cột tương ứng (Application, ID, Thread Count).

A screenshot of a computer

Description automatically generated

1. Phần thực hiện ở Server:

* Sau khi thực hiện hàm def handle\_connection(self, conn: socket.socket, addr: str, is\_main: bool) sẽ điều hướng request được gửi từ Client tương ứng với các hàm thực hiện, ở lệnh truyền vào từ Client là List các applications (‘app//list’) thì được điều hướng đến thực hiện hàm def handle\_app\_commands(self, conn: socket.socket, command: str, args: list) (với các tham số truyền vào là conn : đường truyền (gửi nhận dữ liệu) giữa Client và Server thông qua socket, command : chức năng yêu cầu thực hiện) và thực hiện trong mục command == list (thực hiện hàm def list\_apps(self)).
* Hàm def list\_apps(self) được thực hiện: Sử dụng thư viện subprocess để thực hiện lấy các danh sách application có format như csv, thiết lập thời gian thực hiện lệnh là 5 giây và gán kết quả vào biến output và error. Nếu có error thì ghi vào lịch sử log (remote\_control.log) và trả về biến dict keyword “app” có danh sách rỗng. Nếu không có error thì push từng dòng giá trị được gán ở biến output vào biến dict có keyword là “app” và trả về biến dict đó ở hàm def handle\_app\_commands(self, conn: socket.socket, command: str, args: list) và gán vào biến app\_data, sau đó thực hiện hàm def send\_formatted\_response(self, conn: socket.socket, status: str, data: any = None) để format lại dữ liệu trả về và thực hiện trả response về Client qua socket.

#### Chức Năng 3. Bật (start) một ứng dụng ở Server

1. Chọn nút **Start**, hàm def connect\_signals(self) ở class Dialog\_app(QDialog, Ui\_dialog\_app) thực hiện lệnh self.btn\_start.clicked.connect(self.start\_app), tức sau khi chọn Start nút btn\_start được chỉ thị nhấp chọn và sau khi được nhấp chọn sẽ thực hiện hàm def start\_app(self) và mở ra hộp cửa sổ làm việc Start (class Dialog\_start(QDialog, Ui\_dialog\_start))A screenshot of a computer program

   Description automatically generated
2. Class Dialog\_start(QDialog, Ui\_dialog\_start) được thực hiện với hàm chạy chính là def \_\_init\_\_(self, status: str, network: NetworkManager, parent=None) bao gồm:

* self.setupUi(self) : Khai báo các đối tượng (nút nhấn, layout,..) giao diện người dùng từ lớp class Ui\_dialog\_start(object) trong file dialog\_ui.py
* self.status = status : loại chức năng được truyền vào (‘app’ hoặc ‘process’)
* self.network = network : Lưu đối tượng network
* self.btn\_start.clicked.connect(self.start) : sau khi btn\_start được nhấp chọn thì thực hiện hàm def start(self)

1. Hàm def start(self) ở class Dialog\_start(QDialog, Ui\_dialog\_start) được thực hiện :

* Kiểm tra tên application nhập ở textbox là hợp lệ.
* Request lên server và chờ nhận response từ server về việc thực hiện chức năng Start application đang chạy ở Server bằng cách gọi hàm send\_message(f'{self.status}//start//{self.lineEdit.text().strip()}') với biến truyền vào “app//start//tên application” để nhận diện đang gọi đến server thực hiện chức năng start application có tên được Client nhập vào trong textbox.
* Hàm def send\_message(self, msg: str) -> str thực hiện với msg được truyền vào là ' app//start//tên application ' : Kiểm tra kết nối giữa Client và Server, nếu kết nối không ổn định thì báo lỗi, nếu kết nối ổn định thì sẽ phân chia tùy theo chức năng mà sẽ có port thực hiện khác nhau (như process và app sẽ thực hiện ở port 8080 ở hàm def \_send\_main\_message(self, msg: str) -> str.
* Hàm def \_send\_main\_message(self, msg: str) -> str được thực hiện với msg được truyền vào là ' app//start//tên application ': Kiểm tra kết nối của socket (main\_socket), nếu kết nối không còn thì báo lỗi, nếu còn kết nối thì thiết lập thời gian để request và nhận response từ phía server là 300 (giây). Sau khi thiết lập thời gian thực hiện thì gửi yêu cầu Start application đang chạy trên Server bằng socket (self.main\_socket.sendall(msg.encode("utf-8")) với msg được truyền vào là ‘app//start//tên application’ (Phần Server thực hiện sẽ được đề cập ở phần bên dưới) và nhận các đoạn dữ liệu trả về từ Server qua việc thực hiện hàm def \_receive\_main\_response(self) -> str (với các đoạn dữ liệu trả về có size là 4096). Hàm def \_receive\_main\_response(self) -> str thực hiện vòng lặp while cho đến khi hết các đoạn dữ liệu (có size 4096) được trả về từ Server, mỗi lần thực hiện vòng lặp thì các đoạn dữ liệu trả về được nối lại với nhau, sau đó hàm trả về dữ liệu hoàn chỉnh và nhận kết quả trả về ở đích đến cuối cùng là response của hàm def start(self) ở class Dialog\_start(QDialog, Ui\_dialog\_start). Sau khi quay trở lại hàm def start(self) và nhận response từ Server, thực hiện hàm def parse\_server\_response(self, response: str) -> tuple[bool, any] (với tham số truyền vào là response của Server) để kiểm tra format response trả về và hàm trả về với 2 giá trị (keyword) là status và data.
* Sau khi nhận kết quả trả về từ hàm def parse\_server\_response(self, response: str) -> tuple[bool, any] ở hàm def start(self) thì gán vào 2 biến success và data (success, data = self.network.parse\_server\_response(response)). Kiểm tra trạng thái thành công (dựa vào biến success). Nếu không success thì thông báo lỗi đến người dùng, nếu success thì hiển thị hộp thoại thông báo start thành công và refresh lại parent dialog (gọi lại hàm view tương ứng).A screenshot of a computer

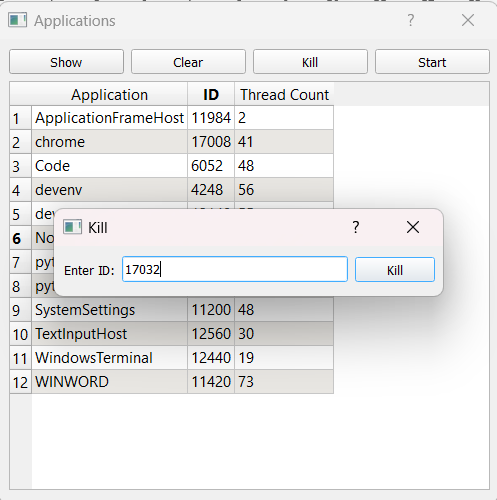
  Description automatically generated

1. Phần thực hiện ở Server

* Sau khi thực hiện hàm def handle\_connection(self, conn: socket.socket, addr: str, is\_main: bool) sẽ điều hướng request được gửi từ Client tương ứng với các hàm thực hiện, ở lệnh truyền vào từ Client là Start application (‘app//start//tên application’) thì được điều hướng đến thực hiện hàm def handle\_app\_commands(self, conn: socket.socket, command: str, args: list) (với các tham số truyền vào là conn : đường truyền (gửi nhận dữ liệu) giữa Client và Server thông qua socket, command : chức năng yêu cầu thực hiện), args : tên application cần start và thực hiện trong mục command == start.
* Sử dụng thư viện subprocess để thực hiện start application. Sau đó thực hiện hàm def send\_formatted\_response(self, conn: socket.socket, status: str, data: any = None) để format lại dữ liệu trả về và thực hiện trả response về Client qua socket.

#### Chức Năng 4. Dừng (stop) ứng dụng theo ID ở Server

1. Chọn nút **Kill**, hàm def connect\_signals(self) ở class Dialog\_app(QDialog, Ui\_dialog\_app) thực hiện lệnh self.btn\_kill.clicked.connect(self.kill\_app), tức sau khi chọn Kill nút btn\_kill được chỉ thị nhấp chọn và sau khi được nhấp chọn sẽ thực hiện hàm def kill\_app(self) và mở ra hộp cửa sổ làm việc Kill (class Dialog\_kill(QDialog, Ui\_dialog\_kill)).



1. Lớp class Dialog\_kill(QDialog, Ui\_dialog\_kill) được thực hiện với hàm chạy tiên khởi là def \_\_init\_\_(self, status: str, network: NetworkManager, parent=None) bao gồm:

* self.setupUi(self) : Khai báo các đối tượng (nút nhấn, layout,..) giao diện người dùng từ lớp class Ui\_dialog\_kill(object) trong file dialog\_ui.py.
* self.status = status : loại chức năng được truyền vào (‘app’ hoặc ‘process’).
* self.network = network : Lưu đối tượng network.
* self.btn\_kill.clicked.connect(self.kill) : sau khi btn\_kill được nhấp chọn thì thực hiện hàm def kill(self).

1. Hàm def kill(self) được thực hiện:

* Kiểm tra id application nhập ở textbox là hợp lệ.
* Request lên server và chờ nhận response từ server về việc thực hiện chức năng Stop application đang chạy ở Server bằng cách gọi hàm send\_message(f'{self.status}//kill//{pid}') với biến truyền vào “app//kill//id application” để nhận diện đang gọi đến server thực hiện chức năng stop application có id được Client nhập vào trong textbox.
* Hàm def send\_message(self, msg: str) -> str thực hiện với msg được truyền vào là ' app//kill//id application ' : Kiểm tra kết nối giữa Client và Server, nếu kết nối không ổn định thì báo lỗi, nếu kết nối ổn định thì sẽ phân chia tùy theo chức năng mà sẽ có port thực hiện khác nhau (như process và app sẽ thực hiện ở port 8080 ở hàm def \_send\_main\_message(self, msg: str) -> str.
* Hàm def \_send\_main\_message(self, msg: str) -> str được thực hiện với msg được truyền vào là ' app//kill//id application ': Kiểm tra kết nối của socket (main\_socket), nếu kết nối không còn thì báo lỗi, nếu còn kết nối thì thiết lập thời gian để request và nhận response từ phía server là 300 giây. Sau khi thiết lập thời gian thực hiện thì gửi yêu cầu Stop application đang chạy trên Server bằng socket (self.main\_socket.sendall(msg.encode("utf-8")) với msg được truyền vào là ‘app//kill//id application’ (Phần Server thực hiện sẽ được đề cập ở phần bên dưới) và nhận các đoạn dữ liệu trả về từ Server qua việc thực hiện hàm def \_receive\_main\_response(self) -> str (với các đoạn dữ liệu trả về có size là 4096). Hàm def \_receive\_main\_response(self) -> str thực hiện vòng lặp while cho đến khi hết các đoạn dữ liệu (có size 4096) được trả về từ Server, mỗi lần thực hiện vòng lặp thì các đoạn dữ liệu trả về được nối lại với nhau, sau đó hàm trả về dữ liệu hoàn chỉnh và nhận kết quả trả về ở đích đến cuối cùng là response của hàm def kill(self) ở class Dialog\_kill(QDialog, Ui\_dialog\_start). Sau khi quay trở lại hàm def kill(self) và nhận response từ Server, thực hiện hàm def parse\_server\_response(self, response: str) -> tuple[bool, any] (với tham số truyền vào là response của Server) để kiểm tra format response trả về và hàm trả về với 2 giá trị (keyword) là status và data.
* Sau khi nhận kết quả trả về từ hàm def parse\_server\_response(self, response: str) -> tuple[bool, any] ở hàm def kill(self) thì gán vào 2 biến success và data (success, data = self.network.parse\_server\_response(response)). Kiểm tra trạng thái thành công (dựa vào biến success). Nếu không success thì thông báo lỗi đến người dùng, nếu success thì hiển thị hộp thoại thông báo start thành công và refresh lại parent dialog (gọi lại hàm view tương ứng).

A screenshot of a computer

Description automatically generated

1. Phần thực hiện ở Server:

* Sau khi thực hiện hàm def handle\_connection(self, conn: socket.socket, addr: str, is\_main: bool) sẽ điều hướng request được gửi từ Client tương ứng với các hàm thực hiện, ở lệnh truyền vào từ Client là Stop application (‘app//kill//id application’) thì được điều hướng đến thực hiện hàm def handle\_app\_commands(self, conn: socket.socket, command: str, args: list) (với các tham số truyền vào là conn : đường truyền (gửi nhận dữ liệu) giữa Client và Server thông qua socket, command : chức năng yêu cầu thực hiện), args : id application cần start và thực hiện trong mục command == kill.
* Sử dụng thư viện subprocess để kiểm tra id application có đang trong danh sách các application đang chạy không và thực hiện kill application. Sau đó thực hiện hàm def send\_formatted\_response(self, conn: socket.socket, status: str, data: any = None) để format lại dữ liệu trả về và thực hiện trả response về Client qua socket.

## 3. Chức Năng Quản Lý Process

### a. Tổng Quan Về Cơ Chế

**1. Các Phương Thức của Client**

Nằm trong Class *Dialog\_process(QDialog, Ui\_dialog\_process)* trong file main.py.

| **Phương thức** | **Chức năng** |
| --- | --- |
| def setup\_connections(self) | Các nút thực hiện các mục lớn (Application, Process,…) |
| def process(self) | Kiểm tra kết nối và mở cửa sổ làm việc cho mục Process |
| class Dialog\_process(QDialog, Ui\_dialog\_process) | Hiển thị cửa sổ làm việc cho mục Process |
| def view\_processes(self) | Thực hiện request list các processes trên Server và hiển thị các list processes trả về từ phía Server trong cửa sổ làm việc của mục Process |
| def send\_message(self, msg: str) -> str | Chọn socket phù hợp (với process thì chọn port 8080) |
| def \_send\_main\_message(self, msg: str) -> str | Gửi lệnh đến Server qua socket (có port 8080) |
| def \_receive\_main\_response(self) -> str | Nhận dữ liệu truyền về từ Server đến Client |
| def parse\_server\_response(self, response: str) -> tuple[bool, any] | Kiểm tra dữ liệu truyền về từ Server đến Client |
| def start\_process(self) | Mở cửa sổ làm việc cho mục nhập tên process |
| class Dialog\_start(QDialog, Ui\_dialog\_start) | Hiển thị cửa sổ làm việc cho mục nhập tên process |
| def start(self) | Thực hiện request start process trên Server và refresh các list processes trong cửa sổ làm việc của mục Process |
| def kill\_process(self) | Mở cửa sổ làm việc cho mục nhập id process |
| class Dialog\_kill(QDialog, Ui\_dialog\_kill) | Hiển thị cửa sổ làm việc cho mục nhập id process |
| def kill(self) | Thực hiện request stop process trên Server và refresh các list processes trong cửa sổ làm việc của mục Process |

**2. Các Phương Thức của Server**

Nằm trong def *handle\_process\_commands(self, conn: socket.socket, command: str, args: list)* trong file ps.py.

| **Phương thức** | **Chức năng** |
| --- | --- |
| def handle\_connection(self, conn: socket.socket, addr: str, is\_main: bool) | Xử lý các request từ Client để gọi hàm thực hiện các chức năng tương ứng |
| def handle\_process\_commands(self, conn: socket.socket, command: str, args: list) | Thực hiện một trong các thao tác (list, start, stop) process ở phía Server mà người dùng truyền lệnh thực hiện |
| def list\_process(self) | Thực hiện list các processes đang chạy ở Server |
| def send\_formatted\_response(self, conn: socket.socket, status: str, data: any = None) | Thực hiện format lại response và trả response đã được format về cho Client |

### b. Giao Diện

*Hiển thị các process đang chạy ở Server:*

A screenshot of a computer

Description automatically generated

*Bật (start) một process nào đó ở Server:*

A screenshot of a computer

Description automatically generated

*Tắt (kill) một process theo ID ở Server:*

A screenshot of a computer

Description automatically generated

### c. Cách Hoạt Động Chính

#### Chức Năng 1. Khởi động Processes

1. Chọn nút bấm **Processes**, hàm def setup\_connections(self) thực hiện lệnh self.btn\_process.clicked.connect(self.process), tức khi chọn Processes thì btn\_process được chỉ thị nhấn chọn và sau khi được nhấn chọn sẽ thực hiện hàm def process(self).

A screenshot of a remote control

Description automatically generated

1. Hàm def process(self) được thực hiện: Gọi hàm check\_connection() để kiểm tra hiện trạng kết nối giữa máy Client và Server, nếu kết nối không ổn định thì trả về kết quả báo lỗi bên phía Client, nếu kết nối ổn định thì thực hiện Dialog\_process(self.network, self) với tham số truyền vào là đối tượng quản lý kết nối mạng giữa Client và Server đang kết nối với nhau.
2. Lớp class Dialog\_process(QDialog, Ui\_dialog\_process) được thực hiện với hàm chạy tiên khởi là def \_\_init\_\_(self, network: NetworkManager, parent=None) bao gồm:

* self.setupUi(self) : Khai báo các đối tượng (nút nhấn, layout,..) giao diện người dùng từ lớp class Ui\_dialog\_process(object) trong file dialog\_ui.py.
* self.network = network : Lưu đối tượng network.
* self.setup\_ui() : Thiết lập giao diện hiện sau khi chọn nút Processes.
* self.connect\_signals() : Khai báo các nút nếu được nhấp chọn sẽ thực hiện các hàm tương ứng để thực hiện list hoặc start hoặc stop các processes đang chạy ở Server.

A screenshot of a computer screen

Description automatically generated

#### Chức Năng 2. Hiện (list) các process đang chạy ở Server

1. Chọn nút **Show**, hàm def connect\_signals(self) thực hiện lệnh self.btn\_show.clicked.connect(self.view\_apps), tức sau khi chọn Show nút btn\_show được chỉ thị nhấp chọn và sau khi được nhấp chọn sẽ thực hiện hàm def view\_processes(self).

A screenshot of a computer screen

Description automatically generated

1. Hàm def view\_processes(self) được thực hiện:

* Thiết lập con trỏ chuột xoay vòng khi đang chờ response từ Server (QtWidgets.QApplication.setOverrideCursor(Qt.WaitCursor)).
* Request lên server và chờ nhận response từ server về việc thực hiện chức năng List processes đang chạy ở Server bằng cách gọi hàm send\_message(process//list') với biến truyền vào “process//list” để nhận diện đang gọi đến server thực hiện chức năng list các processes.
* Hàm def send\_message(self, msg: str) -> str thực hiện với msg được truyền vào là process//list' : Kiểm tra kết nối giữa Client và Server, nếu kết nối không ổn định thì báo lỗi, nếu kết nối ổn định thì sẽ phân chia tùy theo chức năng mà sẽ có port thực hiện khác nhau (như process và app sẽ thực hiện ở port 8080 ở hàm def \_send\_main\_message(self, msg: str) -> str.
* Hàm def \_send\_main\_message(self, msg: str) -> str được thực hiện với msg được truyền vào là process//list': Kiểm tra kết nối của socket (main\_socket), nếu kết nối không còn thì báo lỗi, nếu còn kết nối thì thiết lập thời gian để request và nhận response từ phía server là 300 (giây). Sau khi thiết lập thời gian thực hiện thì gửi yêu cầu List các processes đang chạy trên Server bằng socket (self.main\_socket.sendall(msg.encode("utf-8")) với msg được truyền vào là process//list') (Phần Server thực hiện sẽ được đề cập ở phần bên dưới) và nhận các đoạn dữ liệu trả về từ Server qua việc thực hiện hàm def \_receive\_main\_response(self) -> str (với các đoạn dữ liệu trả về có size là 4096). Hàm def \_receive\_main\_response(self) -> str thực hiện vòng lặp while cho đến khi hết các đoạn dữ liệu (có size 4096) được trả về từ Server, mỗi lần thực hiện vòng lặp thì các đoạn dữ liệu trả về được nối lại với nhau, sau đó hàm trả về dữ liệu hoàn chỉnh và nhận kết quả trả về ở đích đến cuối cùng là response của hàm def view\_processes(self) (response = self.network.send\_message(process//list')). Sau khi quay trở lại hàm def view\_processes(self) và nhận response từ Server, thực hiện hàm def parse\_server\_response(self, response: str) -> tuple[bool, any] (với tham số truyền vào là response của Server) để kiểm tra format response trả về và hàm trả về với 2 giá trị (keyword) là status và data.
* Sau khi nhận kết quả trả về từ hàm def parse\_server\_response(self, response: str) -> tuple[bool, any] ở hàm def view\_processes(self) thì gán vào 2 biến success và data (success, data = self.network.parse\_server\_response(response)). Kiểm tra trạng thái thành công (dựa vào biến success), kiểu dữ liệu của data (data phải có dạng dict), và ‘process’ phải có trong data. Nếu lỗi thì thông báo đến người dùng.
* Khai báo biến processes và gán giá trị là các biến trong data có từ khóa là ‘process’.
* In các giá trị (name, pid, threads) trong biến processes và hiện ở các hàng với các cột tương ứng (Name, Process ID, Thread Count).A screenshot of a computer

  Description automatically generated

1. Phần thực hiện ở Server

* Sau khi thực hiện hàm def handle\_connection(self, conn: socket.socket, addr: str, is\_main: bool) sẽ điều hướng request được gửi từ Client tương ứng với các hàm thực hiện, ở lệnh truyền vào từ Client là List các processes (‘process//list’) thì được điều hướng đến thực hiện hàm def handle\_process\_commands(self, conn: socket.socket, command: str, args: list) (với các tham số truyền vào là conn : đường truyền (gửi nhận dữ liệu) giữa Client và Server thông qua socket, command : chức năng yêu cầu thực hiện) và thực hiện trong mục command == list (thực hiện hàm def list\_process(self)).
* Hàm def list\_process(self) được thực hiện : Sử dụng thư viện os để thực hiện lấy các danh sách process bằng câu lệnh wmic. Nếu có lỗi thì ghi vào lịch sử log (remote\_control.log) và trả về biến dict keyword “process” có danh sách rỗng. Nếu không có lỗi thì push từng dòng giá trị được gán ở biến output vào biến kiểu dict có keyword là “process” và trả về biến kiểu dict đó ở hàm def handle\_process\_commands(self, conn: socket.socket, command: str, args: list) và gán vào biến process\_data, sau đó thực hiện hàm def send\_formatted\_response(self, conn: socket.socket, status: str, data: any = None) để format lại dữ liệu trả về và thực hiện trả response về Client qua socket.

#### Chức Năng 3. Bật (start) process ở Server

1. Chọn nút Start, hàm def connect\_signals(self) ở class Dialog\_process(QDialog, Ui\_dialog\_process) thực hiện lệnh self.btn\_start.clicked.connect(self.start\_process), tức sau khi chọn Start nút btn\_start được chỉ thị nhấp chọn và sau khi được nhấp chọn sẽ thực hiện hàm def start\_process(self) và mở ra hộp cửa sổ làm việc Start (class Dialog\_start(QDialog, Ui\_dialog\_start)).

A screenshot of a computer

Description automatically generated

1. Lớp class Dialog\_start(QDialog, Ui\_dialog\_start) được thực hiện với hàm chạy tiên khởi là def \_\_init\_\_(self, status: str, network: NetworkManager, parent=None) bao gồm:

* self.setupUi(self) : Khai báo các đối tượng (nút nhấn, layout,..) giao diện người dùng từ lớp class Ui\_dialog\_start(object) trong file dialog\_ui.py.
* self.status = status : loại chức năng được truyền vào (‘app’ hoặc ‘process’)
* self.network = network : Lưu đối tượng network.
* self.btn\_start.clicked.connect(self.start) : sau khi btn\_start được nhấp chọn thì thực hiện hàm def start(self).

1. Hàm def start(self) ở class Dialog\_start(QDialog, Ui\_dialog\_start) được thực hiện :
   * Kiểm tra tên process nhập ở textbox là hợp lệ
   * Request lên server và chờ nhận response từ server về việc thực hiện chức năng Start process đang chạy ở Server bằng cách gọi hàm send\_message(f'{self.status}//start//{self.lineEdit.text().strip()}') với biến truyền vào “process//start//tên process” để nhận diện đang gọi đến server thực hiện chức năng start process có tên được Client nhập vào trong textbox.
   * Hàm def send\_message(self, msg: str) -> str thực hiện với msg được truyền vào là ' process//start//tên process ' : Kiểm tra kết nối giữa Client và Server, nếu kết nối không ổn định thì báo lỗi, nếu kết nối ổn định thì sẽ phân chia tùy theo chức năng mà sẽ có port thực hiện khác nhau (như process và app sẽ thực hiện ở port 8080 ở hàm def \_send\_main\_message(self, msg: str) -> str.
   * Hàm def \_send\_main\_message(self, msg: str) -> str được thực hiện với msg được truyền vào là ' process//start//tên process ': Kiểm tra kết nối của socket (main\_socket), nếu kết nối không còn thì báo lỗi, nếu còn kết nối thì thiết lập thời gian để request và nhận response từ phía server là 300 (giây). Sau khi thiết lập thời gian thực hiện thì gửi yêu cầu Start process đang chạy trên Server bằng socket (self.main\_socket.sendall(msg.encode("utf-8")) với msg được truyền vào là ‘process//start//tên process (Phần Server thực hiện sẽ được đề cập ở phần bên dưới) và nhận các đoạn dữ liệu trả về từ Server qua việc thực hiện hàm def \_receive\_main\_response(self) -> str (với các đoạn dữ liệu trả về có size là 4096). Hàm def \_receive\_main\_response(self) -> str thực hiện vòng lặp while cho đến khi hết các đoạn dữ liệu (có size 4096) được trả về từ Server, mỗi lần thực hiện vòng lặp thì các đoạn dữ liệu trả về được nối lại với nhau, sau đó hàm trả về dữ liệu hoàn chỉnh và nhận kết quả trả về ở đích đến cuối cùng là response của hàm def start(self) ở class Dialog\_start(QDialog, Ui\_dialog\_start). Sau khi quay trở lại hàm def start(self) và nhận response từ Server, thực hiện hàm def parse\_server\_response(self, response: str) -> tuple[bool, any] (với tham số truyền vào là response của Server) để kiểm tra format response trả về và hàm trả về với 2 giá trị (keyword) là status và data.
   * Sau khi nhận kết quả trả về từ hàm def parse\_server\_response(self, response: str) -> tuple[bool, any] ở hàm def start(self) thì gán vào 2 biến success và data (success, data = self.network.parse\_server\_response(response)). Kiểm tra trạng thái thành công (dựa vào biến success). Nếu không success thì thông báo lỗi đến người dùng, nếu success thì hiển thị hộp thoại thông báo start thành công và refresh lại parent dialog (gọi lại hàm view tương ứng).A screenshot of a computer

     Description automatically generated
2. Phần thực hiện ở Server (ps.py)

* Sau khi thực hiện hàm def handle\_connection(self, conn: socket.socket, addr: str, is\_main: bool) sẽ điều hướng request được gửi từ Client tương ứng với các hàm thực hiện, ở lệnh truyền vào từ Client là Start process (‘process//start//tên process) thì được điều hướng đến thực hiện hàm def handle\_process\_commands(self, conn: socket.socket, command: str, args: list) (với các tham số truyền vào là conn : đường truyền (gửi nhận dữ liệu) giữa Client và Server thông qua socket, command : chức năng yêu cầu thực hiện), args : tên process cần start và thực hiện trong mục command == start.
* Sử dụng thư viện subprocess để thực hiện start process. Sau đó thực hiện hàm def send\_formatted\_response(self, conn: socket.socket, status: str, data: any = None) để format lại dữ liệu trả về và thực hiện trả response về Client qua socket.

#### Chức Năng 4. Dừng (stop) process theo ID ở Server

1. Chọn nút Kill, hàm def connect\_signals(self) ở class Dialog\_process(QDialog, Ui\_dialog\_process) thực hiện lệnh self.btn\_kill.clicked.connect(self.kill\_process), tức sau khi chọn Kill nút btn\_kill được chỉ thị nhấp chọn và sau khi được nhấp chọn sẽ thực hiện hàm def kill\_process(self) và mở ra hộp cửa sổ làm việc Kill (class Dialog\_kill(QDialog, Ui\_dialog\_kill)).A screenshot of a computer

   Description automatically generated
2. Lớp class Dialog\_kill(QDialog, Ui\_dialog\_kill) được thực hiện với hàm chạy tiên khởi là def \_\_init\_\_(self, status: str, network: NetworkManager, parent=None) bao gồm:

* self.setupUi(self) : Khai báo các đối tượng (nút nhấn, layout,..) giao diện người dùng từ lớp class Ui\_dialog\_kill(object) trong file dialog\_ui.py.
* self.status = status : loại chức năng được truyền vào (‘app’ hoặc ‘process’).
* self.network = network : Lưu đối tượng network.
* self.btn\_kill.clicked.connect(self.kill) : sau khi btn\_kill được nhấp chọn thì thực hiện hàm def kill(self).

1. Hàm def kill(self) được thực hiện :
   * Kiểm tra id process nhập ở textbox là hợp lệ.
   * Request lên server và chờ nhận response từ server về việc thực hiện chức năng Stop process đang chạy ở Server bằng cách gọi hàm send\_message(f'{self.status}//kill//{pid}') với biến truyền vào “process//kill//id process” để nhận diện đang gọi đến server thực hiện chức năng stop process có id được Client nhập vào trong textbox.
   * Hàm def send\_message(self, msg: str) -> str thực hiện với msg được truyền vào là ' process//kill//id process ' : Kiểm tra kết nối giữa Client và Server, nếu kết nối không ổn định thì báo lỗi, nếu kết nối ổn định thì sẽ phân chia tùy theo chức năng mà sẽ có port thực hiện khác nhau (như process và app sẽ thực hiện ở port 8080 ở hàm def \_send\_main\_message(self, msg: str) -> str.
   * Hàm def \_send\_main\_message(self, msg: str) -> str được thực hiện với msg được truyền vào là ' process//kill//id process ': Kiểm tra kết nối của socket (main\_socket), nếu kết nối không còn thì báo lỗi, nếu còn kết nối thì thiết lập thời gian để request và nhận response từ phía server là 300 (giây). Sau khi thiết lập thời gian thực hiện thì gửi yêu cầu Stop process đang chạy trên Server bằng socket (self.main\_socket.sendall(msg.encode("utf-8")) với msg được truyền vào là ‘process//kill//id process (Phần Server thực hiện sẽ được đề cập ở phần bên dưới) và nhận các đoạn dữ liệu trả về từ Server qua việc thực hiện hàm def \_receive\_main\_response(self) -> str (với các đoạn dữ liệu trả về có size là 4096). Hàm def \_receive\_main\_response(self) -> str thực hiện vòng lặp while cho đến khi hết các đoạn dữ liệu (có size 4096) được trả về từ Server, mỗi lần thực hiện vòng lặp thì các đoạn dữ liệu trả về được nối lại với nhau, sau đó hàm trả về dữ liệu hoàn chỉnh và nhận kết quả trả về ở đích đến cuối cùng là response của hàm def kill(self) ở class Dialog\_kill(QDialog, Ui\_dialog\_start). Sau khi quay trở lại hàm def kill(self) và nhận response từ Server, thực hiện hàm def parse\_server\_response(self, response: str) -> tuple[bool, any] (với tham số truyền vào là response của Server) để kiểm tra format response trả về và hàm trả về với 2 giá trị (keyword) là status và data.
   * Sau khi nhận kết quả trả về từ hàm def parse\_server\_response(self, response: str) -> tuple[bool, any] ở hàm def kill(self) thì gán vào 2 biến success và data (success, data = self.network.parse\_server\_response(response)). Kiểm tra trạng thái thành công (dựa vào biến success). Nếu không success thì thông báo lỗi đến người dùng, nếu success thì hiển thị hộp thoại thông báo start thành công và refresh lại parent dialog (gọi lại hàm view tương ứng).A screenshot of a computer

     Description automatically generated
2. Phần thực hiện ở Server (ps.py)

* Sau khi thực hiện hàm def handle\_connection(self, conn: socket.socket, addr: str, is\_main: bool) sẽ điều hướng request được gửi từ Client tương ứng với các hàm thực hiện, ở lệnh truyền vào từ Client là Stop process (‘process//kill//id process) thì được điều hướng đến thực hiện hàm def handle\_process\_commands(self, conn: socket.socket, command: str, args: list) (với các tham số truyền vào là conn : đường truyền (gửi nhận dữ liệu) giữa Client và Server thông qua socket, command : chức năng yêu cầu thực hiện), args : id process cần start và thực hiện trong mục command == kill.
* Sử dụng thư viện subprocess để kiểm tra id process có đang trong danh sách các application đang chạy không và thực hiện kill application. Sau đó thực hiện hàm def send\_formatted\_response(self, conn: socket.socket, status: str, data: any = None) để format lại dữ liệu trả về và thực hiện trả response về Client qua socket.

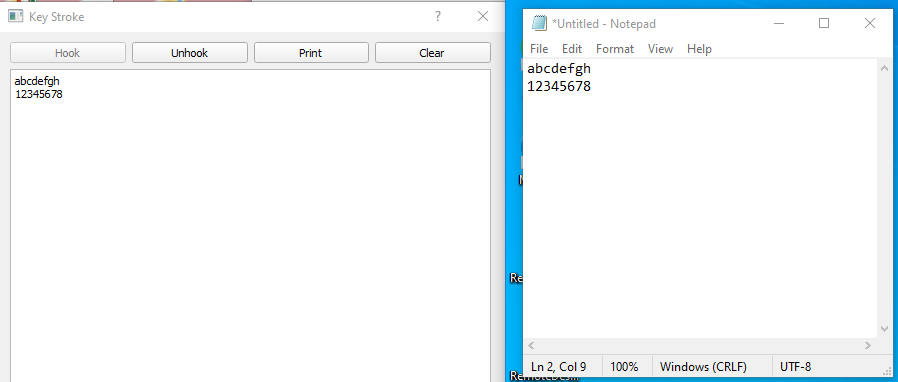
## 4. Chức Năng Keystroke

### a. Tổng Quan Về Cơ Chế

**Module**: dialog\_keystroke()

| **Phương thức** | **Chức năng** |
| --- | --- |
| toggle\_hook() | Bắt đầu ghi lại các phím nhấn bằng cách gửi yêu cầu 'hook' tới server. |
| unhook() | Ngừng ghi lại các phím nhấn bằng cách gửi yêu cầu 'unhook' tới server. |
| get\_keystrokes() | Lấy nhật ký các phím nhấn từ server và hiển thị trong giao diện. |
| clear\_log() | Xóa nhật ký các phím nhấn trong giao diện. |
| setup\_ui() | Cài đặt giao diện ban đầu, bao gồm cài đặt chế độ chỉ đọc và trạng thái nút. |
| connect\_signals() | Kết nối các nút giao diện với các chức năng tương ứng. |
| update\_hook\_button\_state() | Cập nhật trạng thái các nút bắt đầu và ngừng keylogger dựa trên trạng thái hiện tại. |

### b. Giao Diện



### c. Cách Hoạt Động Chính

#### Chức Năng 1. Khởi động Keystroke

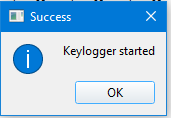
* Khi nhấn nút Keystroke , tức là lớp Dialog\_keystroke được khởi tạo, hàm **\_\_init\_\_** sẽ được gọi.
* Hàm này thực hiện việc thiết lập giao diện người dùng qua **setupUi(self)**.
* Gọi hàm **setup\_ui(self)** để cấu hình các thành phần giao diện như: đặt trạng thái của textBrowser\_2 là chỉ đọc (read-only) và thiết lập **is\_hooked** là False (không khóa phím).
* Gọi hàm **connect\_signals(self)** để kết nối các nút bấm với các hàm xử lý sự kiện:
* Nút **Hook** sẽ gọi hàm **toggle\_hook(self)** để bắt đầu keylogger.
* Nút **Unhook** sẽ gọi hàm **unhook(self)** để dừng keylogger.
* Nút **Print** sẽ gọi hàm **get\_keystrokes(self)** để lấy và in ra các phím đã ghi lại.
* Nút **Clear** sẽ gọi hàm **clear\_log(self)** để xóa log phím.

#### Chức Năng 2. Khóa (hook) phím của Server

* Khi người dùng nhấn nút Hook, hàm toggle\_hook(self) sẽ được gọi.
* Hàm này gửi một yêu cầu qua mạng tới server để bắt đầu keylogger bằng cách gửi lệnh 'key//hook':



* Nút **Hook** sẽ bị vô hiệu hóa (disabled).
* Nút **Unhook** sẽ được kích hoạt (enabled).
* Hiển thị người dùng qua hộp thoại:



* Nếu có lỗi xảy ra, thông báo lỗi sẽ được hiển thị cho người dùng qua hộp thoại QMessageBox.critical.

#### Chức Năng 3. In (print) phím đã log

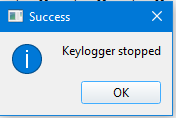
* Khi người dùng nhấn nút **Print**, hàm get\_keystrokes(self) sẽ được gọi.
* Hàm này gửi một yêu cầu qua mạng tới server để lấy dữ liệu các phím đã gõ thông qua lệnh 'key//getkey':
* Nếu server trả về một chuỗi không phải là '404' (tức là có dữ liệu phím), các phím đó sẽ được thêm vào textBrowser\_2.
  + Nội dung cũ trong textBrowser\_2 sẽ được giữ lại và các phím mới sẽ được thêm vào.
  + Sau khi thêm, thanh cuộn trong textBrowser\_2 sẽ tự động cuộn xuống dưới để hiển thị phím mới.
* Nếu không có dữ liệu hoặc có lỗi xảy ra, một thông báo lỗi sẽ được hiển thị qua QMessageBox.critical.

#### Chức Năng 4. Bỏ khóa (unhook) phím của Server

* Khi người dùng nhấn nút Unhook, hàm unhook(self) sẽ được gọi.
* Hàm này gửi yêu cầu qua mạng tới server để dừng keylogger bằng lệnh 'key//unhook':



* Nếu server trả về 'ok', keylogger sẽ bị dừng, và trạng thái is\_hooked sẽ được cập nhật thành False. Giao diện sẽ thay đổi trạng thái các nút:
* Nút **Hook** sẽ được kích hoạt lại.
* Nút **Unhook** sẽ bị vô hiệu hóa.
* Hiển thị người dùng qua hộp thoại:



* Nếu có lỗi xảy ra, thông báo lỗi sẽ được hiển thị cho người dùng qua QMessageBox.critical.

#### Chức Năng 5. Phím Clear

Nút **Clear** được kết nối với phương thức clear\_log(), và phương thức này có chức năng:

* Xóa nội dung trong textBrowser\_2, là nơi hiển thị log các phím được ghi lại từ keylogger.
* Cụ thể, nó sử dụng phương thức clear() của QTextBrowser để làm trống toàn bộ nội dung hiển thị.

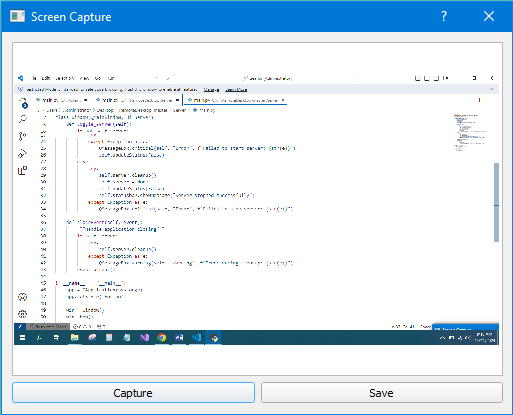
## 4. Chức Năng Screenshot

### a. Tổng Quan Về Cơ Chế

**Module**: dialog\_capture()

| **Phương thức** | **Chức năng** |
| --- | --- |
| capture() | Chụp ảnh màn hình từ server và hiển thị trực tiếp trên giao diện. |
| save() | Lưu ảnh chụp màn hình vào tệp tin do người dùng chọn. |

### b. Giao Diện



### c. Cách Hoạt Động Chính

#### Chức Năng 1. Khởi động Screenshot

1. Khi ứng dụng khởi động, phương thức setup\_connections(self) sẽ được gọi. Đây là nơi thiết lập các kết nối sự kiện giữa các nút bấm trong giao diện người dùng và các hàm xử lý của chương trình.

2. Trong setup\_connections(self), lệnh sau được sử dụng để kết nối nút **Screenshot** (nút pushButton\_2) với hàm capture():



Điều này có nghĩa là khi người dùng nhấn nút **Screenshot**, chương trình sẽ gọi hàm capture () để bắt đầu quá trình chụp ảnh.

#### Chức Năng 2. Chụp màn hình (capture) của Server

Khi người dùng nhấn nút Screenshot, hàm capture(self) sẽ được gọi và thực hiện các bước sau:

1. Kiểm tra kết nối mạng:

* Chương trình sẽ kiểm tra xem socket có được kết nối với server hay không. Nếu không có kết nối, chương trình sẽ đưa ra thông báo lỗi và không tiếp tục thực hiện.
* Nếu kết nối không tồn tại, sẽ có thông báo:



2. Gửi lệnh chụp ảnh:

* Nếu kết nối thành công, chương trình sẽ gửi lệnh capture đến server thông qua socket để yêu cầu server thực hiện chụp ảnh màn hình.
* Câu lệnh gửi sẽ được thực hiện như sau:

3. Nhận dữ liệu ảnh từ server:

* Sau khi gửi lệnh, chương trình bắt đầu nhận dữ liệu ảnh từ server qua socket.
* Chương trình sử dụng một vòng lặp để nhận dữ liệu theo từng phần nhỏ (chunk). Dữ liệu này được lưu vào bộ đệm img\_data.
* Vòng lặp sẽ tiếp tục nhận dữ liệu cho đến khi gặp chuỗi kết thúc <<END>>, thông báo rằng dữ liệu ảnh đã nhận xong.

4. Kiểm tra và hiển thị ảnh:

* Khi dữ liệu ảnh đã nhận đầy đủ, chương trình kiểm tra xem dữ liệu có hợp lệ hay không. Nếu không, chương trình sẽ đưa ra thông báo lỗi.
* Nếu ảnh hợp lệ, chương trình sẽ sử dụng QPixmap để tải và hiển thị ảnh lên QGraphicsView:





5. Xử lý lỗi:

* Nếu có lỗi xảy ra trong quá trình nhận dữ liệu (như socket timeout, kết nối bị mất), chương trình sẽ hiển thị thông báo lỗi cho người dùng:

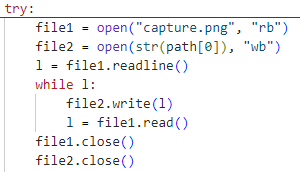
#### Chức Năng 3. Lưu (save) ảnh chụp về Client

Sau khi ảnh được chụp và hiển thị, người dùng có thể lưu ảnh vào máy tính của mình. Khi nhấn nút Save, hàm save(self) sẽ được gọi và thực hiện các bước sau:

1. Mở hộp thoại lưu tệp:

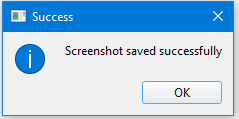
* Khi người dùng nhấn nút Save, một hộp thoại QFileDialog.getSaveFileName() sẽ mở ra, cho phép người dùng chọn vị trí và tên tệp để lưu ảnh.
* Dữ liệu của tệp ảnh sẽ được lưu từ capture.png (ảnh mặc định trong ứng dụng).

1. Sao chép và lưu ảnh:

* Sau khi người dùng chọn vị trí và tên tệp, chương trình sẽ mở tệp capture.png và sao chép nội dung vào tệp mà người dùng đã chọn.
* Đây là mã thực hiện sao chép dữ liệu ảnh:

1. Thông báo kết quả:

* Sau khi ảnh được lưu thành công, chương trình sẽ hiển thị thông báo thành công cho người dùng:



* Nếu có lỗi xảy ra trong quá trình lưu ảnh, chương trình sẽ hiển thị thông báo lỗi:

## 5. Chức Năng Quản Lý File

### a. Tổng Quan Về Cơ Chế

**Feature**: Files

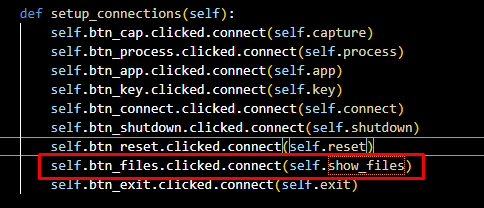
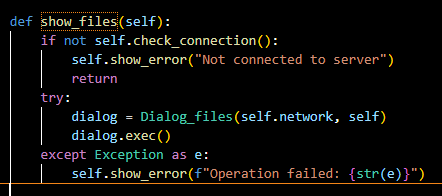
| **Phương thức** | | **Chức năng** |
| --- | --- | --- |
| **Client** | **Server** |
| show\_files() |  | Check connect tới server và gọi dialog\_files() |
| dialog\_files() | handle\_file\_commands() | Khởi tạo giao diện và các sự kiện cần thiết. |
| browse\_typed\_path() | list\_files() | Duyệt đường dẫn thư mục từ textbox. |
| refresh\_files(), update\_file\_list() | list\_files() | Duyệt lại đường dẫn và cập nhật DS file |
| on\_item\_double\_clicked() | list\_files() | Duyệt và show tất cả file, folder nếu là thư mục |
| go\_up\_directory() | list\_files() | Trờ về thư mục trước đó |
| download\_file() | send\_file() | Tải file từ server về máy client |
| delete\_file() | delete\_file() | Xóa file trên máy server |

### b. Giao Diện

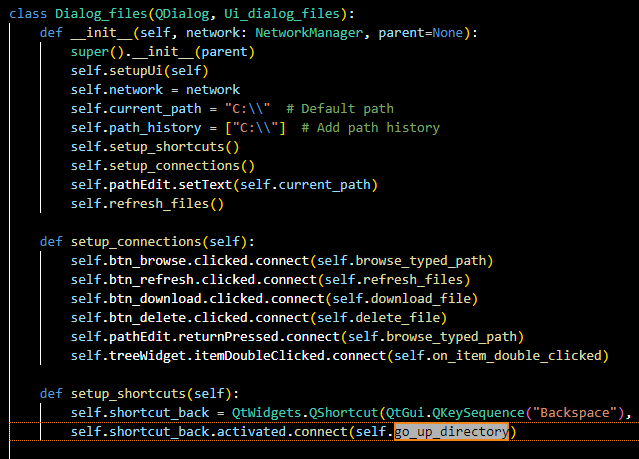


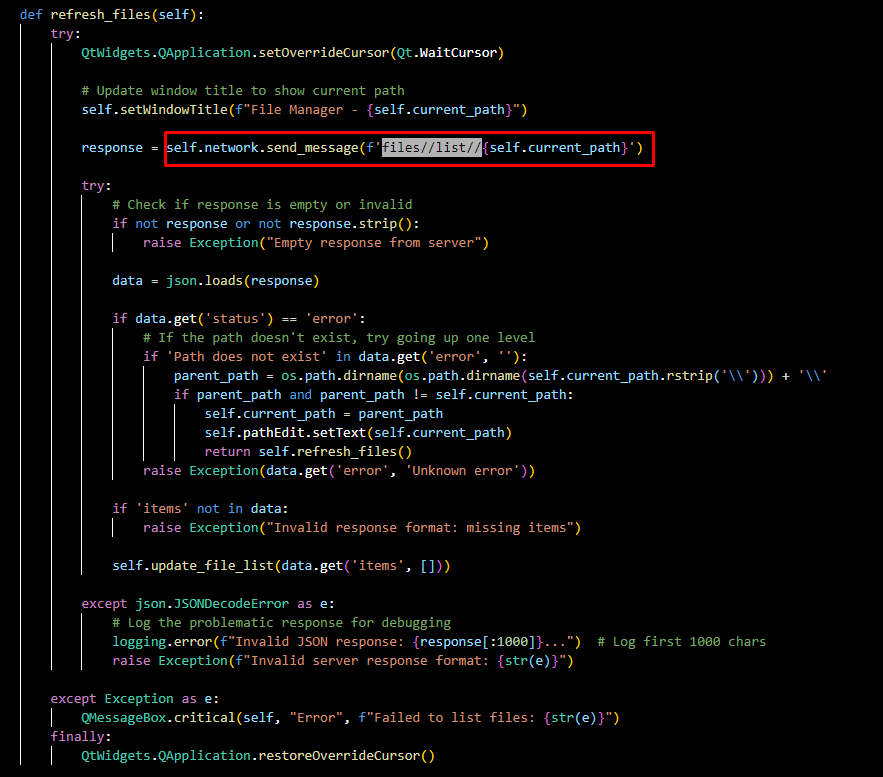
### c. Cách Hoạt Động Chính

#### Chức Năng 1. Khởi động File Manager và Duyệt Thư Mục

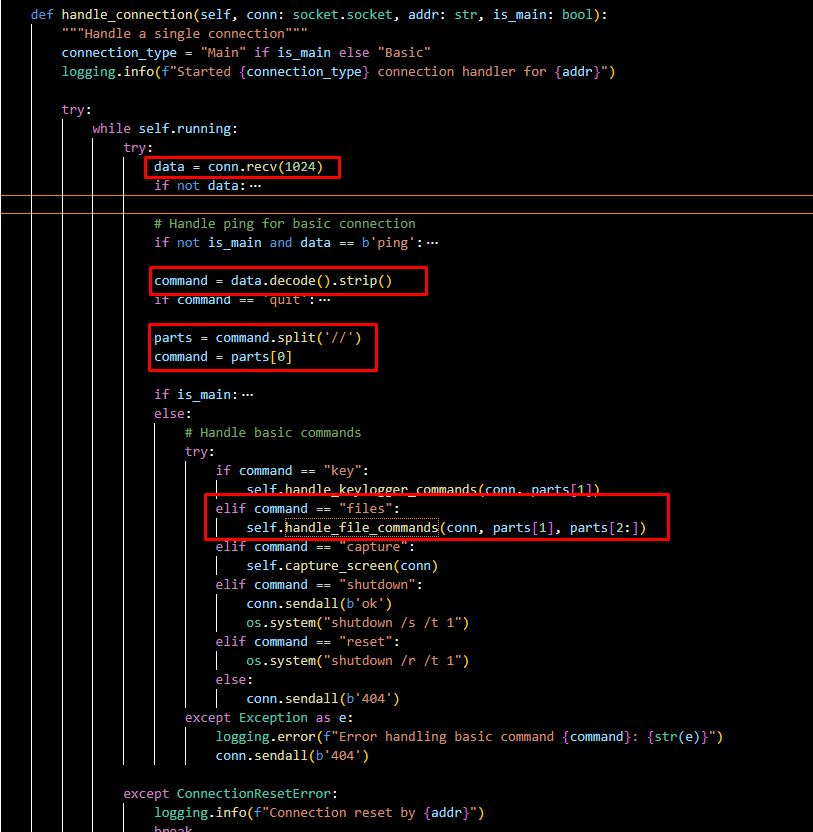
1. Nhấn vào nút Files từ giao diện client sẽ call function show\_files
2. Trong function này sẽ check connection tới server, nếu có thì thì gọi function Dialog\_files để khởi tạo và thực thi (\*) các nghiệp vụ và show form File Manager với ổ đĩa đặt mặc định là C. Ngược lại sẽ hiện “not connect to server”.

* Các nghiệp vụ ở phần (\*) là: Khởi tạo sự kiện cho các button, key Backspace và đọc tất cả folder, file (cấp 1) bên trong ổ C trên Server hiện lên form File Manager thông qua các function \_init\_, setup\_connections, setup\_shortcuts, refresh\_files.

1. Xử lý phía Client:

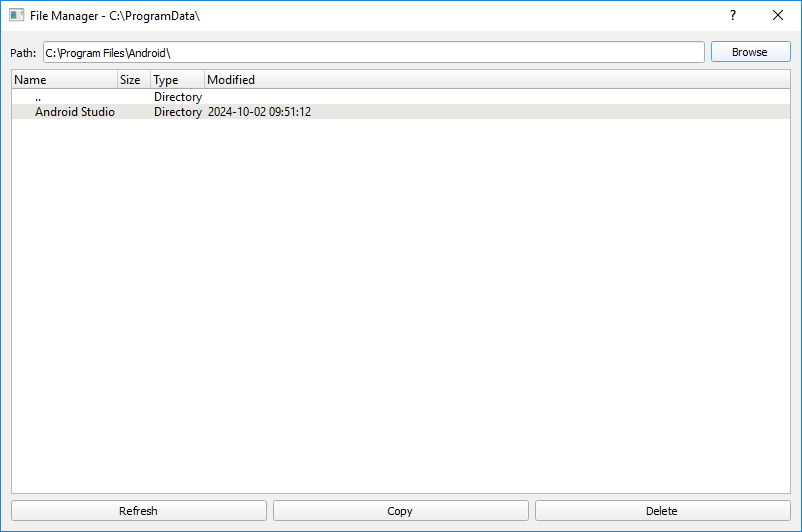
* Client sẽ gửi message “files//list//{path}” qua Server xử lý và nhận lại kết quả.

1. Xử lý phía Server:

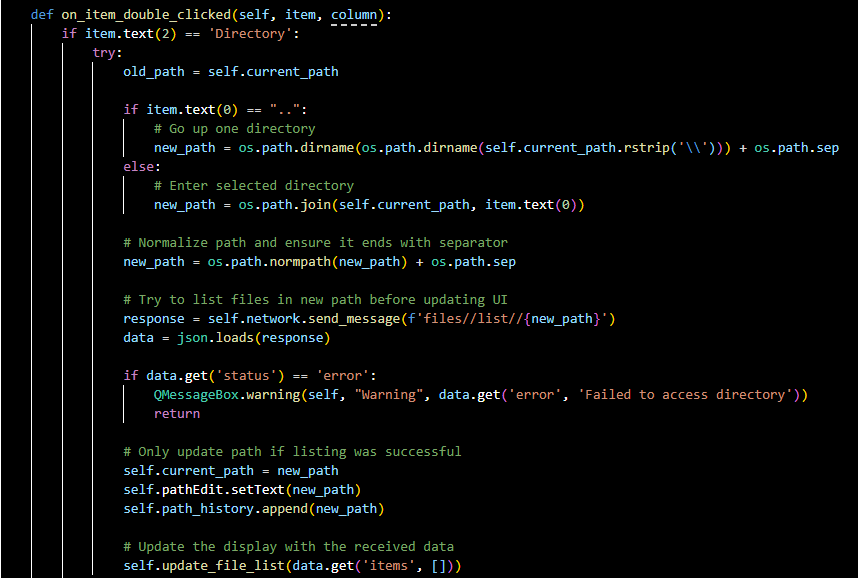
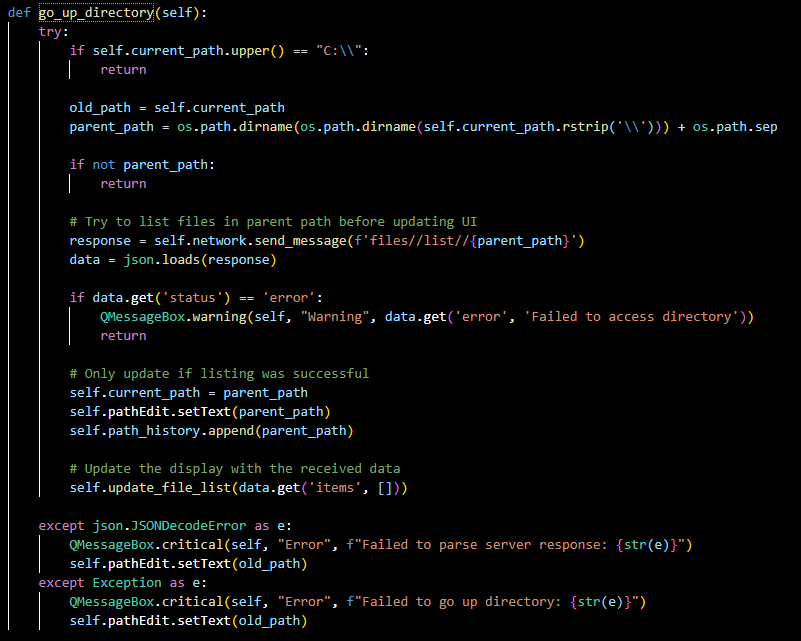
* Server sẽ xử lý cắt chuỗi message để biết command cần thực hiện là files và action là list folder như hình qua 2 function (handle\_connection, handle\_file\_commands) và list\_files.
* Bên trong **list\_files** sẽ dùng thư viện **os** đọc tất cả files và folders bên trong gồm các thông tin **tên, size, type** và **DateModified.**  Sau đó trả ra 1 danh sách **items** và trạng thái **success** hoặc **error** và thông báo lỗi



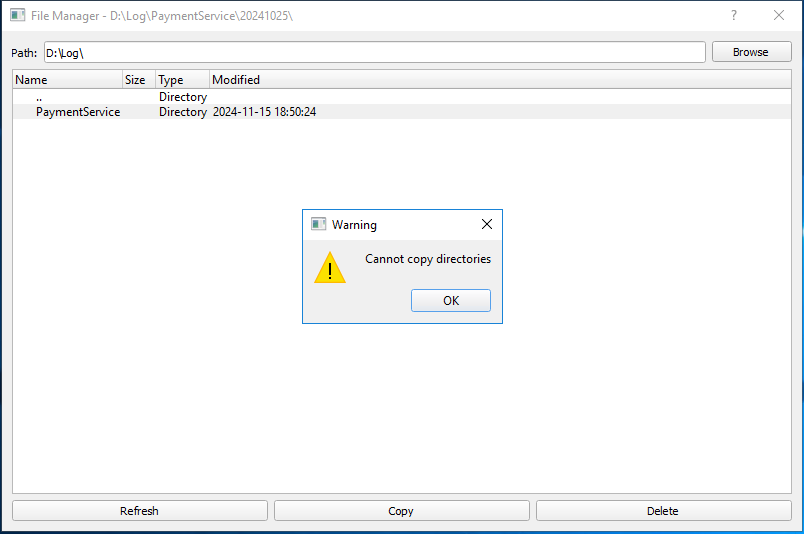
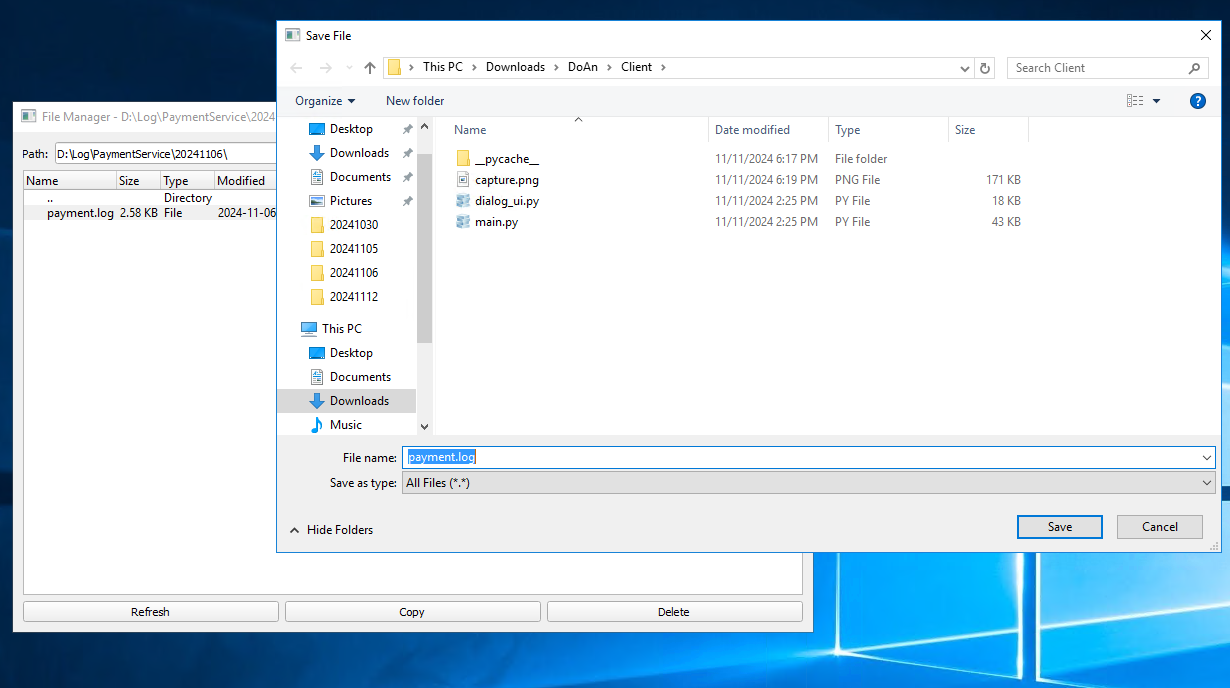
1. Trên giao diện File Manager các tính năng như:

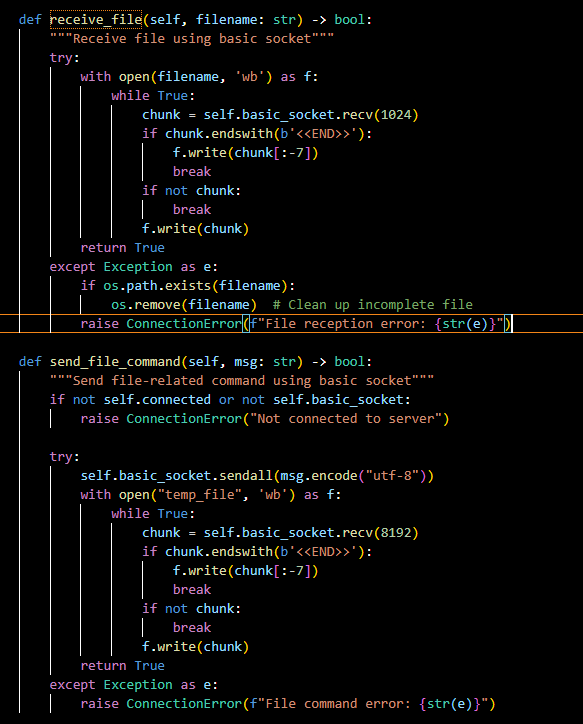
* Duyệt đường dẫn từ ô textbox bằng function browse\_typed\_path
  + Tải lại danh sách file và folder trong ổ C sau đó trình bày lên giao diện bằng function refresh\_files và update\_file\_list:

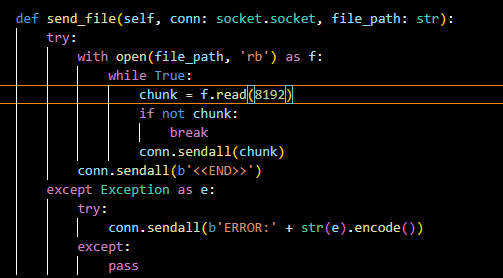


* Khi double click vào 1 item trong danh sách nếu là folder thì sẽ duyệt các thư mục con và file bên trong folder vừa click và show ra form bằng function on\_item\_double\_clicked:
* Khi nhấn phím Backspace trên bàn phím thì sẽ trở ra folder trước và show danh sách file và folder của folder trước bằng function go\_up\_director: bên trong sẽ xử lý lưu lại đường dẫn hiện tại, sao đó thực hiện lấy đường dẫn cha bằng cách gọi lòng func **os.path.dirname**, nếu đường dẫn cha hợp lệ thì call qua server để lấy danh sách files và folders con. Nếu không hợp lệ thì giữ nguyên hiện trạng.

#### Chức Năng 2. Sao chép (copy) file từ Server về Client

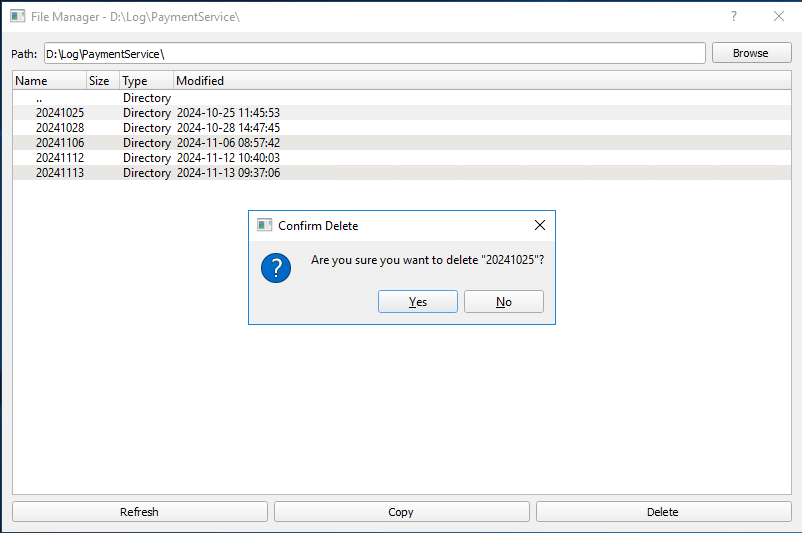
1. Khi click vào 1 item trong danh sách và ấn nút **Copy**, nếu là file thì sẽ tải về máy client ngược lại là thư mục thì thông báo là “Cannot copy directories” bằng function download\_file.
2. Nếu là file, chọn đường dẫn để lưu lại ở Client.
3. Xử lý phía Client:Khi gọi hàm download\_file thì sẽ lấy thông tin tên và loại của dòng được chọn, nếu là file thì tạo full path, và gửi command download/fullpath lên server, Server xử lý và trả về trong hàm send\_file\_command với tên file là temp\_file(quá trình này sẽ chạy ngầm). Sau đó sẽ copy từ temp\_file vào cái savePath mà người dùng chọn từ FileDialog

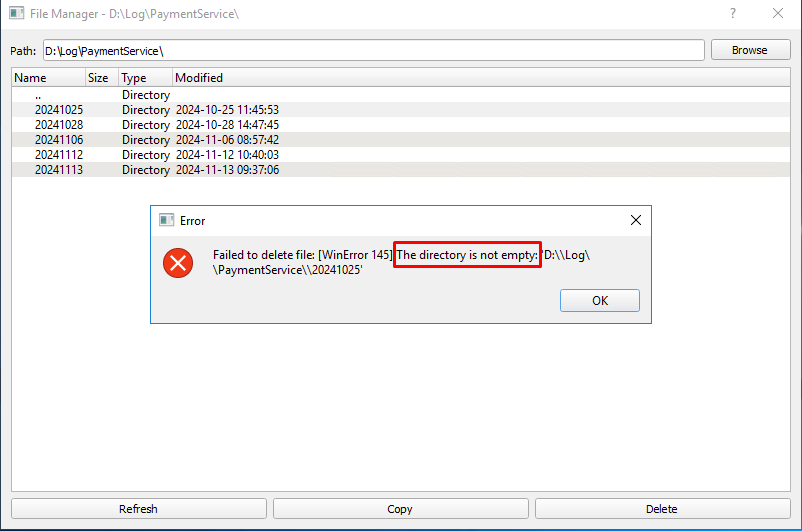


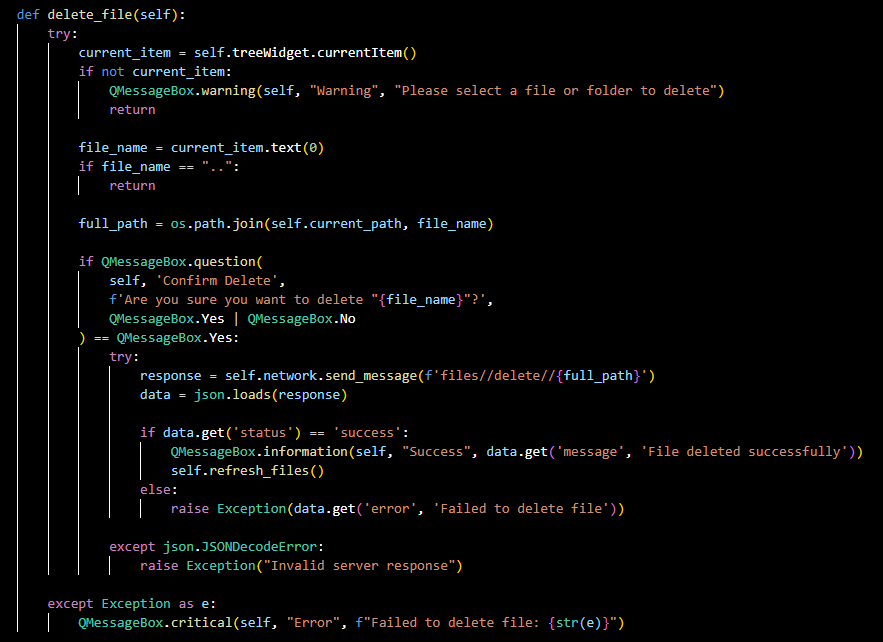
1. Xử lý phía Server:

#### Chức Năng 3. Xóa (delete) file ở Server

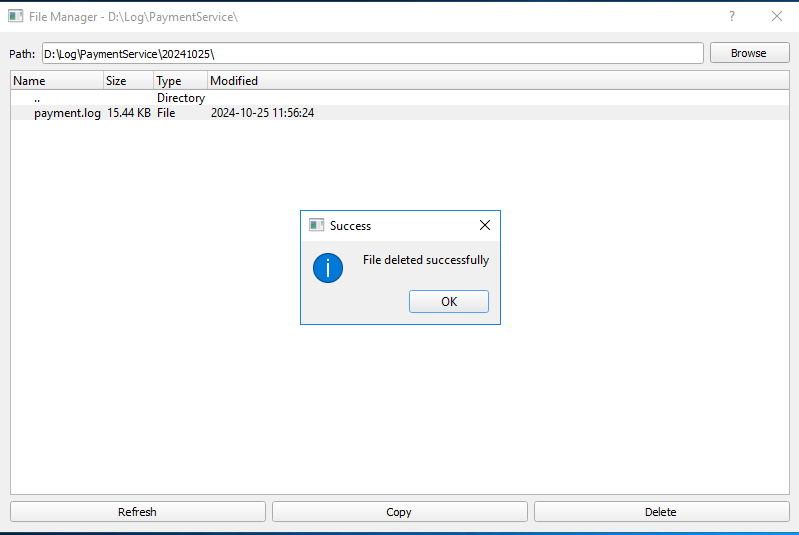
1. Khi click vào 1 item trong danh sách và ấn nút **Delete** thì sẽ hiện popup xác nhận xóa.



* Khi người dùng xác nhận, chương trình kiểm tra nếu folder tồn tại folder hoặc file con sẽ hiển thị lỗi bằng function delete\_file.

1. Xử lý phía Client: khi call hàm **delete\_file** sẽ check nếu là thư mục cha hoặc k tồn tại thì return, nếu là file thì tạo ra fullpath và show con confirm box cho người dùng xác nhận, nếu Yes thì gừi command **delete/fulpath** lên server. Nếu server trả về thành công thì call hàm **refresh\_files** để cập nhật lại danh sách.  
   
2. Xử lý phía Server:



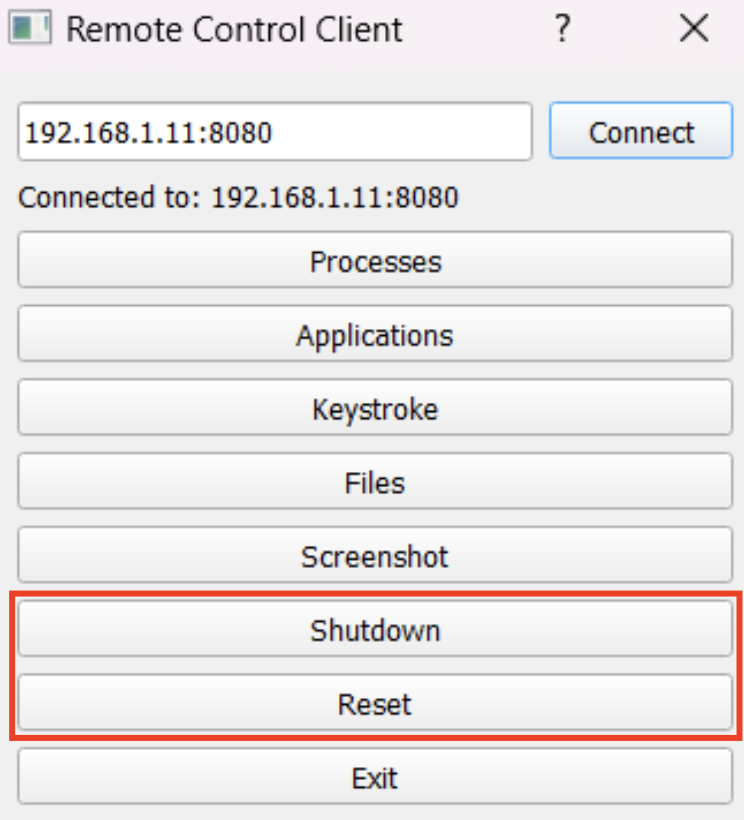
1. Nếu thoả hết điều kiện thì thực hiện xóa file và hiển thị kết quả:

## 5. Chức Năng Shutdown và Reset

### a. Tổng Quan Về Cơ Chế

| **Hàm** | **Thiết bị** | **Chức năng** |
| --- | --- | --- |
| reset() | client | Thực hiện các nhiệm vụ gửi lệnh reset cho server. |
| shutdown() | client | Thực hiện các nhiệm vụ gửi lệnh shutdown cho server. |
| QMessageBox.question | client | Tạo bảng thông báo để xác nhận lệnh. |
| send\_message() | client | Chọn socket phù hợp với lệnh của người dùng |
| \_send\_basic\_message() | client | Gửi nội dung lệnh reset (hoặc shutdown) cho socket. |
| settimeout() | client | Thiết lập thời gian chờ cho socket (được cấu hình là 300ms). |
| sendall() | client | Gửi lệnh reset (hoặc shutdown) vào shocket. |
| time.time() | client | Lấy giá trị thời gian hiện tại. |
| handle\_connection() | server | Điều hành và thực hiện các chức năng nhân và thực thi từ socket. |
| recv() | server | Hàm của socket để nhận thông tin từ socket đồng thời giải tín hiệu từ bit thành byte. |
| decode().strip() | server | chuyển mã từ byte sang bảng chữ cái và cắt bỏ những khoảng trống ở đầu và cuối chuỗi để tạo 1 câu lệnh hoàn chỉnh. |
| os.system() | server | Truy cập vào hệ thống mà server đang hoạt động thông qua thư viện os. |

### b. Giao Diện



### c. Cách Hoạt Động Chính

#### Chức Năng 1. Reset Server

1. Nhấn nút **Reset** ở màn hình chính.
2. Xử lý phía Client:

* Cấu hình phím chọn “reset” ở client sẽ gọi đến hàm reset khi người dùng nhấn vào phím “reset”A screen shot of a computer code

  Description automatically generated
* Hàm reset sau đó sẽ được thực hiện:A screenshot of a computer error message

  Description automatically generated
* Sau khi kiểm tra sự kết nối giữa client và server vẫn sẵn sàng, hàm QmessageBox.question() gọi ra bảng thông báo để xác nhận rằng người dùng muốn thực hiện chức năng reset.

A screenshot of a computer

Description automatically generated

* Khi người dùng chọn “Yes”, khi này đối tượng network sẽ thực hiện hàm send\_message() với thông điệp ‘reset’ đính kèm.A screenshot of a computer program

  Description automatically generated
* Lúc này hàm send\_message() kiểm tra thông điệp ‘reset’ với các điều kiện được thiết lập sẵn và hàm \_send\_basic\_message() cùng với message ‘reset’.A screenshot of a computer program

  Description automatically generated
* Sau khi kiểm tra , tự động thiết lập thời gian chờ timeout cho basic\_socket. Lúc này sẽ kiểm tra xem thời gian chờ timeout có bị vượt quá hay không, nếu có thì đưa ra ngoại lệ socket.timeoutA white background with black and green text

  Description automatically generated
* Nếu có socket.timeout thì kiểm tra kết quả nhận từ server, nếu không, sẽ ném lại ngoại lệ timeout.
* Nhận kết quả từ server, nếu thất bại thì hiện mã lỗi.

1. Xử lý phía Server:

A screen shot of a computer code

Description automatically generated

* Ở phía Server, dữ liệu từ socket sau khi nhận từ client sẽ được gắn vào biến data. Và tiếp tục đc giải mã đồng thời cắt bỏ những khoảng trắng ở đầu và cuối chuỗi thông qua hàm strip() để có được dữ liệu rõ ràng lưu vào biến command. A screen shot of a computer code

  Description automatically generated
* Ứng với điều kiện ‘reset’ phù hợp với command sẽ thực hiện reset. Với sự hỗ trợ của hàm system trong thư viện os, ta có thể truy cập chức năng reset hoặc shutdown của máy tính.
* Nếu command không trùng với bất cứ điều kiện nào thì server sẽ gửi lỗi 404 về cho client.

#### Chức Năng 2. Shutdown Server

Giống như reset, nhưng phía Client sẽ gửi thông điệp ‘shutdown’ cho Server và hoạt động còn lại tuần tự như chức năng ‘reset’.